Τεχνολογικό Εκπαιδευτικό Ίδρυμα Κρήτης
Σχολή Τεχνολογικών Εφαρμογών
Τμήμα Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Υλοποίηση Αλγορίθμων σε Hardware για Επιτάχυνση της Λειτουργίας τους

Γεωργία-Ειρήνη Τρούλη (Α.Μ:3923)

Επιβλέπων καθηγητής: Δρ. Γεώργιος Κορνάρος

Εξεταστική Επιτροπή:

Δρ. Κορνάρος Γεώργιος

Δρ. Δημήτριος Στρατάκης

Δρ. Βιδάκης Νικόλαος

Ηράκλειο – Σεπτέμβριος 2017

Σύνοψη

Στόχος της πτυχιακής εργασίας είναι η βελτιστοποίηση του χρόνου εκτέλεσης υπολογιστικά χρονοβόρων αλγορίθμων με τη χρήση αυτοματοποιημένων εργαλείων σχεδίασης ενσωματωμένων συστημάτων.

Επιδιώκεται η ελαχιστοποίηση του συνολικού χρόνου εκτέλεσης ενσωματωμένων εφαρμογών που χρησιμοποιούν αντίστοιχους αλγορίθμους. Για την επιτάχυνση των αλγορίθμων χρησιμοποιείται η τεχνική της υλοποίησης των τμημάτων του αλγόριθμου, που είναι υπολογιστικά χρονοβόρα, σε ψηφιακά υποσυστήματα ειδικού σκοπού που ονομάζονται επιταχυντές (accelerators).

Συγκεκριμένα,στην παρούσα εργασία πραγματοποιείται βελτιστοποίηση της απόδοσης του αλγορίθμου k-NN και η υλοποίησή του με τη χρήση της σουίτας λογισμικού Xilinx SDSoC, η οποία προσφέρει ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού για Hardware. Επιπρόσθετα, γίνεται η χρήση ενός ενσωματωμένου συστήματος (Zedboard), το οποίο είναι βασισμένο σε αρχιτεκτονική ARM, για την εφαρμογή του αλγορίθμου k-NN.

Abstract

The objective of this thesis is the optimization of time consuming computational algorithms by using automated design tools for embedded systems.

It is pursued to minimize the total execution time of embedded applications that use such algorithms. The speed-up of the algorithms is achieved by using techniques where the computationally time consuming parts of the algorithm are implemented in special purpose digital subsystems that are distinguished as accelerators.

Specifically, in the present thesis the optimization and implementation of the k-NN algorithm is accomplished by using the Xilinx SDSoc software suite that offers a complete software development environment for Hardware. In addition, an ARM-based embedded system (Zedboard) is used to apply the k-NN algorithm.

Περιεχόμενα

Ευρετήριο Εικόνων

Ευρετήριο Πινάκων

2. Εισαγωγή στη Μηχανική Μάθηση

3.ΕισαγωγήσταΕνσωματωμέναΣυστήματα

5.Εντολές για Επιτάχυνση του k-NN Αλγορίθμου σε Hardware

6. Υλοποίηση του k-NN Αλγορίθμου σε Hardrware

1. Εισαγωγή

Όταν πρόκειται για την καλύτερη απόδοση ενός αλγόριθμου και πλέον το Software δεν παρέχει περαιτέρω βελτιστοποίηση, υπάρχει η λύση της επιτάχυνσης μέσω Hardware.

Το FPGA παρέχει όλα όσα χρειάζονται για την επιτάχυνση ενός αλγόριθμού. Στην προσπάθεια να επιτευχθεί καλύτερη απόδοση στον κώδικα απαιτείται βελτιστοποίηση στους αλγορίθμους, εφαρμόζοντας για παράδειγμα ξετύλιγμα βρόγχων, παραλληλοποίηση βρόγχων, παραλληλοποίηση κλήσεων μιας συνάρτησης κ.α. και όλα αυτά χρησιμοποιώντας την προγραμματιζόμενη λογική για την επιτάχυνση Hardware.

Η χαμηλού κόστους προγραμματιζόμενη λογική που υπάρχει στα ενσωματωμένα συστήματα παρέχει μια άλλη επιλογή για υψηλότερες επιδόσεις χωρίς να χρειάζεται η χρήση αλλαγής επεξεργαστών ή αρχιτεκτονικών. Με την προγραμματιζόμενη λογική, οι συναρτήσεις μετατρέπονται από compute intensive σε συναρτήσεις επιτάχυνσης Hardware.

Ο όρος επιτάχυνση σε Hardware σημαίνει την αντικατάσταση ενός Software αλγορίθμου με μια ενότητα Hardware προκειμένου να επωφεληθεί από την εσωτερική του ταχύτητά (του Hardware).

Η βελτίωση του χρόνου εκτέλεσης μπορεί να φθάσει το 10.000% (ή 100 φορές) ανάλογα με τον αλγόριθμο. Το Hardware είναι πολύ πιο γρήγορο κατά την εκτέλεση εργασιών συμπεριλαμβανομένης της εκτέλεσης σύνθετων μαθηματικών λειτουργιών, μεταφοράς δεδομένων από το ένα μέρος στο άλλο και την εκτέλεση της ίδιας της συνάρτησης πολλές φορές.

Με τη χρήση του field-programmable gate arrays (FPGAs) υπάρχει η δυνατότητα επιλογής όσον αφορά ποια μέρη του κώδικα θα εφαρμοστούν σε Hardware και ποια όχι. Με τα εργαλεία διαχείρισης του, δημιουργείται ο κώδικας HDL για τη «bus logic» και «interrupt logic» καθώς και προσαρμόζουν τις βιβλιοθήκες λογισμικού και περιλαμβάνουν αρχεία ανάλογα με τη διαμόρφωση του συστήματος.

Για τη βελτιστοποίηση του χρόνου εκτέλεσης στη συνέχεια θα εισαχθούν κάποιες ειδικές εντολές οι οποίες σχετίζονται με τη μετακίνηση των δεδομένων από την εξωτερική μνήμη DDR προς την τοπική μνήμη της προγραμματιζόμενης λογικής καθώς και όσο το δυνατόν μεγαλύτερη παραλληλοποίηση στις λειτουργίες του αλγορίθμου , αυτό αναπτύσσεται στο πέμπτο κεφαλαίο αυτής της εργασίας.

Ο αλγόριθμος που θα χρησιμοποιηθεί είναι ο k-NN από τους πιο απλούς αλγορίθμους μηχανικής μάθησης. Η παράμετρος k της τεχνικής , ορίζει τον αριθμό των κοντινότερων γειτόνων που έχουν λόγο στην κατηγοριοποίηση. Η υπό εξέταση πλειάδα τοποθετείται στην κατηγορία στην οποία ανήκουν οι περισσότεροι από τους k πλησιέστερους γείτονες . Το πόσοι κοντινότεροι γείτονες θα ληφθούν υπόψιν για να γίνει η κατηγοριοποίηση είναι ένα σημαντικό ζήτημα αφού η ακρίβεια της κατηγοριοποίησης εξαρτάται από αυτόν τον αριθμό. Έτσι, είναι απαραίτητο να βρεθεί ο αριθμός k που δίνει την καλύτερη κατηγοριοποίηση .

Σε αυτή την εργασία εκτός από **το ζήτημα της επιτάχυνσης** του αλγορίθμου πρέπει να λυθεί και το **πρόβλημα διαχείρισης δεδομένων** διότι ο συγκεκριμένος αλγόριθμός απαιτεί μεγάλο αποθηκευτικό χώρο. Θα υπάρξει σταδιακή αύξηση των δεδομένων (από λίγα έως πολλά) ούτως ώστε να μετρηθούν οι διαφορές ταχυτήτων ανάλογα τον όγκο των δεδομένων.

Στόχος αυτής της εργασίας είναι να μελετηθεί η απόδοση της κατηγοριοποίησης της τεχνικής των k-κοντινότερων γειτόνων από την πλευρά του hardware που αφορά την ταχύτητα εκτέλεσης. Σε όλη την μελέτη , η αναζήτηση των κοντινότερων γειτόνων πραγματοποιείται χρησιμοποιώντας ως μέτρο την απόσταση.

Τα εργαλεία που θα χρησιμοποιήθουν για τη βελτιστοποίηση του χρόνου εκτέλεσης του k-NN αλγορίθμου είναι ένα ενσωματωμένο συστήμα το οποίο είναι βασισμένο σε αρχιτεκτονική ARM. Εκτός από το την πλακέτα όμως θα γίνει χρήση του προγράμματος SDSoc το οποίο είναι σουίτα λογισμικού η οποία προσφέρει ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού για Hardware.

Στάδια Πτυχιακής Εργασίας

Η εκπόνηση της πτυχιακής διαιρείται στα παρακάτω κεφάλαια :

- Το δευτερο κεφάλαιο «Εισαγωγή στη Μηχανική Μάθηση» περιγράφει τον αλγόριθμο k-NN, τη λειτουργία του, τις δυνατότητες του σε τι είδους εφαρμογές μπορεί να χρησιμοποιηθεί.

- Για τον αλγόριθμο k-NN θα γίνει χρήση ενός ενσωματωμένου συστήματος αυτό θα απασχολήσει το τρίτο κεφάλαιο«Εισαγωγή στα Ενσωματωμένα Συστήματα».

- Το περιβάλλον ανάπτυξης λογισμικού για Hardware που θα χρησιμοποιηθεί αναλύεται στο κεφάλαιο τέσσερα«Εισαγωγή στο SDX»

- Οι εντολές με τις οποίες ο αλγόριθμος k-NN θα επιτύχυνθει την καθώς και η επεξήγηση τους θα απασχολήσουν στο κεφάλαιο πέντε «Εντολές για Επιτάχυνση του k-NN Αλγορίθμου σε Hardware»

- Στο κεφάλαιο έξι «Υλοποίηση του k-NN Αλγορίθμου σε Hardrware» αναλύονται διάφορα πραδείγματα και μετρήσεις μέσω των οποίων επιτυγχάνεται η προσέγγιση της καλύτερης απόδοσης ανά περίπτωση.

- Στο έβδομο κεφάλαιο « Αποτελέσματα» συγκεντρώνονται οι καλύτερες μετρήσεις ανάλογα με την περίπτωση και περιγράφονται τα συμπεράσματα καθώς και οι μεθόδοι προσέγγισης του k-NN αλγορίθμου

2. Εισαγωγή στη Μηχανική Μάθηση

Η «**μηχανική μάθηση**» [12] αποτελεί έναν ξεχωριστό κλάδο της επιστήμης υπολογιστών που αναπτύχθηκε μέσα από τη μελέτη της αναγνώρισης προτύπων και της υπολογιστικής θεωρίας μάθησης στην τεχνητή νοημοσύνη. Ημηχανική μάθησηείναι ένας τύπος τεχνητής νοημοσύνης (Artificial Intelligence) που παρέχει στους υπολογιστές τη δυνατότητα να μάθουν χωρίς να έχουν προγραμματιστεί «ρητά» (το πρόγραμμα έχει τη δυνατότητα να εξάγει τη γνώση από μόνο του). Η μηχανική μάθηση επικεντρώνεται στην ανάπτυξη προγραμμάτων ηλεκτρονικών υπολογιστών που δύναται να αλλάξουν όταν εκτίθενται σε νέα δεδομένα[13].

Στη μηχανική μάθηση εξάγονται δεδομένα για κατανόηση από τον άνθρωπο τα οποία τα χρησιμοποιούνται για την ανίχνευση των πρότυπων στα δεδομένα και για την προσαρμογή τους ανάλογα με τις ενέργειες του προγράμματος. Οι αλγόριθμοι μηχανικής μάθησης κατηγοριοποιούνται συχνά είτε υπό επίβλεψη (supervised) είτε χωρίς επίβλεψη (unsupervised). Οι επιτηρούμενοι αλγόριθμοι (**Supervised algorithms**) μπορούν να εφαρμόσουν όσα έχουν μάθει στο παρελθόν σε νέα δεδομένα. Οι μη επιτηρούμενοι (**Unsupervised algorithms**) αλγόριθμοι μπορούν να αντλήσουν συμπεράσματα από τα σύνολα δεδομένων.

Εφαρμογές οι οποίες χρησιμοποιούν τη μηχανική μάθηση είναι τα φίλτρα spam (spam filtering), η οπτική αναγνώριση (OCR), οι μηχανές αναζήτησης και η υπολογιστική όραση.

Υπάρχουν διάφοροι αλγόριθμοι μηχανικής μάθησης (**Εικόνα 1:** Αλγόριθμοι Μηχανικής Μάθησης) όπου ο κάθε ένας χρησιμοποιείται για διαφορετικό σκοπό[12].

Εικόνα 1:Αλγόριθμοι Μηχανικής Μάθησης

2.1 Αλγόριθμος Ταξινόμησης k-Nearest Neighbor (k-NN)

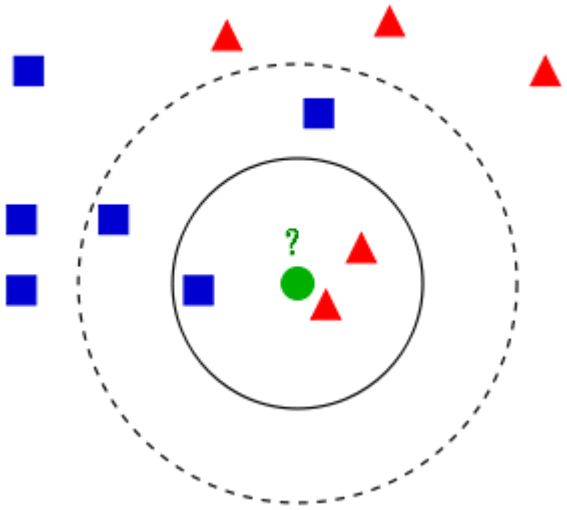
Ο αλγόριθμος εκτίμησης k-πλησιέστερων γειτόνων είναι ένας αλγόριθμος **μηχανικής μάθησης.** Χρησιμοποιείται για την εύρεση και την εξέταση των πλησιέστερων γειτόνων ενός συγκεκριμένου σημείου μεταξύ ενός συνόλου μη δομημένων σημείων .

Ο k-NN αλγόριθμος λειτουργεί ως εξής :

Για ένα σύνολο S των n σημείων αναφοράς σε ένα n-δισδιάστατο χώρο και ένα μη ταξινομημένο σημείο q, ο αλγόριθμος k-κοντινότερος γείτονας βρίσκει τα k κοντινότερα σημεία στο S ως προς το σημείο q (αυτό απεικονίζεται στην **Εικόνα 2:** Ταξινομητής πλησιέστερου γείτονα (k-NN)). Για n=5 (κόκκινα τριγωνάκια) και n=5(μπλε τετραγωνάκια). Το πράσινο κυκλάκι αναπαριστά το μη ταξινομημένο σημείο ενώ τα μπλε και κόκκινα δείγματα αναπαριστούν τα σημεία που ειναι κατηγοριοποιημένα (ομάδα A και ομάδα B). Στην παρακάτω εικόνα (**Εικόνα 2:** Ταξινομητής πλησιέστερου γείτονα (k-NN)) υπάρχουν δύο ομάδες (ομάδα A και ομάδα B). Με βάση τους k γείτονες το σημείο κυκλάκι θα καθοριστεί σε ποια ομάδα θα κατηγοριοποιηθεί. Λαμβάνοντας υπόψιν τη περίπτωση 1 όπου k=3 το μη ταξινομημένο σημείο θα ταξινομηθεί στην ομάδα A (εφόσον 2>1). Αν όμως θέσουμε k=5 θα ταξινομηθεί στην ομάδα B (εφόσον 3>2).

Ομάδα Α (κόκκινα ▲)
Ομάδα Β (μπλε ■)

Περίπτωση 1: k=3
Περίπτωση 2: k=5



Εικόνα 2: Ταξινομητής πλησιέστερου γείτονα (k-NN)

Ο αλγόριθμος k-NN είναι μια από τις πιο γνωστές μεθόδους **κατηγοριοποίησης**[1]. Στη κατηγοριοποίησηένα δείγμα ταξινομείται με βάση τη πλειοψηφία των υποψήφιων γειτόνων (k), με το δείγμα να εντάσσεται στη κατηγορία όπου καταλαμβάνει τη πλειοψηφία ανάμεσα στους k κοντινότερους γείτονες (k είναι ένας θετικός ακέραιος ,συνήθως μικρός).Αν το k=1, τότε το δείγμα απλά θα ανήκει στη κατηγορία του κοντινότερου γείτονα.

Ο k-NN χρησιμοποιείται στο ένα τρίτο των προβλημάτων ταξινόμησης, εκτός όμως από τη ταξινόμηση μπορεί να χρησιμοποιηθεί και για **παλινδρόμηση** [1][8] όπου η έξοδος αποτελεί την κύρια τιμή του δείγματος. Αυτή η μέθοδος μπορεί να χρησιμοποιηθεί σε περιπτώσεις όπου τα χαρακτηριστικά των δεδομένων είναι συνεχείς και όχι διακριτές μεταβλητές. Η κατηγορία όπου εν τέλει καταχωρείται το μη ταξινομημένο σημείο υπολογίζεται βάσει του μέσου όρου των χαρακτηριστικών των πλησιέστερων γειτόνων του.

Ο τρόπος κατάταξης των νέων σημείων υλοποιείται με βάση κάποιο μέτρο ομοιότητας (πχ. απόσταση, χρώμα, σχήμα ή ομοιομορφία). Υπάρχουν διάφορες συναρτήσεις με τις οποίες λαμβάνονται οι κοντινότεροι γείτονες μεταξύ των σημείων (**Πίνακας 1:** Συναρτήσεις απόστασης)[9]. Η πιο συνήθης χρησιμοποιούμενη συνάρτηση μέτρησης απόστασης είναι η ευκλείδεια συνάρτηση .

Euclidean	$d(x,y) = \sqrt{\sum (x_i - y_i)^2}$
Squared Euclidean	$d(x,y) = \sum (x_i - y_i)^2$
Manhattan	$d(x,y) = \sum \lvert (x_i - y_i) \rvert$
Canberra	$d(x,y) = \sum \frac{\lvert x_i - y_i \rvert}{\lvert x_i + y_i \rvert}$
Chebychev	$d(x,y) = \max(\lvert x_i - y_i \rvert)$
Bray Curtis	$d(x,y) = \frac{\sum \lvert x_i - y_i \rvert}{\sum x_i + y_i}$
Cosine Correlation	$d(x,y) = \frac{\sum (x_i y_i)}{\sqrt{\sum (x_i)^2 \sum (y_i)^2}}$
Pearson Correlation	$d(x,y) = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (y_i - \overline{y})^2} \sqrt{\sum (y_i - \overline{y})^2}}$
Uncentered Peason Correlation	$d(x,y) = \frac{\sum x_i y_i}{\sqrt{\sum (y_i - \overline{y})^2} \sqrt{\sum (y_i - \overline{y})^2}}$
Euclidean Nullweighted	Same as Euclidean, but only the indexes where both x and y have a value (not NULL) are used, and the result is weighted by the number of values calculated. Nulls must be replaced by the missing value calculator (in dataloader).

Πίνακας 1: Συναρτήσεις απόστασης

Τα παραπάνω μέτρα απόστασης είναι για μονοδιάστατο χώρο και ισχύουν μόνο για συνεχείς μεταβλητές. Στην περίπτωση διακριτών ή απόλυτων μεταβλητών πρέπει να χρησιμοποιηθεί η απόσταση Hamming (**Εικόνα 3:** Απόσταση Hamming), η οποία είναι ένα μέτρο του αριθμού των περιπτώσεων στις οποίες τα αντίστοιχα σύμβολα είναι διαφορετικά σε δύο συμβολοσειρές ίσου μήκους (π.χ. το ζήτημα τυποποίησης των αριθμητικών μεταβλητών μεταξύ 0 και 1 όταν υπάρχει ένα σύνολο αριθμητικών και απόλυτων μεταβλητών στο σύνολο δεδομένων)[9].

$$D_H = \sum_{i=1}^{k} \lvert x_i - y_i \rvert$$

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

Εικόνα 3:Απόσταση Hamming

Ο αλγόριθμος k-NN χρησιμοποιεί **μη παραμετρική τεχνική**[3](μη παραμετρική τεχνική, σημαίνει ότι δεν γνωρίζει την υποκείμενη κατανομή δεδομένων). Αυτό είναι πολύ χρήσιμο, όπως στον πραγματικό κόσμο, τα περισσότερα από τα πρακτικά δεδομένα δεν υπακούουν σε γνωστές τυπικές θεωρητικές κατανομές (π.χ. Γκαουσιανά μοντέλα* κ.α.). Σε αυτές τις περιπτώσεις οι μη παραμετρικοί αλγόριθμοι όπως ο k-NN βοηθούν.

Ο k-NN επίσης είναι ένας «**instance-basedlearning**» ή«**memory-based learning**»ή«**lazy learning**» [4] αλγόριθμος. Αυτό σημαίνει ότι δεν χρησιμοποιεί τα σημεία δεδομένων εκπαίδευσης για να κάνει οποιαδήποτε γενίκευση. Εν ολίγοις ότι η φάση της εκπαίδευσης είναι αρκετά γρήγορη. Η έλλειψη γενίκευσης σημαίνει ότι το k-NN διατηρεί όλα τα δεδομένα εκπαίδευσης. Πιο συγκεκριμένα, όλα τα δεδομένα εκπαίδευσης είναι απαραίτητα κατά τη φάση της δοκιμής. Οι περισσότεροι από τους lazy αλγόριθμους ειδικά ο k-NN λαμβάνουν απόφαση με βάση ολόκληρο το σύνολο δεδομένων εκπαίδευσης .

Ο αλγόριθμος k-NN χρησιμοποιείται σε διάφορους τομείς και εφαρμογές όπως είναι η αναγνώριση προτύπων και η στατιστική από τις αρχές της δεκαετίας του 1970. Επίσης χρησιμοποιείται στη μηχανική οραση, στη θεωρία κωδικοποίησης και στη μηχανική μάθηση[2] .

Ο αλγόριθμος k-NN είναι από πιο απλούς αλγορίθμους **μηχανικής μάθησης** και ανήκει στους 10 κορυφαίους αλγόριθμους εξόρυξης δεδομένων.

Ο k-NN χρησιμοποιείται συχνά σε εφαρμογές αναζήτησης όπου αναζητούνται «παρόμοια» αντικείμενα δηλαδή, όταν το καθήκον του είναι της μορφής «βρες αντικείμενα παρόμοια με ένα συγκεκριμένο». Σε αυτή τη περίπτωση ο k-NN είναι ο πιο κατάλληλος αλγόριθμος. Πολύ συχνή είναι η χρήσης της αναζήτησης k-NN πίσω από τα «**συστήματα Recommender»**[10]. Εάν για παράδειγμα ένας χρήστης αναζητά ένα συγκεκριμένο αντικείμενο και στόχος είναι η πρόταση παρόμοιων αντικειμένων .Για να βρεθούν παρόμοια αντικείμενα, συγκρίνεται το σύνολο των χρηστών στους οποίους αρέσει κάθε αντικείμενο, αν ένα παρόμοιο σύνολο χρηστών στους οποίους αρέσουν δύο διαφορετικά αντικείμενα, τότε τα ίδια τα αντικείμενα είναι μάλλον παρόμοια!

Αυτό ισχύει για σύσταση προϊόντων, συστηνόμενη κατανάλωση μέσω των μέσων ενημέρωσης ή ακόμη και τη «συστηνόμενη» προβολή διαφημίσεων σε ένα χρήστη!

Ένα άλλο είδος εφαρμογών πίσω από το οποίο βρίσκεται ο k-NN είναι η αναζήτηση σημασιολογικά παρόμοιων έγγραφα (δηλ. Έγγραφα που περιέχουν παρόμοια θέματα), αυτό αναφέρεται ως «**Concept Search**»[10]. Το «Concept Search» είναι ένα χαρακτηριστικό που θα βρείτε σε πολλά πακέτα λογισμικού «electronic discovery», τα οποία χρησιμοποιούνται για να βοηθήσουν τις εταιρείες να βρουν όλα τα μηνύματα ηλεκτρονικού ταχυδρομείου, τα συμβόλαια κ.λπ. που σχετίζονται για παράδειγμα με μια δίκη.

Μια εφαρμογή πίσω από την οποία δουλεύει ο k-NN είναι η «**Hi-TechFacialRecognition**» [11] η οποία αναπτύσσει λογισμικό ασφαλείας, που ειδικεύεται στην αναγνώριση ενός πλήθους ατόμων σε πραγματικό χρόνο μέσω κάμερας IP. Χρησιμοποιεί αλγόριθμους «deep learning» για τη δημιουργία χαρακτήρων που αντιπροσωπεύουν τα πρόσωπα των ανθρώπων. Στη συνέχεια χρησιμοποιείται ο k-NN για να εντοπιστεί ένα άτομο, συγκρίνοντας το πρόσωπο του με τη λίστα παρακολούθησής τους. Ο k-NN είναι μια αρκετά καλή επιλογή και θα ήταν ανέφικτο να εκπαιδευτεί ένας ξεχωριστό ταξινομητή για κάθε άτομο στη λίστα παρακολούθησης.

Ο ταξινομητής k-NN μπορεί μέχρι και **αναγνώριση χειρόγραφου ψηφίου** να κάνει[6], αλλά απαιτεί αρκετά μεγάλη μνήμη και είναι πολύ πιο αργό κατά την ταξινόμηση. Παρόλα αυτά, η απλότητα του αλγορίθμου και τα χαρακτηριστικά γρήγορου «training» καθιστούν τον ταξινομητή k-NN περιζήτητο

σε διεργασίες ταξινόμησης που υποστηρίζονται από Hardware.

Ο k-NN αλγόριθμος ίσως τρέχει πιο αργά και έχει χαμηλότερη ακρίβεια σε σχέση με άλλους αλγορίθμους μηχανικής μάθησης , αλλά έχει κάποιες καλές πρακτικές ιδιότητες. Είναι εύκολο να εκπαιδεύσει (επειδή στην ουσία δεν εκπαιδεύει), εύκολο στη χρήση, και είναι εύκολο για την κατανόηση των αποτελεσμάτων. Γι' αυτό το λόγο η ταξινόμηση k-NN χρησιμοποιείται περισσότερο σε σχέση με άλλους αλγορίθμους ταξινόμησης.

Στον αλγόριθμο υπάρχει μια ελάχιστη φάση συγκρότησης αλλά μια δαπανηρή φάση δοκιμών. Το κόστος μπορεί να είναι μεγάλο από την άποψη του χρόνου όσο και της μνήμης. Θα χρειαστεί αρκετό χρόνο, καθώς όλα τα σημεία δεδομένων ενδέχεται να έχουν νόημα στην απόφαση. **Απαιτείται περισσότερη μνήμη** καθώς πρέπει να αποθηκεύσουμε όλα τα δεδομένα εκπαίδευσης .

Επίσης πολύ σημαντικό ρόλο στον αλγόριθμο όσον αφορά τη λειτουργικότητα του έχει αν θα κάνει «online» ή «batch» μάθηση.

Ο όρος «**online**» σημαίνει ότι μετά το training του κάθε δείγματος το πλέον ταξινομημένο δείγμα λαμβάνεται υπόψην για σύνολο του training . Αν συμπεριληφθεί αυτός ο τρόπος στον

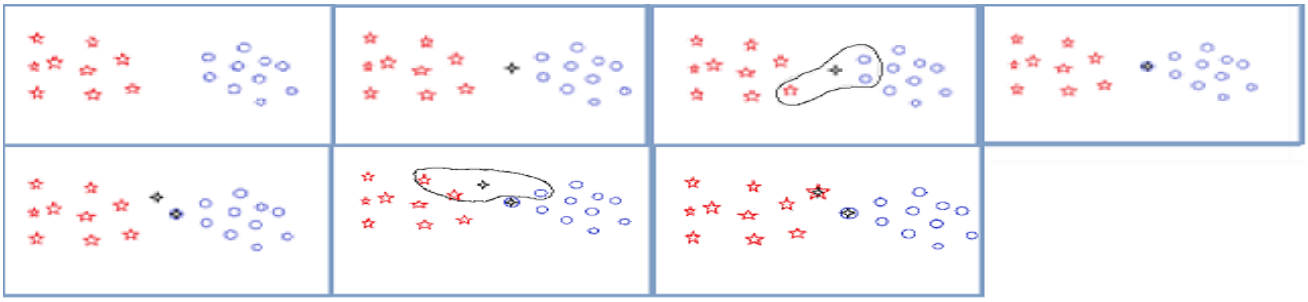
k-NN αλγόριθμο έχει ως αποτέλεσμα την υπερφόρτωση και ως εκ τούτου προκαλεί σύγχυση στον τομέα της μηχανικής μάθησης. Ο online k-NN χρησιμοποιείται σε πολλές περιπτώσεις όπως είναι η αναγνώριση εικόνας . Για την online μάθηση, υπάρχουν (τυπικά) περισσότερα δεδομένα που πρέπει να συμπεριληφθούν, αλλά υπάρχουν περιορισμοί χρόνου. Κάτι αρνητικό που θα μπορούσε να συμβεί στην online μάθηση είναι να επηρεάσει τις κατατάξεις και τις κατηγορίες αλλάζοντας τις διαχρονικά. Το πρόβλημα απεικονίζεται αναλυτικότερα παρακάτω (**Εικόνα 4:**Online μάθηση).

Εικόνα 4: Online μάθηση

Ο όρος «**batch**» σημαίνει πως υπάρχει ένα στατικό σετ δεδομένων στον αλγόριθμο το οποίο δεν

αλλάζει παρά μόνο μετά το τέλος εκπαίδευσης ολόκληρου του συνόλου (**Εικόνα 5:** Batch μάθηση). Συνεπώς με αυτόν τον τρόπο ενημερώνει τις παραμέτρους του μετά το τέλος της μάθησης. Με αυτόν τον τρόπο ο αλγόριθμος είναι πιο απλός , πιο γρήγορος και πιο ασφαλής σε σχέση με την online τεχνική

Η online μάθηση πιο αργή όσον αφορά τη υλοποίηση του αλγορίθμου ,αν συγκριθεί με τη batch μάθηση. Ωστόσο, σε περιπτώσεις όπου ολόκληρο το σετ δεδομένων δεν χωράει στη μνήμη, η χρήση την online μάθησης αποτελεί αποδεκτό συμβιβασμό. Αυτό γίνεται εφόσον με την online ταξινόμηση έχω για κάθε μη ταξινομημένο δείγμα όλο και περισσότερα ταξινομημένα δείγματα.



Εικόνα 5: Batch μάθηση

Περιορισμοί του αλγορίθμου(k-NN):

- Στη περίπτωση όπου οι κατηγορίες (ομάδες) είναι ίσες με τους *k* γείτονες τότε το μη ταξινομημένο δείγμα ταξινομείται με βάση την κοντινότερη απόσταση.
- Υπάρχει το ενδεχόμενο οι *k*-γείτονες έχουν ισοψηφία αλλά αυτή τη φορά ανάμεσα σε δύο από όλες τις ομάδες. Μια λύση σε αυτή τη περίπτωση θα ήταν να βρίσκει τη μικρότερη απόσταση (ανάμεσα στις δύο ομάδες).
- Ο αριθμός των *k*-γειτόνων συνίσταται να βρίσκεται μεταξύ 3-10.

Για την υλοποίηση του αλγορίθμου θα πρέπει να προσδιοριστούν εκ των προτέρων:

1. Η παράμετρος ***k*** (ο αριθμός των πλησιέστερων σημείων του συνόλου σε σχέση με το νέο μη ταξινομημένο σημείο).
2. Ο αριθμός των δειγμάτων (ταξινομημένα και μη).
3. Ο αριθμός των διαστάσεων των δειγμάτων.
4. Ο αριθμός των κατηγοριών και οι κατηγορίες που υπάρχουν.
5. Το μέτρο με το οποίο θα υλοποιηθεί η ταξινόμηση (πχ. απόσταση, χρώμα, σχήμα κ.α).

Ο αλγόριθμος περιλαμβάνει τα ακόλουθα βήματα:

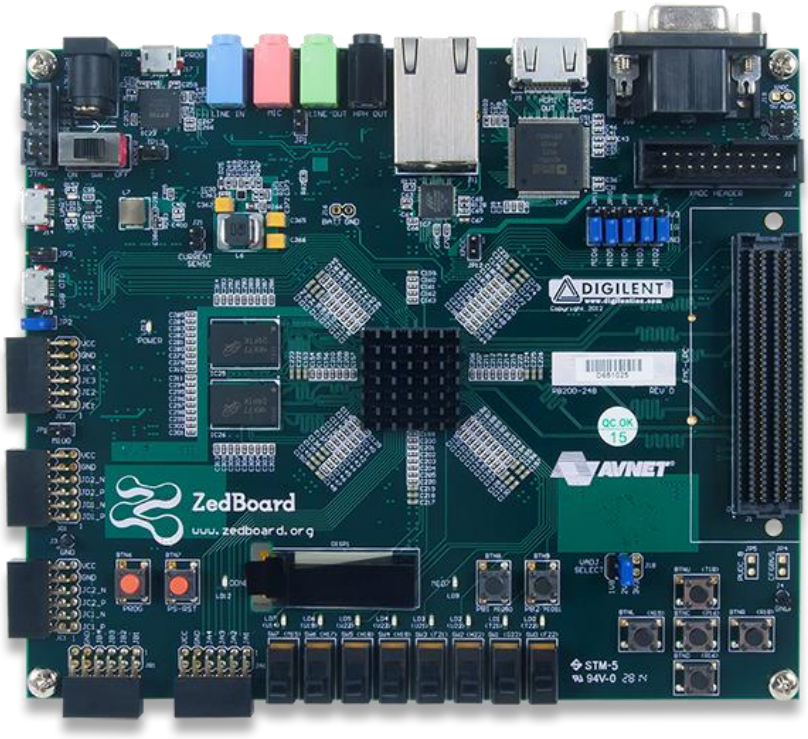
1. Υπολογίζει n διαστάσεις μεταξύ του νέου μη ταξινομημένου σημείου q και των n σημείων αναφοράς του συνόλου S. Για την εύρεση αποστάσεων χρησιμοποιήθηκε η συνάρτηση της Ευκλείδειας απόστασης η οποία δίνεται από το σύνολο δύο δισδιάστατων σημείων
   $d = \sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$.
2. Βρίσκει και ταξινομεί τις ***k*** αποστάσεις κατά αύξουσα σειρά. Επιλέγει τα ***k*** (πιο κοντινά σημεία) από το S (σύνολο) που αντιστοιχούν στις χαμηλότερες αποστάσεις της κατηγορίας ταξινομημένων αποστάσεων.

3. Έπειτα επιστρέφει το νέο σημείο ταξινομημένο με την πληροφορία της κατηγορίας στην οποία πλέον ανήκει. Αυτό υλοποιείται με βάση την κατηγορία (group) η οποία περιέχει τη πλειοψηφία των περισσότερων γειτόνων (σημεία *k*) σε σχέση με το σημείο q .

Τα πλέον ταξινομημένα δείγματα (πρώην q) μαζί με το group στο οποίο ανήκουν εκτυπώνονται .

3. Εισαγωγή στα Ενσωματωμένα Συστήματα

Η πλατφόρμα πάνω στην οποία έτρεξε ο αλγόριθμος k-NN είναι το ZedBoard (Zynq Evakuation and Development), ένα ενσωματωμένο επικοινωνιακό σύστημα. Το ZedBoard είναι μια προγραμματιζόμενη πλατφόρμα χαμηλού κόστους. Υποστηρίζει την υλοποίηση λειτουργικών συστήματα όπως Linux,Windows και Android ή με άλλο σχεδιασμό βασισμένο σε λειτουργικό σύστημα και Real Time λειτουργικό σύστημα. Το ZedBoard είναι βασισμένο σε AXI interfaces όπου παρέχει μεγάλο εύρος ζώνης και συνδέσεις με μικρή καθυστέρηση. Το ολοκληρωμένο αυτό σύστημα βασίζεται στο Xilinx XC7Z020 και είναι ένα από τα μικρότερα της οικογένειας των Zynq-7000 All Programmable SoC. Παρακάτω (**Εικόνα 6:** Πλατφόρμα ZedBoard) απεικονίζεται η πλατφόρμα ZedBoard.



Εικόνα 6: Πλατφόρμα ZedBoard

Το ZedBoard περιέχει τον επεξεργαστή ZYNQ της XILINX. Συνδυάζει διπύρινο επεξεργαστή μαζί με το fpga (Field Programmable Gate Array). Αποτελείται από τα δύο τμήματα του Zynq, το υποσύστημα επεξεργαστή (Processing System) και το υποσύστημα προγραμματιζόμενης λογικής (Programmable Logic) όπου μπορούν να λειτουργήσουν αυτόνομα ή σε συνεργασία δημιουργώντας έτσι ένα ευέλικτο σύστημα υψηλής απόδοσης[21][41], και από περιφερειακές μονάδες,

ενσωματωμένες στο σύστημα. Οι περιφερειακές μονάδες συνδέονται με τα παραπάνω τμήματα με φυσικές ή προγραμματιζόμενες διασυνδέσεις υψηλής ταχύτητας, δίνοντας στο σύστημα πολλαπλές δυνατότητες επικοινωνίας με άλλα συστήματα.

Έτσι το Zedboard και το XC7Z020 αποτελούν ιδανικές επιλογές για ανάπτυξη πολυδιάστατων συστημάτων, λόγω της δυνατότητας επικοινωνίας με ανομοιόμορφων προδιαγραφών συστήματα, αλλά ταυτόχρονα υψηλής απόδοσης, λόγω των πολλαπλών δυνατοτήτων για επεξεργασία των δεδομένων που διαχειρίζεται [39][40]. Συνδυάζει την ύπαρξη Υπολογιστικού Συστήματος με δύο επεξεργαστές ARM με την ύπαρξη επαναπρογραμματιζόμενης Λογικής.

Στην εικόνα παρακάτω (**Εικόνα 7:** Χαρακτηριστικά του ZedBoard) απεικονίζεται το επεξεργαστικό υποσύστημα το οποίο αποτελείται από :

1. Μνήμη: δυναμική (DDR3, 256 MB ενσωματωμένη μνήμη) και στατική μνήμη (SPIFlash,SDCardInterface)

2. USBs: USB-to-UART σύνδεση, λειτουργικότητα JTAG, μονάδα επικοινωνίας με τη διεπαφή USB (πχ ως είσοδος δεδομένων)

3. Οθόνη και Ήχος : HDMI Transmitter, Analog Device Audio Codec, OLED Display

4. ClockSources: 33.3333 MHz ρολόι για το Υπολογιστικό Σύστημα ενώ το Υπολογιστικό Σύστημα παράγει έως 4 ρολόγια για το Επαναπρογραμματιζόμενο μέρος της πλακέτας

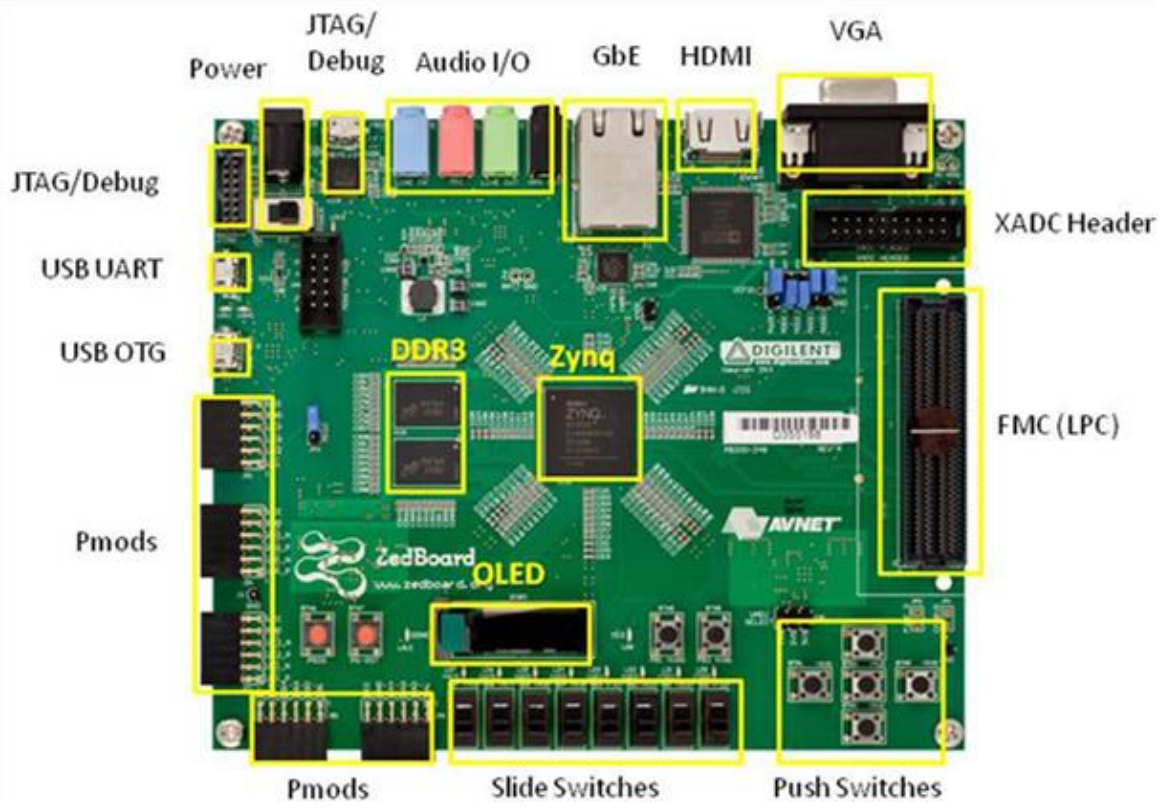
5. ResetSources: εξωτερικοί διακόπτες για επανεκκίνηση της πλακέτας και επαναπρογραμματισμό του FPGA

6. User I/O: 7 user GPIO push button, 8 user dip switches, 8 LEDs.

7. Θύρα Ethernet για σύνδεση στο διαδίκτυο

8. PS (διπύρηνος επεξεργαστή ARMCortex-A9, ρυθμιζόμενης συχνότητας μέχρι τα 667 MH) και PLI/O επεκτάσεις.

9. Υπάρχει η δυνατότητα εισαγωγής SD κάρτας στην πίσω πλευρά του board (πχ για εισαγωγή OS)



Εικόνα 7: Χαρακτηριστικά του ZedBoard

3.1 Field Programmable Gate Array (FPGA)

Το **FPGA** (Field Programmable Gate Array) είναι τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης το οποίο διαθέτει πολύ μεγάλο αριθμό τυποποιημένων πυλών και άλλων ψηφιακών λειτουργιών όπως απαριθμητές, καταχωρητές μνήμης, γεννήτριες PLL κα. Κατά τον προγραμματισμό του FPGA, ο οποίος γίνεται πάντοτε ενώ αυτό είναι τοποθετημένο στο τυποποιημένο κύκλωμα, ενεργοποιούνται οι επιθυμητές λειτουργίες και διασυνδέονται μεταξύ τους έτσι ώστε το FPGA να συμπεριφέρεται ως ολοκληρωμένο κύκλωμα με συγκεκριμένη λειτουργία. Το FPGA έχει τη δυνατότητα να επαναγραπρογραμματίζεται και να αλλάζει σύμφωνα με τη τροποποίηση του λογισμικού [15].

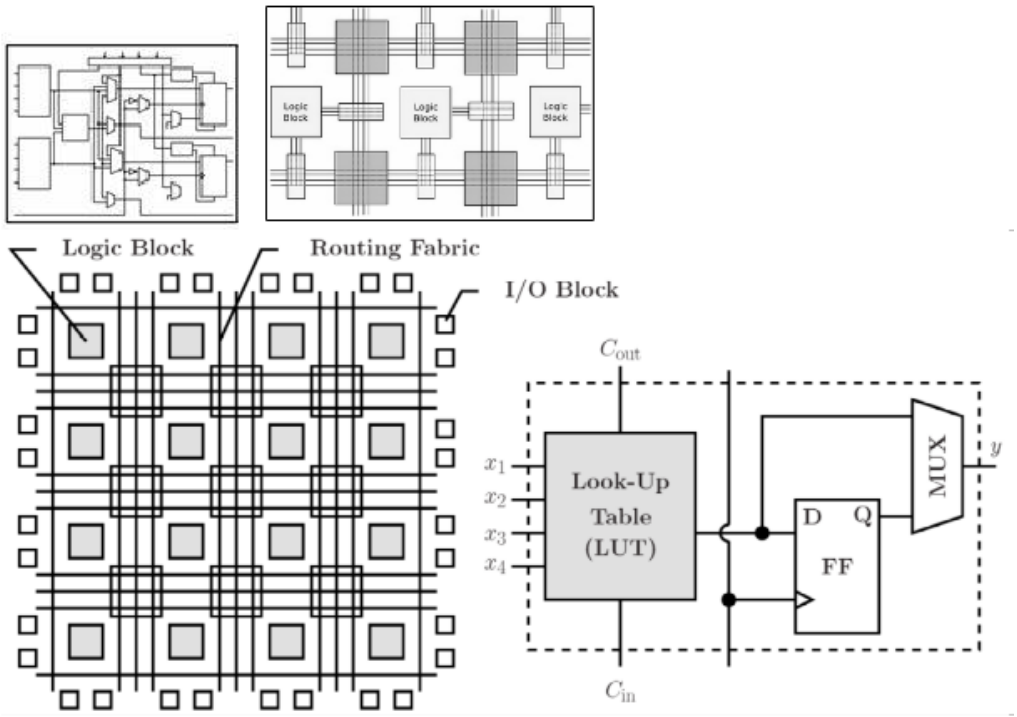
Στην εικόνα παρακάτω (**Εικόνα 8:** Field Programmable Gate Array) απεικονίζονται τα χαρακτηριστικά του FPGA :

1. Configurable Logic Blocks : CLBs ρυθμίζονται από τον προγραμματιστή για να παράγουνν αναδιαμορφώσιμες λογικές πύλες. Επίσης είναι εκεί όπου γίνονται οι υπολογισμοί συγκεκριμένων function του χρήστη.

2. I/O blocks : Συνδέουν το FPGA με άλλα στοιχεία της εφαρμογής. Τα I/O blocks διαμορφώνονται όταν το υπόλοιπο chip έχει διαμορφωθεί.

3. Routing Channels: Τα Routing Channels χρησιμοποιούνται για την οριζόντια ή κάθετη διασύνδεση (interconnection) των Logic Blocks.

4. Interconnect: Η διασύνδεση είναι ουσιώδης για να γράψει μεταξύ των CLB και από I/O blocks στα CLBs.



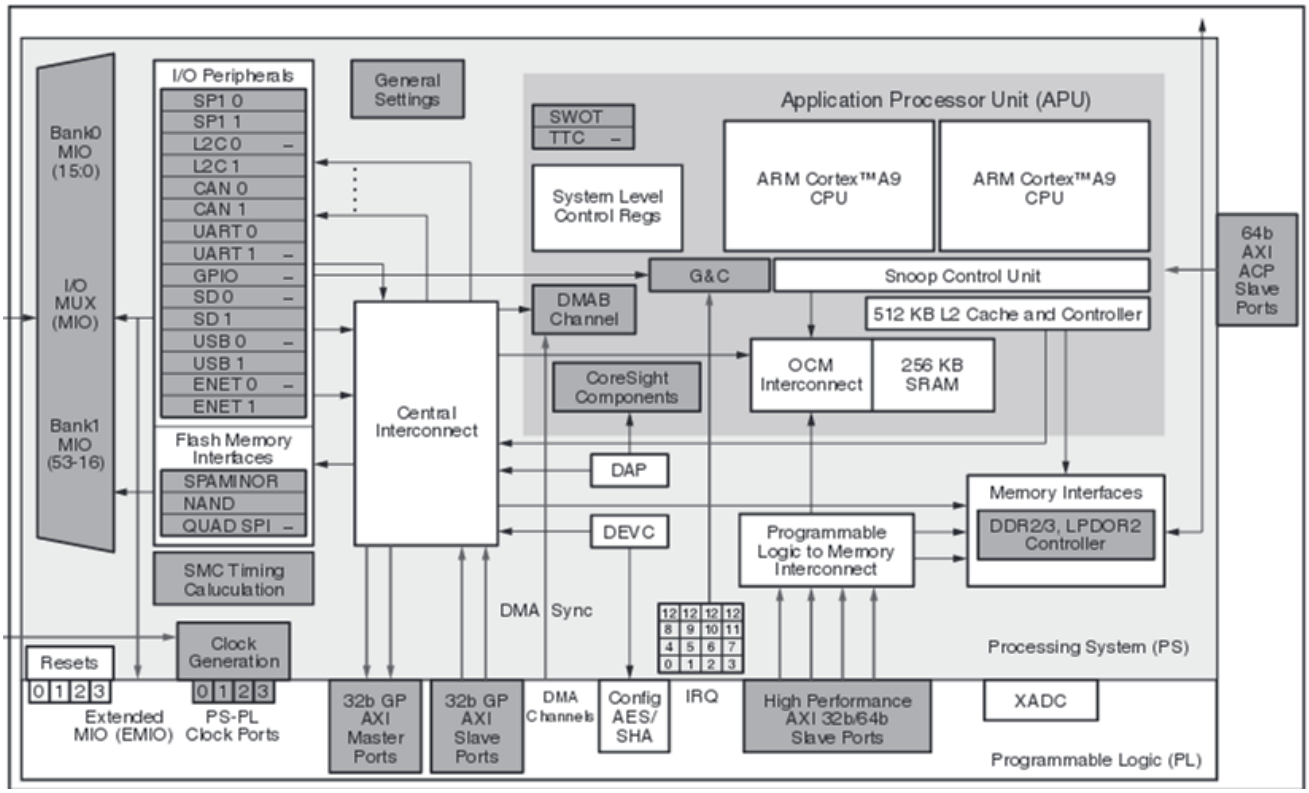
Εικόνα 8: Field Programmable Gate Array

Το Zedboard διαθέτει έναν δίαυλο επικοινωνίας με το υπολογιστικό τμήμα, ο οποίος εξασφαλίζει επικοινωνία κλιμακώμενης απόδοσης και υψηλών επιδόσεων (High bandwidth AMBA interconnect όπως προαναφέρθηκε). Οι διαθέσιμοι πόροι του FPGA αναγράφονται παρακάτω(**Πίνακας 2 :** Πόροι του fpga)

Πόροι	BRAM_18K	DSP48E	FF	LUT
Διαθέσιμοι	280	220	106400	53200

Πίνακας 2: Πόροι του fpga

Στόχος της εργασίας είναι η επιτάχυνση του k-NN με έναν ή περισσότερους επιταχυντές στην Προγραμματιζόμενη Λογική (PL). Οι επιταχυντές θα πρέπει ακόμα να επικοινωνούν με το Υπολογιστικό Σύστημα (PS), συνεπώς κατά κύριο λόγο το σημαντικό κομμάτι βρίσκεται στους πόρους της προγραμματιζόμενης λογικής και στα χαρακτηριστικά του τρόπου διασύνδεσης των δύο μερών της πλακέτας (**Εικόνα 9:** Zynq Block Design).



Εικόνα 9: Zynq Block Design

4. Εισαγωγή στο SDX

Το **SDx** (Software Defined XILINX) είναι ένα περιβάλλον ανάπτυξης λογισμικού για συστήματα και μηχανικούς λογισμικού. Το SDx επιτρέπει στους προγραμματιστές με λίγη ή καθόλου εξειδίκευση στο FPGA να χρησιμοποιήσουν high level γλώσσες προγραμματισμού (C , C++ ,OPEN CL). Αποτελείται από:

1. SDAccel Development Enviroment
2. SDSoc Development Enviroment
3. SDNet Development Enviroment

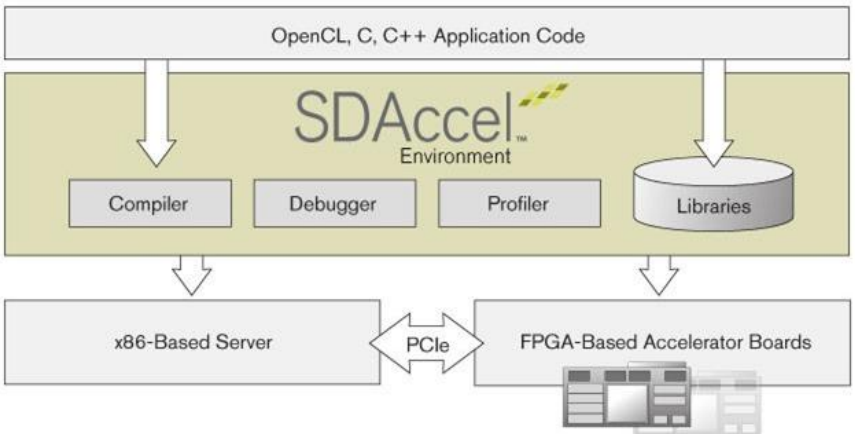
Τα τρία παραπάνω development Enviroment είναι toolchains του SDX. Αναλυτικότερα εξηγούνται παρακάτω η χρησιμότητα και η λειτουργικότητα του καθε toolchain.

4.1 Εισαγωγή στο SDAccel

Το **SDAccel** (Software Defined Accelerator) είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (C , C++ ,OPEN CL) βασισμένο στο FPGA το οποίο εγγυάται 25 φορές καλύτερη απόδοση εφαρμογής [37].Οι δυνατότητες του προγράμματος είναι οι εξής:

1. Δίνει τη δυνατότητα βελτιστοποίησης του compiler εως 25 φορες καλύτερη απόδοση σε σύγκριση με το CPU / GPU.
2. Επιτρέπει με τη χρήση των OpenCL, C και C ++ να παραχθεί κώδικας για να δημιουργηθεί υψηλή απόδοσης στους accelerators.
3. Βελτιστοποιεί εφαρμογές πάνω στο FPGA .
4. Υποστηρίζει μεγάλες εφαρμογές με πολλαπλά προγράμματα (πχ για CPU/GPU)
5. Επιτρέπει στους FPGA accelerators να μοιράζονται σε πολλές εφαρμογές που χρησιμοποιούν on-the- fly μονάδα υπολογιστικής αναδιαμόρφωσης.

Η παρακάτω εικόνα (**Εικόνα 10:** SDAccel Development Enviroment) απεικονίζει το περιβάλλον του SDAccel και τις δυνατότητες που παρέχει .



Εικόνα 10: SDAccel Development Enviroment

4.2 Εισαγωγή στο SDSoc

Ένα από τα εργαλεία σχεδίασης για την πλατφόρμα Zynq είναι το SDSoc. Το **SDSoc** (Software-Defined System On Chip) επιτρέπει στην ευρύτερη κοινότητα των προγραμματιστών λογισμικού ενσωματωμένων συστημάτων με τη χρήση των C/C++/OpenCL συμπεριλαμβανομένου το εύκολο στη χρήση Eclipse IDE να ενισχύσουν τις δυνατότητες του Hardware και Software των «all programmable» συσκευών ανάπτυξης (SoC and MPSoC). Το SDSoc παρέχει ένα υψηλό profile το οποίο δημιουργεί αυτόματη σύνδεση συστήματος και είναι αυτοποιημένο με Harware acceleration και βιβλιοθήκες για μεγαλύτερη ταχύτητα προγραμματισμού. Υποστηρίζει Zynq-7000 All Programmable SoC and Zynq UltraScale+™ MPSoC platforms[36].

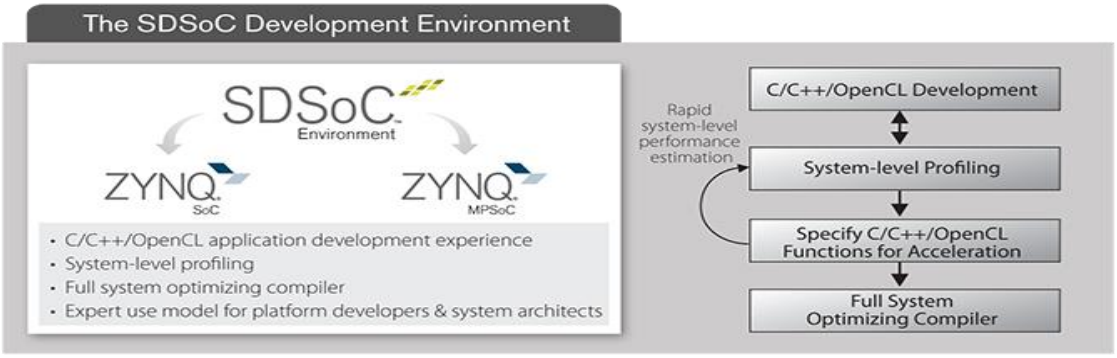
Το πρόγραμμα παρέχει:

1. Γρήγορη Performane Estimation και Area Estimation συμπεριλαμβανομένου του PS (Processing System) δεδομένων επικοινωνίας και PL(Programmable Logic) σε λεπτά.
2. Αυτοποιημένος χρόνος-εκτέλεσης οργάνωση της cache, της μνήμης και της χρησιμότητας του bus.
3. Εκτελεί C/C++/OpenCL εφαρμογές σε ένα πλήρως λειτουργικό Zynq SoC και MPSoC σύστημα.
4. Αυτοποιημένες λειτουργίες acceleration που παρέχουν στη PL λογισμικό ARM και FPGA bitstream.
5. Στοχοποιεί συγκεκριμένα boards με μια συγκεκριμένη μεθοδολογία για τη μετατροπή του ήδη υπάρχοντος Vivado και Software project μέσα στο SDSoC.
6. Το Board Support Packages (BSP) είναι διαθέσιμο και υποστηρίζεται από πολλών ειδών boards.

Το πρόγραμμα υποστηρίζει :

1. bare metal, Linux and FreeRTOS ως λειτουργικό σύστημα .
2. Xilinx libraries οι οποίες είναι διαθέσιμες ως τμήμα του Vivado HLS

Η παρακάτω εικόνα (**Εικόνα 11:** SDSoc Development Enviroment) απεικονίζει το περιβάλλον του SDSoc και τις δυνατότητες παρέχει.



Εικόνα 11: SDSoc Development Enviroment

4.2.1 Περιβάλλον SDSoc

Το SDSoc (Sofware-Defined System On Chip) περιβάλλον είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) Eclipse-based για ετερογενείς ενσωματωμένα συστήματα Zynq-7000 All Programmable SoC και Zynq UltraScale+ MPSoC platforms [38]. Το SDSoc παρέχει ένα ενσωματωμένο C/C++ περιβάλλον ανάπτυξης εφαρμογής μαζί με το εύκολο στη χρήση Eclipse IDE και καθώς και design tools για ετερογενείς Zynq SoC. Περιλαμβάνει ένα full-system βελτιστοποίησης του μεταγλωττιστή C/C++ το οποίο παρέχει αυτοποιημένο Software acceleration στο PL (Programmable Logic) συνδυασμένο με αυτοματοποιημένη συνδεσιμότητα του συστήματος [7]. Μια εφαρμογή πρέπει να είναι γραμμένη σε C/C++ κώδικα με το προγραμματιστή να προσδιορίζει μία συγκεριμένη πλατφόρμα από τις διαθέσιμες του προγράμματος καθώς και ένα υποσύνολο λειτουργίων όπου μέσα από την εφαρμογή θα πρέπει να καταρτίζεται σε Hardware.

Στο SDSoC ο μεταγλωτιστής του συστήματος θα εκτελέσει την εφαρμογή μέσα στο Hardware και στο Software για να ελέγξει το πλήρες ενσωματωμένο σύστημα το οποίο απαιτεί να υλοποιηθεί σε μια συσκευή Zynq. Το SDSoc περιλαμβάνει cross-compilation και σύνδεση των C/C++ functions μέσα στο PL (Programmable Logic) και στις ARM CPUs.

Οι μεταγωτιστές του SDSoc (sdscc/sds++) μετατρέπουν τα C, C++ προγράμματα για Software/Hardware συστήματα βασισμένα σε επιλογές «command line» οι οποίες στοχοποιούν μια πλατφόρμα και τις συναρτήσεις μέσα στο πρόγραμμα για τις εκτελέσούν.

Για να πετύχει υψηλή απόδοση σε κάθε Hardware συνάρτηση τρέχει ξεχωριστά σαν ανεξάρτητα threads. Το SDSoc compilers σύστημα παράγει Hardware και Software στοιχεία τα οποία διατηρούν τη σημασιολογία του προγράμματος και εξασφαλίζουν τον συγχρονισμό μεταξύ των Hardware και Software threads ενώ επιτρέπουν τον pipeline υπολογισμό και επικοινωνία.

Με το κώδικα της εφαρμογής υπάρχει η δυνατότητα να επικαλεστεί πολλές Hardware συναρτήσεις καθώς και πολλαπλά «instances» μίας συγκεκριμένης Hardware συνάρτησηςεπίσης μπορεί να καλεί αυτή τη Hardware συνάρτηση από διαφορετικά μέρη του προγράμματος.

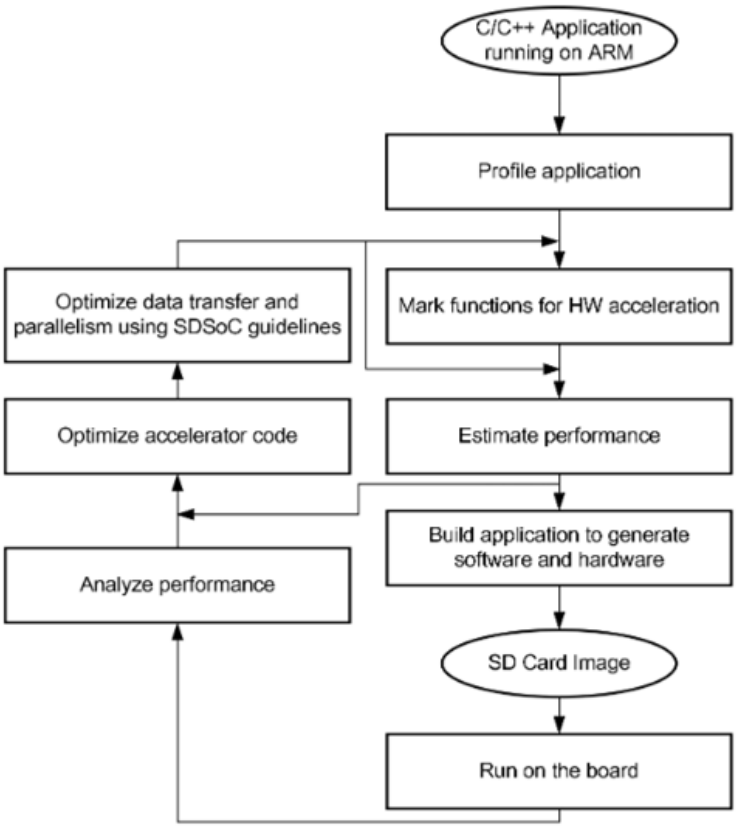
Το SDSoc compiler σύστημαεκτελεί Hardware συναρτήσεις μαζί με το Vivado High-Level Synthesis (HLS) στο PL (ProgrammableLogic) και παράγει ένα ολοκληρωμένο συστημα Hardware βασισμένο σε επιλεγμένη πλατφόρμα συμπεριλαμβανομένων των DMAs, των αλληλοσυνδέσεων ,των Hardware buffers, IP intergrator ,IP library ,RTL synthesis , routing και εργαλείων για να δημιουργήσει το bitstream.Συγκεκριμένα για να παράγει ενα FPGA bitstream επικαλείται το Vivado Design Suite tools.

Το SDSoc compiler σύστημα επίσης παράγει Software stubs δεδομένα διαμόρφωσης τα εκτελεί και συνδέει με το κώδικα της εφαρμογής χρησιμοποιώντας το GNU toolchain μέσα στο binary της εφαρμογής. Για να εναρμονίσει τη μεταφορά των δεδομένων και να ελεγξει τους Hardware accelerators, οι μεταγλωττιστές αυτόματα δημιουργούν Hardware specific Software κώδικα διαμόρφωσης ενσωματώνοντας κάθε driver που σχετίζεται με τα IP block που παράχθηκαν.

Μια πλατφόρμα SDSoc καθορίζει τη Hardware / Software βάση αρχιτεκτονικής και τη γενική έκφραση της εφαρμογής συμπεριλαμβανομένου του processing system , των interfaces εξωτερικής μνήμης , προσαρμοσμένα input /output και Software run-time μαζί με το λειτουργικό του συστήματος (πιθανόν και baremetal) , boot loaders, drivers για τα περιφερειακά της πλατφόρμας και root file system.

Το SDSoc IDE (integrated development environment) υποστηρίζει ένα προτυπο ανάπτυξης λογισμικού με μεταγλώττιση , debugging καθώς και γρήγορη performance estimation. Το SDSoc IDE προσαρμόζει τη πλατφόρμα με συγκεκριμένη εφαρμογή Hardware accelerators και data motion networks που κάνει σύνδεση με accelerators στη πλατφόρμα.

Στο σχήμα παρακάτω φαίνεται η ροή του συστήματος. Περιλαμβάνει επανάληψη σε βήματα μέχρι το παραγόμενο σύστημα επιτύχει σωστή απόδοση.

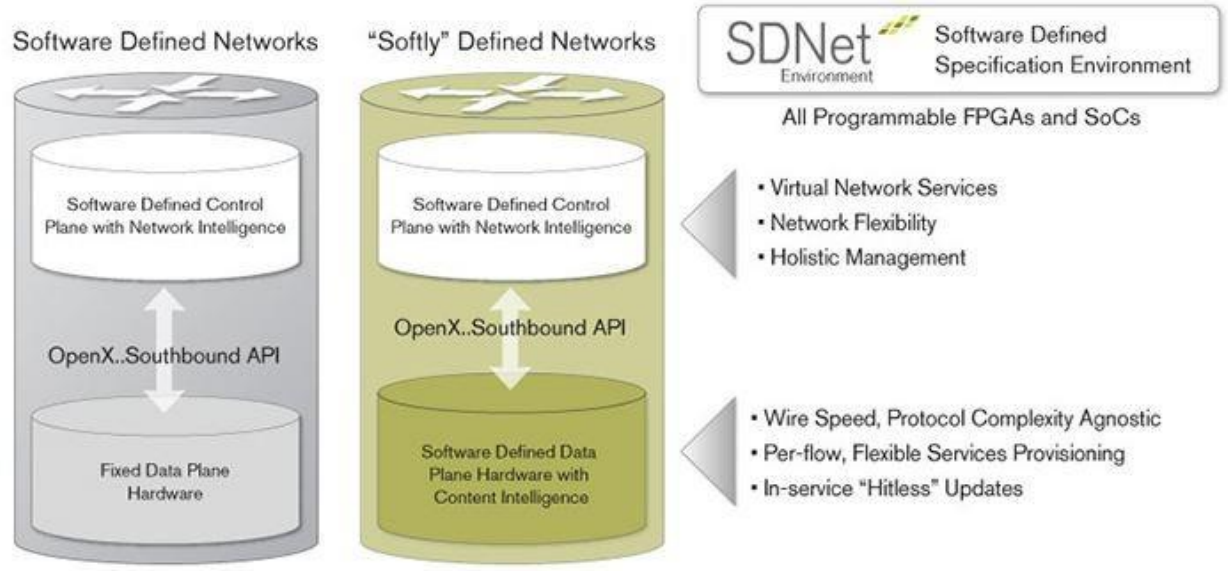


Εικόνα 12: User Designed Flow (SDSoc)

4.3 Εισαγωγή στο SDNet

Το **SDNet** (Software Defined Network) σε συνδυασμό με το Xilinx All Programmable FPGAs και SoCs επιτρέπειτηδημιουργία 'Softly' Defined Networks μια τεχνολογία που πηγαίνει πολύ πιο πέρα από τη σημερινή Software Defined Networking (SDN) αρχιτεκτονική. Το SDNet περιβάλλον επιτρέπει πολλαπλές διασπαστικές δυνατότητες [35].Παρέχει:

1. Υποστήριξη υπηρεσιών ταχύτητας καλωδίου η οποία είναι ανεξάρτητη από τη πολυπλοκότητα πρωτόκολλου.
2. Υποστήριξη της υπηρεσίας "hitless" χαρακτηριστικό για ενημέρωση ενώ λειτουργεί στα 100% line rate (ρυθμός μετάδοσης) .
3. Το per-flow, ευέλικτες υπηρεσίες.



Εικόνα 13:SDNet – Software Defined Specification Enviroment for Netorking (1)

Η παραπάνω εικόνα (**Εικόνα 13:** SDNet Development EnviromentforNetorking (1)) απεικονίζει το περιβάλλον του SDNet και τις δυνατότητες που παρέχει.

Αυτές οι μοναδικές δυνατότητες επιτρέπουν στα carriers και MSOs να παρέχουν δυναμικές μονάδες διαφοροποιημένων υπηρεσιών. Το Softly Defined Network επιτρέπει την εκτεταμένη διαφοροποίηση μέσω της ανάπτυξης του περιεχομένου data plane (κομμάτι δικτύου μέσα από το οποίο τα δεδομένα μεταδίδονται) σε επίπεδο Harware που καθορίζεται από το περιβάλλον SDNet. Το Softly Defined Network έχει κάποια επιπλέον οφέλη-λειτουργίες όπως :

1. Βελτιστοποιημένο ιδιαίτερα ευέλικτο Quality of Service (QoS)

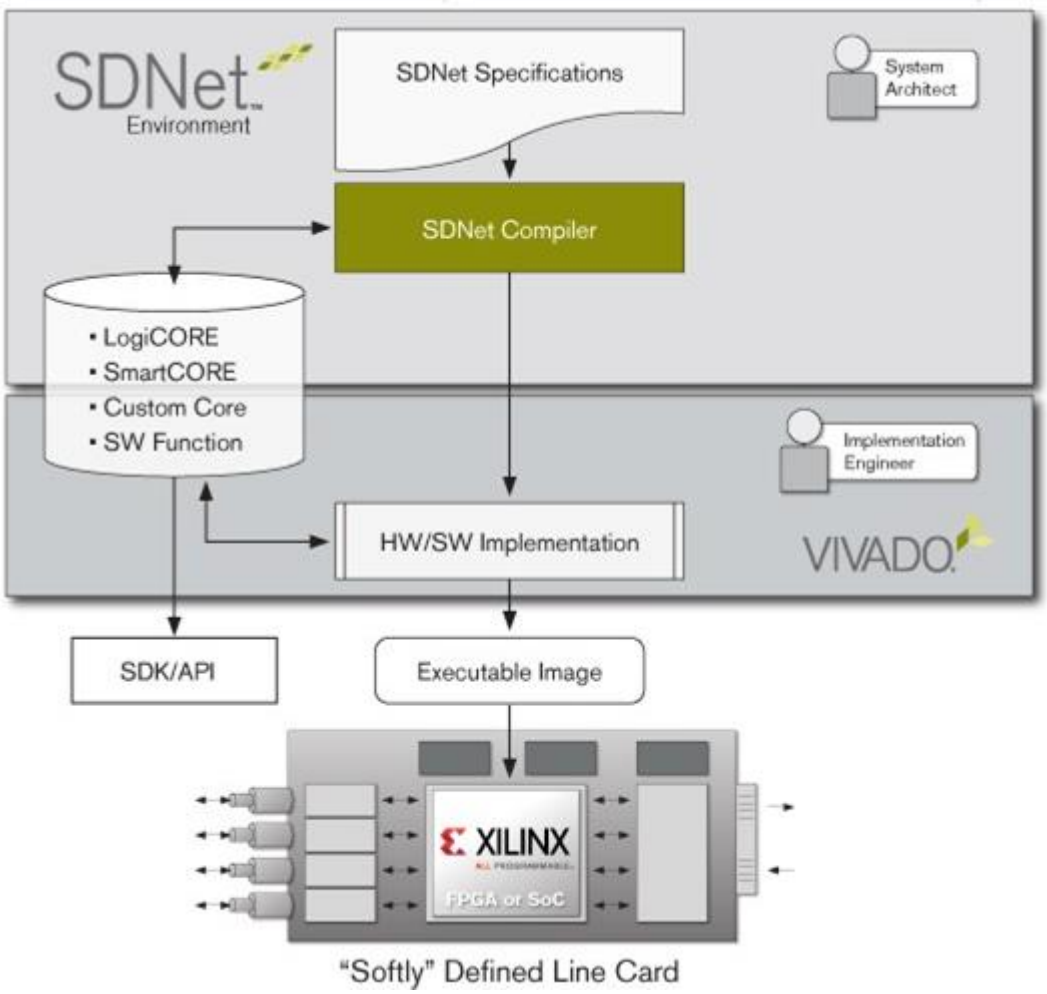
2. Row and Session, επίγνωση των δυνατοτήτων

3. Πλήρως προγραμματιζόμενο data plane υλικού και I / O (Digital Cable Cervice).

4. Υποστήριξη για NFV στην ταχύτητα σύρματος, συμπεριλαμβανομένου ότι ορίζονται από το χρήστη, προσαρμοσμένες δυνατότητες.

5. Κλιμακώνεται σε line rates (physical layer) απο 1G εώς 400G.

Το παρακάτω σχήμα (SDNet-Software Defined Specification Enviroment for Networking) δείχνει πως με την αρχιτεκτονική του συστήματος μπορεί να χάσεις τα προνόμια του συστήματος ενώ με το Softly Defined Network χρησιμοποιούνται τα δίκτυα χωρίς να απαιτείται μια λεπτομερή γνώση της υποκείμενης All Programmable αρχιτεκτονικής της συσκευής. Αυτή η εκτέλεση ροής επιτρέπει επίσης οι αρχιτεκτονικές του συστήματος να επικεντρωθούν μόνο στις υπηρεσίες που ψάχνουν την παροχή, χωρίς να χρειάζεται να επικεντρωθούν στο πώς ακριβώς υλοποιούνται αυτές οι υπηρεσίες.



Εικόνα 14: SDNet – Software Defined Specification Enviroment for Netorking (2)

5. Εντολές για Επιτάχυνση του k-NN Αλγορίθμου σε Hardware

Για να την επιτάχυνση ενός αλγορίθμου πάνω σε Hardware είναι απαραίτητες κάποιες εντολές (#pragma) οι οποίες απευθύνονται στον προεπεξεργαστή (preprocessor). Οι εντολές αυτές παρέχουν οδηγίες για τον μεταγλωττιστή που απευθύνονται στο συγκεκριμένο μηχανισμό. Αναθέτει δηλαδή στον μεταγλωττιστή να κάνει κάτι, να ρυθμίσει κάποια επιλογή, να πάρει κάποια ενέργεια, να παρακάμψει κάποια προεπιλογή κλπ.. Στο πρόγραμμα SDSoc οι εντολές pragmas διακρίνονται σε SDS και HLS[31].

5.1 Εντολές Software-Defined **SoC** (SDS)

Στο περιβάλλον SDSoC, υπάρχει η δυνατότητα ελέγχου της διαδικασίας του «process» του συστήματος, διαμορφώνοντας τις Hardware συναρτήσεις και τις κλήσεις των Hardware συναρτήσεωνγια να υπάρξει ισσοροπία μεταξύ της επικοινωνίας και του υπολογισμού.Για να γίνει αυτό πρέπει να εισάγχθούν pragma εντολές στον κώδικα ούτως ώστε να καθοδηγήσει τον μεταγλωττιστή του συστήματος σύμφωνα με τις απαιτήσεις του προγραμματιστή. Ο μεταγλωττιστής SDSoC είναι προκαθορισμένος να επιλέγει αυτόματα την καλύτερη δυνατή θύρα του συστήματος που θα χρησιμοποιήσει για οποιαδήποτε μεταφορά δεδομένων, αλλά επιτρέπει την αλλαγή αυτής της επιλογής χρησιμοποιώντας pragmas. Τα pragmas μπορεί επίσης να καθοριστούν για την επιλογή διάφορων data mover για τα ορίσματα της Hardware συνάρτησης.

Όλα τα pragmas που είναι συγκεκριμένα για το SDSoC περιβάλλον παίρνουν από μπροστά το #pragma SDS και θα πρέπει να εισαχθούν στον κώδικα (C / C ++), είτε αμέσως πριν από μια δήλωση συνάρτησης είτε σε έναν τόπο κλήσεων συνάρτησης για τη βελτιστοποίηση μιας συνάρτησης .

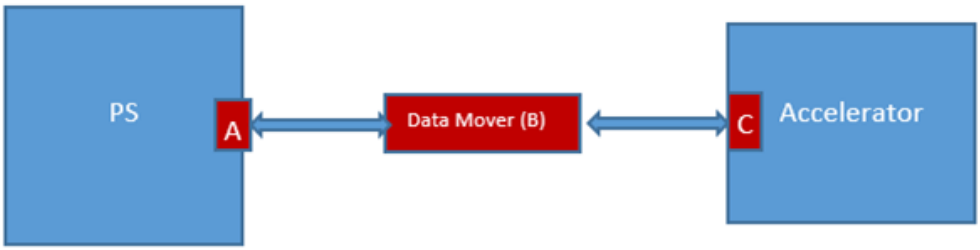
5.1.1 Εντολές για Data Motion

Γενικότερα με την Data Motion Βελτίωση γίνεται η μεταφορά των δεδομένων ανάμεσα στο PS (Processing System) και το PL (Programmable Logic)(**Εικόνα 15:** Data Motion Network Components). Ο τρόπος μεταφοράς δεδομένων χρησιμεύει για παράδειγμα για να εξασφαλιστεί ότι τα δεδομένα αποθηκεύονται σε συνεχής μνήμη ούτως ώστε να βελτιωθεί η αποδοτικότητα της μεταφοράς δεδομένων είτε ως simple είτε ως scatter-gather(τα οποία εξηούνται στη συνέχεια) η οποία χρησιμοποιείται για πιο αποδοτική μεταφορά όταν το σετ των δεδομένων είναι μεγάλο . Γενικότερα ο DMA controller (που λειτουργεί είτε ως simple είτε ως scatter-gather) είναι αρκετά χρήσιμος στη μεταφορά των δεδομένων εφόσον με αυτόν τον τρόπο δεν επιβαρύνεται η cpu.

Το Data Motion Network είναι κατασκευασμένη από τα εξής στοιχεία:

1. Το Hardware Interface του Accelerator (A)
2. Τους data movers ανάμεσα στο PS (Proccessing System) και τον Accelerator ή μεταξύ των Accelerators (B)

3. Τα ports της μνήμης του συστήματος στο PS (C)



Εικόνα 15: Data Motion Network Components

Χωρίς τις pragma εντολές που καθορίζουν τους datamovers που θα χρησιμοποιηθούν, το πρόγραμμα παράγει ένα Data Motion Network με βάση τον κώδικα.

Hardware interfaces (κανάλια):

1. **FIFO/ RAM**

Η εντολή «data access pattern» καθορίζει το χαρακτηριστικό πρόσβασης στα δεδομένα στη Hardware function. Εάν οριστεί ο COPY data mover για ένα όρισμα πίνακα, θα γίνει έλεγχος του μεγέθους αυτής της εντολής για να καθορίσει το Hardware interface . Αντο access pattern είναι SEQUENTIAL θαδημιουργηθείένα streaming interface (ap_fifo) . Ειδάλλως αν το access pattern είναιRANDOM θα δημιουργηθεί ένα RAMinterface . Με αυτή την εντολή γενικότερα καθοδηγείται ο accelerator να δημιουργήσει ένα interface για τη μεταφορά των δεδομένων. Συνδέεται αυτόματα στο AXI4-Stream κανάλι (το οποίο ειναι συμβατό με το DMA πρωτόκολλο)[20]. Γενικά μια Hardware function δεν μπορεί να έχει περισσότερα από οχτώ input bram ή ap_fifo ορίσματα και περισσότερα από οχτώ output bram ή ap_fifo arguments (αν υπάρχουν περισσότερα από οχτώ ορίσματα συνίσταται να χρησιμοποιηθεί το AXI4-Stream interface)[22].

Αυτή η εντολή δεν τοποθετείται πάνω από το τον ZEROCOPYdatamover. Θα την αγνοήσει.

Επίσης αν υπάρχει πίνακας ο οποίος περνάει από τη συνάρτηση ως «pointer» πρέπει απαραίτητα να εισαχθεί και ο datamoverCOPY[18]. Χρησιμοποιείται για μεγάλο όγκο δεδομένων μεχρι 8 ορίσματα.

2. **AXI-memory mapped (AXI-MM) master**

Σε αυτή τη λειτουργία, η Hardware function λειτουργεί σαν να είναι η ίδια ένας data mover. Όταν μια Hardware function αντιστοιχίζει (map) ένα όρισμα σε AXI-MM master, πρέπει επίσης να συμπεριλάβει οπωσδήποτε είτε ένα όρισμα εξόδου είτε μια τιμή επιστροφής[19]. Επιτρέπει burst μέχρι 256 κύκλους

μεταφοράς δεδομένων [19] (ZERO COPY data mover πάνω από τη δήλωση της συνάρτησης). Χρησιμοποιείται για αντιστοίχιση ορίσματος σε κοινή μνήμη.

3. AXI4-Lite

Οι scalar μεταβλητές αυτόματα αντιστοιχίζονται με βασικούς αριθμητικούς τύπους (8, 16, or 32 bits) μέσα σε ένα register προσβάσιμο μέσω ενός AXI-Lite interface[20]. Οι registers αντιμετωπίζονται σαν FIFOs για να υποστηρίζουν task pipelining με πολλαπλές in-flight task κλήσεις. Μια Hardware function μπορεί να έχει μέχρι 8 inout scalar ορίσματα, μέχρι 16 input scalars ή 16 output scalars, μέχρι 24 scalar ορίσματα δηλαδή στο σύνολο συμπεριλαμβανομένου και των τιμών επιστροφής. Αν απαιτούνται περισσότερα scalar ορίσματα πρέπει να εισαχθεί η εντολή: #pragma HLS INTERFACE s_axilite port=arg εντός της Hardware συνάρτησης μαζί με την εντολή: #pragma HLS INTERFACE s_axilite port=return για να δημιουργηθεί memory mapped control interface.)Χρησιμοποιείται για scalar μεταβλητές [22].

4. AXI4-Stream

Χρησιμοποιείται όταν μια Hardware συνάρτηση έχει πάνω από οχτώ ορίσματα και για άμεσες συνδέσεις μεταξύ Hardware functions με αντίστοιχες διασυνδέσεις AXI4-Stream (χρησιμοποιώντας εσωτερικά της συνάρτησης την εντολή: #pragma HLS INTERFACE axis port=arg).)[22].

Data Mover IPs:

- **AXI DMA (Direct Memory Access)**

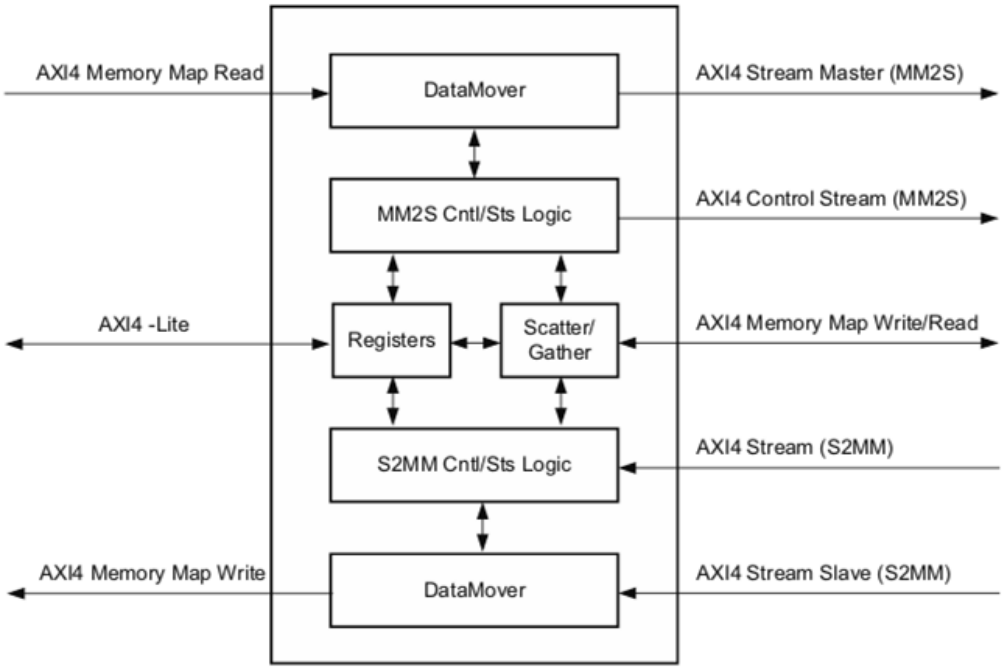
Ο DMAcontrollerπαρέχει τη δυνατότητα στα υποσυστήματα ενός υπολογιστή να έχουν πρόσβαση στη μνήμη του συστήματος για ανάγνωση ή εγγραφή δεδομένων χωρίς να απασχολούν τη CPU.
Οι υπολογιστές που περιλαμβάνουν κανάλια DMA μπορούν να μεταφέρουν όγκο δεδομένων από και προς συσκευές με μικρότερη επιβάρυνση του επεξεργαστή σε σύγκριση με υπολογιστές χωρίς κανάλια DMA. Το πρότυπο DMA υλοποιείται με έναν εξειδικευμένο επεξεργαστή-ελεγκτή (controller) ο οποίος μεταφέρει δεδομένα μεταξύ της μνήμης και μιας συσκευής εισόδου/εξόδου, ενώ την ίδια στιγμή ο επεξεργαστής ασχολείται με άλλες διεργασίες. Γι' αυτό το λόγο βρίσκεται εκτός του επεξεργαστή, κατευθύνοντας την ανάγνωση και την εγγραφή δεδομένων ανάμεσα στη συσκευή και τη μνήμη.[16]

Η AXI Direct Access (AXI DMA) IP παρέχει άμεση πρόσβαση μνήμης υψηλού εύρους ζώνης μεταξύ των IP AXI4 memory-mapped interface και AXI4-Stream Interface (συνδυασμός AXI4-Stream και Memory Mapped Protocols). Για τη λειτουργία του DMA IP πρέπει εισαχθεί AXI-Stream Interface.

Ο AXI DMA controller, που μετακινεί δεδομένα με την υψηλή ταχύτητα του DMA μεταξύ του memory-map και του Stream, δρομολογεί μέσα από το AXI4 memory-mapped στον stream master (MM2S). ToAXI4 streamδρομολογείπροστακανάλιαmemory-mappedslaveinterface (S2MM).Αυτά τα κανάλια λειτουργούν ανεξάρτητα. Το κανάλι **MM2S** υποστηρίζει ένα πρόσθετο AXI control stream για αποστολή δεδομένων (εφαρμογών χρήστη) στον IP .

Παρέχεται ένα πρόσθετο AXI status stream για το κανάλι **S2MM** για τη λήψη δεδομένων εφαρμογής χρήστη από τον IP. Οι καταχωρητές DMA, που έχουν πρόσβαση μέσω του core interface AXI4-Lite, θα

πρέπει να συνδεθούν στις θύρες master PS GP[17]. Στην παρακάτω εικόνα (**Εικόνα 16:** AXI DMA Σχηματικό Διάγραμμα) απεικονίζεται η λειτουργία του DMA που προαναφέρθηκε .



Εικόνα 16: AXIDMA Σχηματικό Διάγραμμα

Ο AXI DMA μπορεί να λειτουργήσει με δύο διαφορετικούς τρόπους:

- Register Direct Mode (Simple DMA)
- Scatter/Gather Mode (SGDMA)

Register Direct Mode (Simple DMA)

Το AXI DMA σε Simple λειτουργία DMA χρησιμοποιείται ειδικά για περιπτώσεις όπου απαιτείται μεταφορά υψηλής ταχύτητας από ένα περιφερειακό στο PL(Programmable Logic).[17]

Το AXI DMA με Simple λειτουργία DMA παρέχει διαμόρφωση για απλές μεταφορές DMA (δηλαδή μεταφορές όπου τα δεδομένα που λαμβάνονται είναι κατανεμημένα με συνέχεια) που απαιτούν λιγότερη χρήση πόρων FPGA . Με αυτόν τον τρόπο, όλες οι μεταφορές ξεκινούν εγγράφοντας στον έλεγχο DMA, την προέλευση ή τη διεύθυνση προορισμού και το μήκος των «registers».Όταν ολοκληρωθεί η μεταφορά, επιδιώκει «interrupt» (διακοπή) του καναλιού και, αν είναι ενεργοποιημένο, παράγει «interrupt».

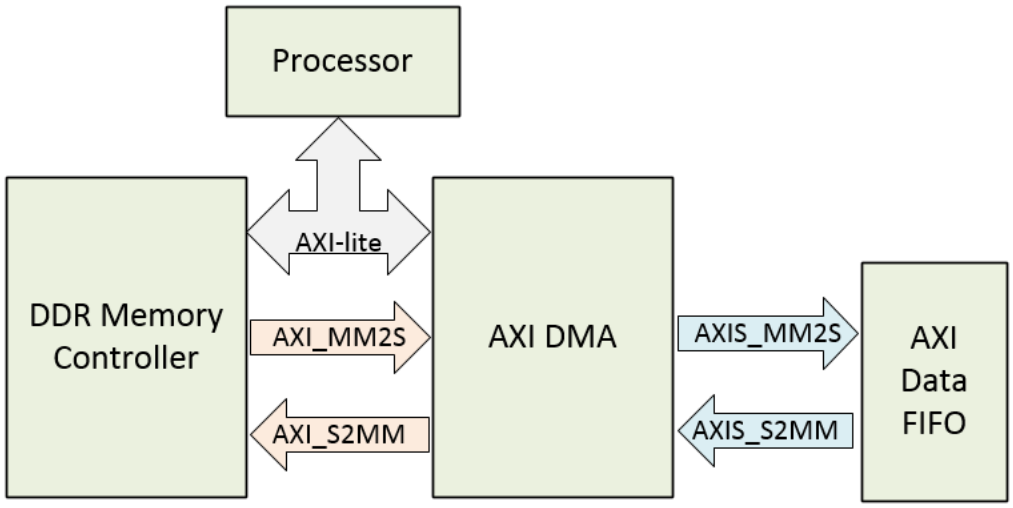
Scatter/Gather Mode (SG DMA)

Σε αντίθεση με την Simple λειτουργία DMA, ο τρόπος scatter/gather επιτρέπει μεταφορές DMA υψηλότερης απόδοσης αλλά με την επιπλέον κατανάλωση πόρων του FPGA. Το AXI DMA διαθέτει ένα προαιρετικό AXI4 scatter / gather read / write master interface μέσω του οποίου ο μηχανισμός

scatter / gather συλλέγει και ενημερώνει τους χαρακτηριστικούς buffer της μνήμη του συστήματος.

Αυτό εξαλείφει τις εργασίες διαχείρισης του DMA (δηλαδή να γράφει τη διεύθυνση, το μήκος και τους «registers» ελέγχου) από την CPU και αντ 'αυτού ανακτά και ενημερώνει αυτόματα τις πληροφορίες από τον πίνακα περιγραφικών στοιχείων, μεγιστοποιώντας έτσι την απόδοση δεδομένων.

Στην παρακάτω εικόνα απεικονίζεται οι τρόποι προσπέλασης της μνήμης (CPU/DMA) και η εξωτερική DDR (**Εικόνα 17:** Διάγραμμα Προσπέλασης Μνήμης).



Εικόνα 17: Διάγραμμα Προσπέλασης Μνήμης

- **AXI FIFO**

Το AXI Streaming FIFO επιτρέπει memory mapped πρόσβασηστο AXI Streaming interface. Μπορεί να χρησιμοποιηθεί στο AXI Ethernet interface χωρίς τη χρήση του DMA. Η κύρια λειτουργία αυτού είναι ότι επιτρέπει να γράψει ή να διαβάσει από τα data packets προς ή από μια συσκευή χωρίς καμία ανησυχία σχετικά με το AXI Streaming interface (μέχρι 32-bit AXI Memory Map)[23].

5.1.1.1 Data Movers

Ένας Data Mover μεταφέρει δεδομένα μεταξύ του PS (processing system) και των επιταχυντων (accelerators) καθώς και μεταξύ των επιταχυντων (accelerators).

Οι «data movers» είναι βασισμένοι στις ιδιότητες και το μέγεθος των δεδομένων που μεταφέρονται. Αυτοί είναι οι AXI DMA SG, AX DMA SIMPLE(<8MB), AXI DMA 2D, AXI FIFO ,AXI M ή AXI LITE ,ZERO COPY ,COPY data movers, οι οποίοι χρησιμοποιούνται για διαφορετικούς σκοπούς και εξαρτώνται από τη μνήμη, τις ιδιότητες και το μέγεθος των δεδομένων ενός πίνακα.

Για παραάδειγμα, αν ο πίνακας δεσμεύσει χώρο στη μνήμη χρησιμοποιώντας τη malloc(),ως εκ τούτου η μνήμη δεν είναι συνεχόμενη και τα δεδομένα μεταφέρονται με AXI_DMA_SG.

Ωστόσο αν το μέγεθος των δεδομένων είναι λιγότερο από 300 bytes, χρησιμοποιείται η AXI_FIFO δεδομένου ότι ο χρόνος μεταφοράς δεδομένων είναι μικρότερος από του AXI_DMA_SG και καταλαμβάνει πολύ λιγότερους πόρους στο PL (Programmable Logic) .

Το ZERO_COPY αντιστοιχίζει το όρισμα σε κοινή μνήμη (για INOUT ορίσματα).Ο παρακάτω πίνακας που ακολουθεί (**Πίνακας 3:** Data Mover) αναλύει τα χαρακρκτηριστικά των Data Movers[34].

SDSoC Data Mover	Vivado IP Data Mover	Accelerator IP Port Types	Transfer Size	Contiguous Memory Only
axi_lite	processing_system7	register, axilite		
axi_dma_simple	axi_dma	bram, ap_fifo, axis	< 8 MB	✓
axi_dma_sg	axi_dma	bram, ap_fifo, axis		
axi_dma_2d	axi_dma	bram		✓
axi_fifo	axi_fifo_mm_s	bram, ap_fifo, axis	(≤ 300 B)	
zero_copy	accelerator IP	aximm master		✓
copy	axi_dma	bram, ap_fifo, axis		

Πίνακας 3: SDSoc Data Mover

Οι datamovers AXI DMA SIMPLE ,AXI DMA 2D και AXI DMA SG χρησιμοποιούν τον DMA controller για τη μεταφορά δεδομένων .Επίσης χρησιμοποιούνται για δεδομένα μέχρι 16K bytes (τα FPGA έχουν περιορισμένη μνήμη BRAM on-chip, ώστε το μέγιστο μέγεθος πίνακα για ένα όρισμα λειτουργίας περιορίζεται στα 16K (16384 στοιχεία) στο SDSoC για τους παρακάτω data movers). Ο AXI FIFO data mover χρησιμοποιεί ένα FIFO controller αντί για DMA. Οι μεταβλητές (ορίσματα της Hardware συνάρτησης) πρέπει να είναι είτε IN(read) είτε OUT(write).

- **AXI DMA SIMPLE** (μονοδιάστατο πίνακα):Ο AXI DMA SIMPLE data mover είναι η πιο αποδοτική bulk transfer μηχανή,αλλά υποστηρίζει μέχρι 8M, επομένως είναι πιο αποδοτικός σε

μεγάλες μεταφορές[14].

```
#pragma SDS data data_mover (samplesu: AXIDMA_SIMPLE, group:AXIDMA_SIMPLE)
 void knn_algorithm_hw (doublesamplesu [MAT*4], doublegroup [MAT])
```

- **AXI DMA 2D** (δισδιάστατο πίνακα): Για τον AXI DMA 2D data mover ισχύει ότι ακριβώς ισχύει και για τον AXI DMA SIMPLE, η μόνη διαφορά είναι ότι AXI DMA 2D data mover υποστηρίζει μόνο δισδιάστατο πίνακα. Το SDS data copy pragma χρειάζεται επίσης για να καθορίσει μια ορθογώνια υποπεριοχή του δισδιάστατου πίνακα που πρόκειται να μεταφερθεί. Στο παράδειγμα που φαίνεται παρακάτω, τα NUMCOLS, row_offset(0:), col_offset(0:) και cols πρέπει να είναι πολλαπλάσια του 8 (κάθε char bitwidth είναι 8) για να λειτουργήσει σωστά το AXIDMA2D.

```
#pragma SDS data data_mover (deigma: AXIDMA_2D, deigma1:AXIDMA_2D)
#pragmaSDS data dim (deigma [MAT][4], deigma1[MAT][4])
#pragma SDS data copy (deigma [0: MAT][0:4], deigma1[0:MAT][0:4])
void eukleideia_best_voters (doubledeigma [MAT][4], double deigma1[MAT][4]);
```

- **AXI DMA SCATTER-GATHER** (μονοδιάστατο πίνακα):Ο AXI DMA SG (scatter-gather DMA) παρέχει όχι και τόσο γρήγορη απόδοση DMA και καταναλώνει περισσότερα Hardware resources, αλλά έχει λιγότερους περιορισμούς, ακόμα και υπό την έλλειψη καθοδήγησης των pragma. Είναι συχνά η καλύτερη επιλογή data mover[14].

```
#pragmaSDSdatadata_mover (samplesu: AXIDMA_SG, group: AXIDMA_SG)
void knn_algorithm_hw (doublesamplesu [MAT*4], doublegroup [MAT]);
```

- **AXI FIFO** (μονοδιάστατο πίνακα):Ο AXI FIFO data mover δεν απαιτεί τόσα Hardware resources όσα ένας DMA αλλά λόγω των βραδύτερων ρυθμών μεταφοράς του, προτιμάται μόνο για δεδομένα μέχρι 300 byte.

```
#pragma SDS data data_mover (A: AXI_FIFO)
void knn_algorithm_hw(double A [15]);
```

Ο data mover ZEROCOPY χρησιμοποιείται όταν είναι να μεταφερθούν πολλά δεδομένα. Επίσης η διαφορά με τους υπόλοιπους datamovers είναι ότι ο ZEROCOPY τα ορίσματα του μπορούν να είναι INOUT δηλαδή αντιστοιχίζει το όρισμα σε κοινή μνήμη. Απαιτεί τα δεδομένα του να είναι

αποθηκευμένα σε συνεχόμενο χώρο μνήμης (sds_alloc). OZERO COPY data mover αντί για DMA χρησιμοποιεί ένα κανάλι master interface το οποίο έχει και δυνατότητα για «busrst transaction» ούτως ώστε να μεταφέρει τα δεδομένα από και προς τη μνήμη.

- **1ος Τρόπος ZERO COPY**(για λίγα δεδομένα):Η απόδοση με λίγα δεδομένα δεν είναι και τόσο καλή όσο ενός DMAdatamover.

```
1  #pragma SDS data zero_copy (samples)
2  voidknn_algorithm_hw (doublesamples [15])
```

- **2ος Τρόπος ZERO COPY**(για πολλά δεδομένα):Όταν οι πίνακες που περνιούνται από τη συνάρτηση έχουν μεγάλο μέγεθος είναι ένας κατάλληλος τρόπος μεταφοράς δεδομένων.

```
1 #pragma SDS data zero_copy (samples)
2 #pragma SDS data mem_attribute (samples: CACHEABLE|PHYSICAL_CONTIGUOUS)
3 voidknn_algorithm_hw(double samples[10000*4]);
```

Αν είναι πολλά τα δεδομένα είναι πολλά τότε εκτός από το sds_alloc() απαιτεί να εισαχθεί και η εντολή: **#pragma SDS data mem_attribute** η οποία χρησιμοποιείται όταν ο compiler sdscc δεν μπορεί να συμπεράνει τη συνέχεια της μνήμης.

Ο data mover **COPY**χρησιμοποιείται κυρίως για να μεταφέρει τα δεδομένα όταν έχουν αρκετά μεγάλο μέγεθος στη Hardware συνάρτηση και ανάλογα με την υλοποίηση της τα δεδομένα μεταφέρονται είτε με AXIDMASG είτε με AXIDMA SIMPLE. Ο **COPY** data mover όπως προαναφέρθηκε παραπάνω μεταφέρει είτε δισδιάστατους πίνακες είτε μονοδιάστατους επίσης αντιγράφει τα δεδομένα ως είσοδος ως έξοδος.

Το COPY μπορεί να υλοποιηθεί με τρεις διαφορετικούς τρόπους και ανάλογα με την περίπτωση αλλάζει και η υλοποίηση του κώδικα.

- 1ος Τρόπος COPY(Για λίγα δεδομένα) :Με αυτόν τον τρόπο το copy θα χρησιμοποιήσει τον AXI DMA SIMPLE. Η σύνταξη του κώδικα με copy είναι η εξής:

```
1  #pragma SDS data_copy (samples, group)
2  void knn_algorithm_hw (doublesamples [15], double deigmax, double
deigmay,doublebest_voters [15]);
```

- 2ος Τρόπος COPY (Για μεγάλο σετ δεδομένων): Όταν οι πίνακες που περνιούνται από τη συνάρτηση έχουν μεγάλο μέγεθος αυτός ο τρόπος υλοποίησης είναι ο ιδανικός. Με αυτόν τον τρόπο τα δεδομένα μεταφέρονται με AXI DMA SIMPLE. Η σύνταξη του κώδικα για να ένας μεγάλος πίνακας είναι η εξής:

```
1 #pragma SDS data access_pattern (samples: SEQUENTIAL, best_voters: SEQUENTIAL)
2 #pragma SDS data copy (samples [0:5000][0:4], best_voters [0:3][0:4])
3 void eukleideia_best_voters (doubledeigma [5000][4], double deigmax, double deigmay, double best_voters [3][4]);
```

- 3ος Τρόπος COPY (Για να περαστεί ένας πίνακας σαν pointer): Οι πίνακες οι οποίοι περνιούνται σαν pointer αν δεν χρησιμοποιήσουν το copy (samples) τότε το πρόγραμμα τους αντιμετωπίζει σαν pointer μεταβλητών. Αυτό είναι πρόβλημα διότι τότε τα δεδομένα θεωρούνται θα είναι «scalar» άρα θα περάσουν αναγκαστικά (για να μην σπαταλούν πόρους) από τη GP0 πόρτα κάτι που είναι αρνητικό για την απόδοση του αλγορίθμου. Το μέγεθος δεν χρειάζεται να είναι σταθερά αλλά να δείχνει σε κάποια σταθερά. Με αυτόν τον τρόπο τα δεδομένα μεταφέρονται με AXI DMA SG. Η σύνταξη του κώδικα για να περαστεί ένας πίνακας σαν pointer είναι η εξής:

```
1 #pragma SDS data access_pattern (samples: SEQUENTIAL, group: SEQUENTIAL)
2 #pragma SDS data copy (samples [0: height*width], group [0: height])
3 void knn_algorithm_hw (double* samples, double*group, int height, int width, intu,);
```

Το COPY απαιτεί να είναι και sequential (#pragma SDS data access_pattern) στις δύο τελευταίες περιπτώσεις (όταν δηλαδή τα δεδομένα είναι πολλά). Το pragma αυτό εξασφαλίζει ότι τα δεδομένα παίρνονται από τη μνήμη σειριακά από τη μνήμη.

5.1.1.2 Επιλογή Σωστού Data Mover Ανάλογα με το Είδος του Ορίσματος

Οι παραπάνω Data Movers επιλέγονται ανάλογα με την υλοποίηση της εφαρμογής και ανάλογα με το είδος των ορισμάτων. Η αντιστοίχιση data mover με όρισμα γίνεται για τη βελτίωση του «data motion network». Το «data motion network» είναι το Hardware που χρησιμοποιείται για συνδέσει την προγραμματιζόμενη εκτέλεση στο PS (Processing System) με τη Hardware function που είναι υλοποιημένη στο PL (Programmable Logic). Αυτό γίνεται με τη βοήθεια τωνdatamovers τύπων. Ωστόσο με αυτά που προσφέρουν μπορούν να βελτιώσουν την υλοποίηση τόσο για την ταχύτητα όσο και για τους πόρους του Hardware[18]. Τα datamotionnetworksπου δημιουργούνται για κάθε τύπο δεδομένων μπορεί να είναι:

- **Scalar:** Οι τύποι scalar μεταφέρονται πάντα με AXILITEdatamover. Αυτό αποτελεί ένα memory-mappedinterface το οποίο απαιτεί πολύ λίγους Hardware πόρους για υλοποίηση του, που αποτελείται από μεταφορές «read/write» με «hand shaking». Αυτό το καθιστά ιδανική επιλογή για τα ορίσματα της Hardware συνάρτησης τα οποία μεταφέρονται μόνο μία φορά ανά κλήση.

- **Πίνακας:** Οι πίνακες περιέχουν πολλαπλές τιμές δεδομένων και μεταφέρονται με πιο αποτελεσματικό τρόπο χρησιμοποιώντας τις γρήγορες μεταφορές του DMA. Υπάχουν διάφορες επιλογές για τον έλεγχο τόσο της αποδοτικότητας του Hardware όσο και των πόρων που χρησιμοποιούνται για την υλοποίηση του Hardware, επιτρέποντας ακόμη και την επιλογή συγκεκριμένων datamovers.

- **Δείκτες:** Ένας δείκτης (pointer) σε μια Hardware συνάρτηση θεωρείται τύπος scalar ακόμα και αν δείχνει σε κάποιο πίνακα. Στη περίπτωση όπου ο συγκεκριμένος δείκτης είναι written toor read πολλές φορές φορές τότε πρέπει να τοποθετηθεί ο ανάλογος τύπος datamover για να πραγματοποιηθεί μαζική μεταφορά δεδομένων.

- **Δομή ή Κλάση:** Στη δομή/κλάση (struct/class) κάθε datamember χρησιμοποιεί το δικό του datamover ανάλογα με το αν είναι Scalar ή Πίνακας. Για ένα Πίνακα δομής/κλάσης, το «datamotionnetwork» είναι το ίδιο συντάσσεται όπως τον Πίνακα παραπάνω. Αν είναι Scalar τότε δεν απαιτείται προηγουμένως κάποια εντολή.

Σημείωση: Για να εξασφαλιστεί η ευθυγράμμιση Software / Hardware interface, θα ήταν προτιμότερο τα ορίσματα του Hardware function με τύπο long ή πίνακας bool ή struct να μην χρησιμοποιούνται.

Κατανοώντας τι υπάρχει από προεπιλογή γίνεται δυνατή η επιλογή του βέλτιστου «data motion network» των δεδομένων ανάλογα με την υλοποίηση που επιθυμείται να υπάρξει για την εφαρμογή.

Το πρόγραμμα συνάγει τύπους Hardwareinterface για το κάθε όρισμα της συνάρτησης βασισμένο στο είδος του κάθε ορίσματος και στις εντολές pragmas .

5.1.1.3 Δεσμευση Χώρου Μνημης (memory allocation)

Υπάρχουν δύο τρόποι αποθήκευσης δεδομένων στη μνήμη :

1. είτε **paged** (όταν είναι malloc ή scalar)
2. είτε **physically contigoous** (sds_alloc() ήposix_memalign () ήsds_alloc() μαζίμετηνεντολή memory attribute)

Τρόποι για καλύτερη εκμετάλλευση της μνήμης:

Προσαρμόζοντας τους πίνακες σύμφωνα με τα όρια της σελίδας (page) ελαχιστοποιείται ο αριθμός των σελίδων που μεταφέρονται με τον DMA scatter-gather (για παράδειγμα, οι πίνακες που δεσμεύουν χώρο με τη malloc).

Για να δεσμευτεί χώρος μνήμης για τους πίνακες που μεταφέρονται από τον DMA μέσα στα όρια γραμμής της μνήμης cache (για caches L1 και L2) το sds_alloc() (εγγυάται φυσικά συνεχόμενο χώρο μνήμης) ή posix_memalign () (δέσμευση χώρου μνήμης μεγέθους bytes σε στοίχιση όσο είναι το ALIGMENT) αντί για malloc () (χωρίζεται σκόρπια στη μνήμη σε pages) είναι προτιμότερο όταν πρόκειται για δέσμευση χώρου μνήμης πινάκων.

Είναι πιθανό ότι λόγω της δομής του προγράμματος, ο compiler sdscc να μην μπορεί να συμπεράνει τη συνέχεια της μνήμης (πχ για ZERO COPY). Μπορεί όμως να ενημερώσει τον compiler ότι τα δεδομένα κατανέμονται σε μια φυσικά συνεχή μνήμη εισάγοντας την ακόλουθη εντολή πριν την δήλωση της Hardware συνάρτησης: #pragmaSDSdatamem_attribute

Το **sds_alloc πρέπει** οπωσδήποτε να χρησιμοποιηθεί για να δεσμευτεί χώρος μνήμης για έναν πίνακα στις τις ακόλουθες δύο περιπτώσεις:

1. Όταν χρησιμοποιείται datamoverZEROCOPY για κάποιο όρισμα πίνακα.
2. Όταν χρησιμοποιείται datamover για να κατευθυνθεί ο compiler του συστήματος να χρησιμοποιήσει το Simple-DMA ή το 2D-DMA.

5.1.2 Εντολή Resource

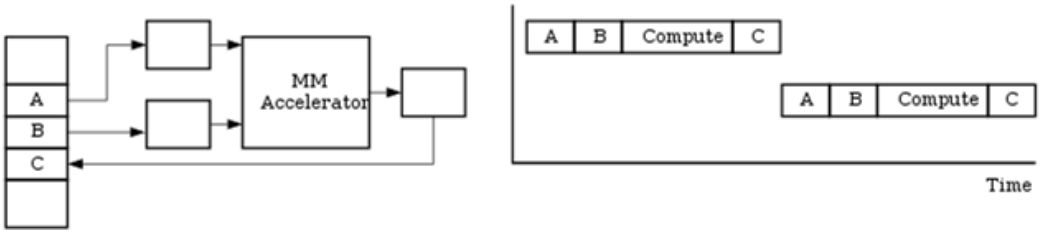
Για μεγαλύτερη αύξηση παραλληλισμού και συγχρονισμού χρησιμοποιώντας τον sdscc compiler. Υπάρχει η δυνατότητα να κατευθυνθεί ο compiler ούτως ώστε να δημιουργήσει πολλαπλά «instances» της Hardware function. Αυτό επιτυγχάνεται με το ακόλουθο pragma τοποθετώντας το ακριβώς πριν τη κλήση της συνάρτησης. Αυτή η εντολή δημιουργεί «Hardware instance» όπου αναφέρεται από το «id»

Ο «async» μηχανισμός δίνει τη δυνατότητα διαχείρησης των «Hardware threads» επίσης επιτυγχάνει πολύ υψηλά επίπεδα παραλληλισμού και συγχρονισμού, αλλά όπως κάθε «multi-threaded» μοντέλο, απαιτεί ιδιαίτερη προσοχή στις λεπτομέρειες του συγχρονισμού για να αποφευχθεί ακαθόριστη συμπεριφορά ή «deadlocks»[24].

5.1.3 Εντολή Task-Level Pipeline

Εάν υπάρχουν πολλαπλές κλήσεις προς έναν επιταχυντή σε μια εφαρμογή, τότε υπάρχει η δυνατότητα να διαμορφωθεί η εφαρμογή έτσι ώστε να γίνεται pipeline σε αυτές τις κλήσεις και το set-up και η μεταφορά δεδομένων να γίνονται στον επιταχυντή[33].

Το ακόλουθο σχήμα απεικονίζει (**Εικόνα 18:** Σειριακή Εκτέλεση των κλήσεων των Πινάκων) από αριστερά τον υπολογισμό των πινάκων και από δεξιά το χρονο-διάγραμμα υλοποίησης για δύο διαδοχικές κλήσεις τις οποίες εκτελεί διαδοχικά.

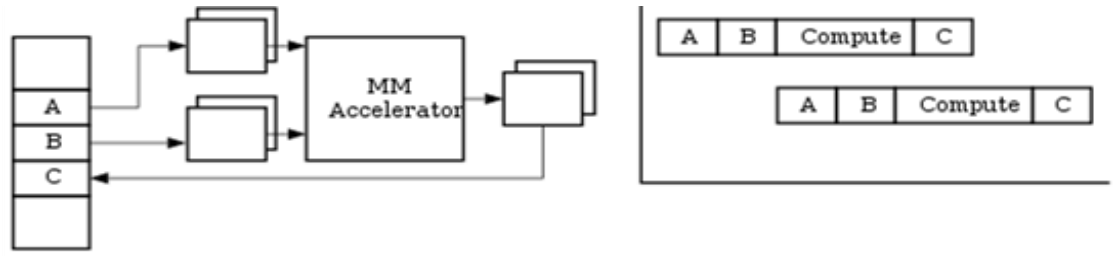


Εικόνα 18: Σειριακή Εκτέλεση των κλήσεων των Πινάκων

Η ακόλουθη εικόνα (**Εικόνα 19:** Εκτέλεση **Pipeline** των κλήσεων των Πινάκων) δείχνει να πραγματοποιούνται οι δύο κλήσεις εκτέλεσης ,σε pipeline διαμόρφωση.

Η μεταφορά δεδομένων στη δεύτερη κλήση ξεκινά ενώ η μεταφορά δεδομένων για τη πρώτη κλήση τελείωσε και συμπίπτει με τον υπολογισμό των δεδομένων της πρώτης κλήσης . Για να ενεργοποιήθει το pipelining ,χρειάζεται να υπάρχει περισσότερη τοπική μνήμη για να αποθηκευτεί το δεύτερο σετ των ορισμάτων ενώ ο accelerator υπολογίζει το πρώτο σετ ορισμάτων.

Το πρόγραμμα παράγει αυτές τις μνήμες υπό την καθοδήγηση του χρήστη. Αυτές οι μνήμες ονομάζονται multi-buffers.



Εικόνα 19: Εκτέλεση **Pipeline** των κλήσεων των Πινάκων

5.1.3.1 Hardware Buffer Depth

Απαραίτητη προϋπόθεση για να λειτουργήσει η εντολή Task Lelel Pipeling εκτός από το ότι πρέπει να καθοριστεί ως data mover ο DMA SCATTER GATHER για τη μεταφορά δεδομένων είναι ότι τα ορίσματα πρέπει να έχουν **Buffer Depth**[32]. Για την εντολή Buffer Depth ισχύουν τα εξής:

- Buffer Depth πρέπει να είναι compile-time σταθερή τιμή.
- Αυτή η εντολή ισχύει μόνο σε πίνακες που αντιστοιχίζονται σε BRAM ή FIFOinterfaces. Για ένα BRAM-mapped πίνακα, η τιμή καθορίζει το βάθος του Hardware multi-buffer (δηλαδή τον αριθμό των πινάκων που δημιουργούνται) .
- Για ένα FIFO-mapped πίνακα, η τιμή καθορίζει το βάθος του HardwareFIFO που έχουν δεσμεύσει χωρο για ένα πίνακα.

  Για τη χρήση αυτού του pragmaπρέπει να ισχύουν τα ακόλουθα :

- BRAM: $1 \leq$ Depth ≤ 4, and $2 \leq$ ArraySize ≤ 16384.
- FIFO: BufferDepth = $2n$, where $4 \leq n \leq 20$.

5.2 Εντολές Hight Level Synthesis (HLS)

Οι εντολές pragmaHLS σε αντίθεση με τις SDSpragma εντολές ως στόχο έχουν τη βελτιστοποίηση του εσωτερικού του επιταχυντή , για ό,τι αφορά το εξωτερικό Hardware του επιταχυντή χρησιμοποιούνται οι SDSpragma εντολές. Το Vivado HLS παρέχει pragmas που μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση του σχεδιασμού: μείωση της καθυστέρησης, βελτίωση της απόδοσης και μείωση της χρήσης πόρων και πόρων του προκύπτοντος κώδικα RTL. Αυτά τα pragmas μπορούν να προστεθούν απευθείας στον κώδικα για τον πυρήνα. Στο περιβάλλον SDSoC οι SDSoC μεταγλωττιστές ορίζουν αυτόματα HLS pragmas βάσει των εσωτερικών λειτουργιών και απαιτήσεων του Hardware. Αυτή η αυτόματη διαδικασία είναι απενεργοποιημένη αν προστεθούν χειροκίνητα pragmas HLS που έρχονται σε σύγκρουση με τις pragmas που επιλέγονται από το sdscc / sds ++[30].

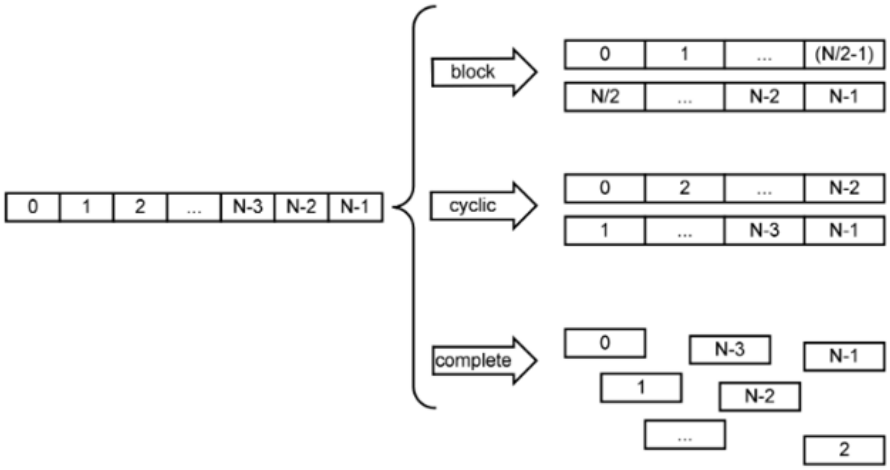
5.2.1 Εντολή Array Partition

Με το Array Partition pragmas οι πίνακες μπορούν να κατατμηθούν σε μικρότερα σετ πινάκων όπως προαναφέρθηκε παραπάνω με αυτόν τον τρόπο υπάρχει η δυνατότητα να διαβάζονται ή να γράφονται παράλληλα περισσότερα στοιχεία[29]. Αυτό μπορεί να γίνει με τρεις τρόπους :

1. **block**: Ο αρχικός πίνακας χωρίζεται σε ίσου μεγέθους blocks διαδοχικών στοιχείων.
2. **cyclic**: Ο αρχικός πίνακας χωρίζεται σε ίσου μεγέθους blocks, τα οποία διαπλεκόνται.
3. **complete**: Η προκαθορισμένη λειτουργία είναι να χωρίσει το πίνακα σε ατομικά στοιχεία.

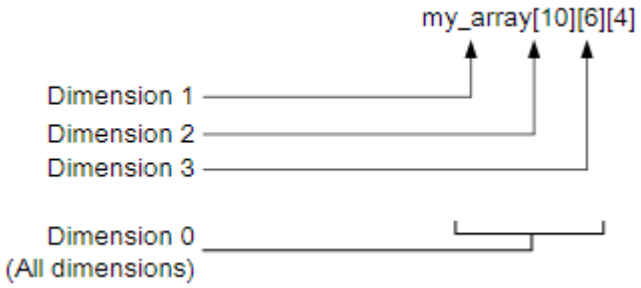
Αυτό αντιστοιχεί στην υλοποίηση ενός πίνακα ώς μια συλλογή από καταχωρητές (registers) αντί να συλλέξει χώρο στη μνήμη κάτι το οποίο έκαναν οι προηγούμενοι δύο μέθοδοι.

Η ακόλουθη εικόνα (**Εικόνα 20**:Array Partitioning) εξηγεί περαιτέρω τις δυνατότητες των τριών μεθόδων.



Εικόνα 20: Array Partitioning

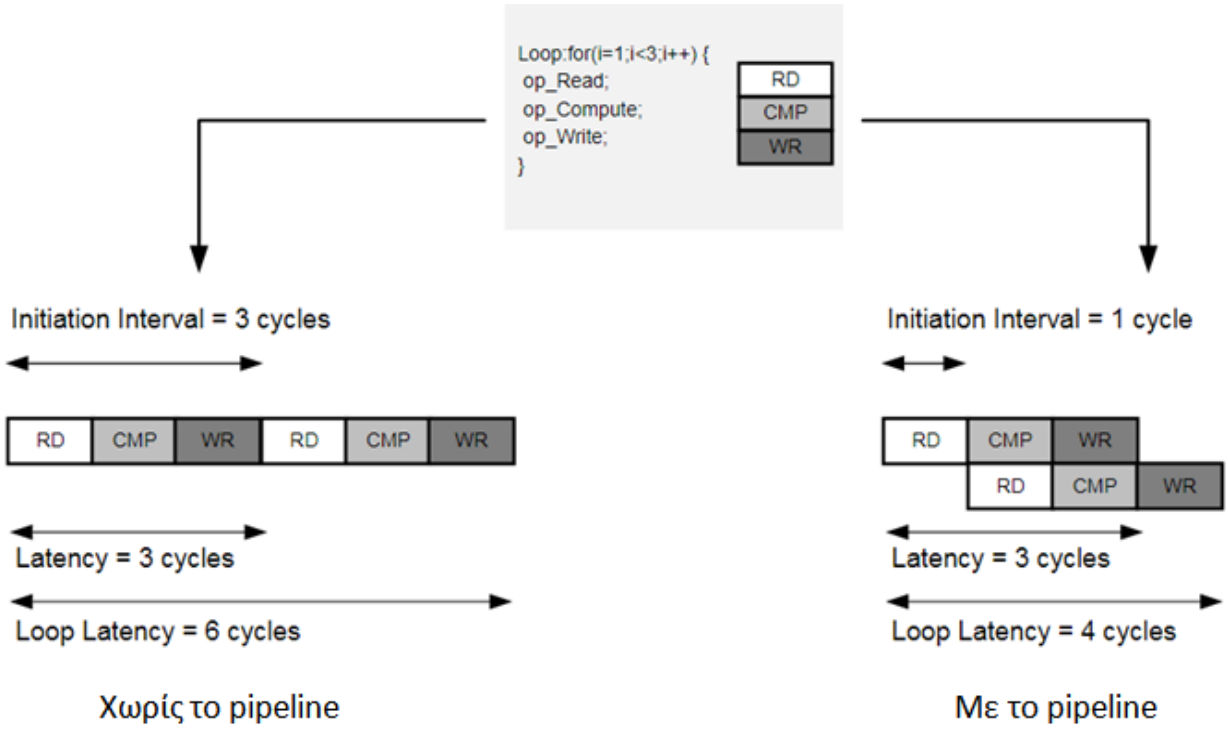
Όταν χωρίζονται πολυδιάστατοι πίνακες η dim (dimension) επιλογή χρησιμοποιείται για να καθορίσει ποια διάσταση επιθυμείται να χωριστεί. Το ακόλουθο σχήμα (**Εικόνα 21:** Multi-dimention Array Partitioning) απεικονίζει τον τρόπο διαχωρισμού ενός πολυδιάστατου πίνακα.



Εικόνα 21: Multi-dimension Array Partitioning

5.2.2 Εντολή Loop Pipeline

Με τη loop pipeline εντολή βελτιώνεται η απόδοση της Hardware συνάρτησης δημιουργώντας παραλληλισμό μεταξύ των λειτουργιών των επαναλήψεων του βρόγχου (loop). Σε σειριακές γλώσσες οι λειτουργίεςτων επαναλήψεων ενός βρόγχου (loop) εκτελούνται σειριακά και ηεπόμενη επανάληψη τουβρόγχου ξεκινάει εφόσον ολοκληρωθεί η προηγούμενη. Η **loop pipeline** δίνει τη δυνατότητα οι λειτουργίες μιας επανάληψης να γίνουν με ταυτόχρονο τρόπο όπως φαίνεται και στην ακόλουθη εικόνα (**Εικόνα 22:** Loop Pipeline).



Εικόνα 22: Loop Pipeline

Χωρίς το pipelining, χρειάζεται 3 κύκλους ρολογιού για τη πρώτη επανάληψη και άλλους 3 για τη δεύτερη επανάληψη δηλαδή στο σύνολο θα χρειαστούν 6 κύκλοι ρολογιού για ολες τις επαναλήψεις.

Εν αντιθέσει με την εισαγωγή του pipelining στο βρόγχο, η διαφορά της πρώτης επανάληψης σε σχέση με τη δεύτερη είναι μόνο ένας κύκλος , δεν απαιτείται δηλαδή να τελειώσουν όλες οι λειτουργίες της πρώτης επανάληψης για να ξεκινήσει η δεύτερη. Όσο γίνεται στη πρώτη loop ο υπολογισμός υπάρχει η δυνατότητα να διαβάσει τα επόμενα δεδομένα κ.ο.κ.

Υπάρχει δυνατότα καθορισμού των κύκλων μέσω του «**Initiation Interval**» **(II)** όπου καθορίζεται ο αριθμός των κύκλων εκτέλεσης των λειτουργιών. Στην περίπτωση που δεν καθοριστεί το πρόγραμμα θα κρίνει από μόνο του πόσοι κύκλοι χρειάζονται για τη κάθε λειτουργία[28].

5.2.3 Εντολή Loop Unroll

Η εντολή Loop Unroll (ξετύλιγμα βρόγχου) παραλληλίζει τις λειτουργίες των επαναλήψεων του βρόγχου μεταξύ τους. Δημιουργεί πολλαπλά αντίγραφα των λειτουργιών του βρόγχου ρυθμίζοντας τον «**factor**» μετρητή ανάλογα με τη περίπτωση.

Ξετυλίγοντας ένα βρόγχο με ένα factorN στην ουσία δημιουργεί N αντίγραφα του εσωτερικού της επανάληψης.

Η loop unroll εντολή δημιουργεί περισσότερες λειτουργίες σε κάθε επανάληψη βρόχου, με αυτό τον τρόπο επιτυγχάνεται μεγαλύτερος παραλληλισμός μεταξύ των λειτουργιών. Μεγαλύτερος παραλληλισμός σημαίνει περισσότερα throughput (ρυθμός μετάδοσης/επεξεργασίας δεδομένων) καθώς και υψηλότερη απόδοση συστήματος. Αν το factorN είναι λιγότερο από το συνολικό αριθμό των επαναλήψεων βρόγχου ,τότε ονομάζεται "partial unroll" (μερικό ξετύλιγμα). Αν το factorN είναι όσο και ο αριθμός των επαναλήψεων του βρόγχου, αυτό ονομάζεται "full unroll" (πλήρης ξετύλιγμα). Προφανώς, το "full unroll" απαιτεί τα όρια του βρόγχου να είναι γνωστά σε compile time αλλά με αυτόν τον τρόπο αυξάνεται ο παραλληλισμός[25].

```
int sum = 0;
for(int i = 0; i < 10; i++) {
    sum += a[i];
}
```

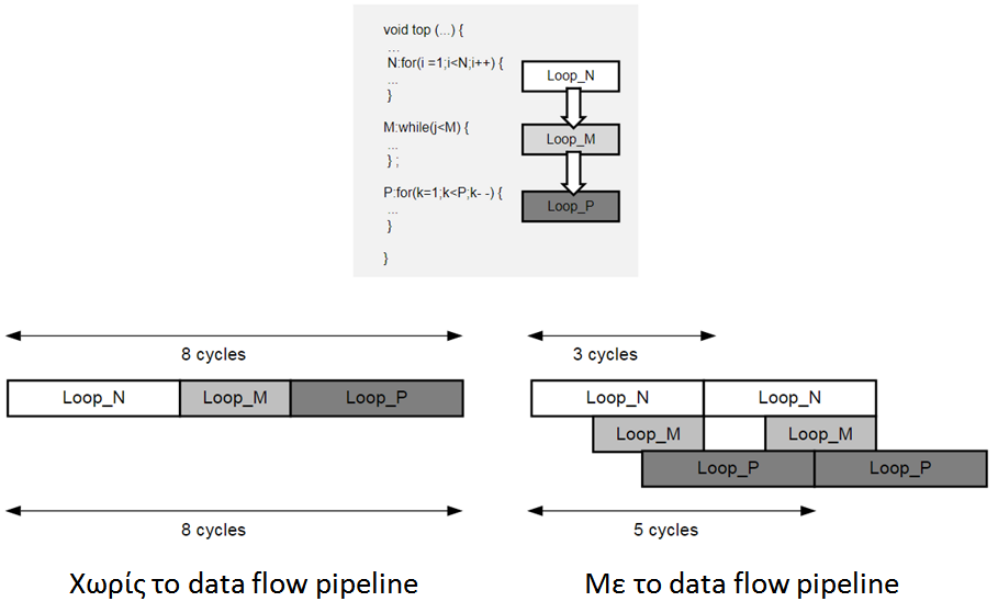
Χωρίς Unroll

```
int sum = 0;
for(int i = 0; i < 10; i+=2) {
    sum += a[i];
    sum += a[i+1];
}
```

Με Unroll και factor=2

5.2.4 Εντολή Data Flow Pipeline (για Βρόγχους)

Το Data Flow Pipelining για βρόγχους όταν εφαρμόζεται δημιουργεί pipeline σε βρόχους. Χωρίς το data flow pipelining, ο βρόχος N πρέπει να εκτελέσει και να ολοκληρώσει όλες τις επαναλήψεις πριν ξεκινήσει ο βρόχος M. Το ίδιο ισχύει για τη σχέση μεταξύ βρόχων M και P. Στο παράδειγμα παρακάτω (**Εικόνα 23:** Data Flow Pipelining), χρειάζεται οκτώ κύκλους προτού ο βρόχος N μπορέσει να ξεκινήσει την επεξεργασία της επόμενης τιμής και οκτώ κύκλους προτού γραφτεί η έξοδος (υποθέτοντας ότι η έξοδος γράφεται όταν ο βρόχος P τελειώνει)



Εικόνα 23: Data Flow Pipelining

Με το Data Flow Pipelining οι βρόχοι μπορούν να λειτουργούν ταυτόχρονα. Αν ο βρόχος M παραπάνω χρειάζεται 3 κύκλους για να εκτελεστεί, ο κώδικας μπορεί να δέχεται νέες εισόδους κάθε τρεις κύκλους. Ομοίως, μπορεί να παράγει μια τιμή εξόδου κάθε πέντε κύκλους, χρησιμοποιώντας τους ίδιους Hardware πόρους. Εισάγονται αυτόματα κανάλια μεταξύ των βρόχων για να διασφαλιστεί ότι τα δεδομένα μπορούν να ρέουν ασύγχρονα από το ένα βρόχο στο επόμενο.

Ενεργοποιεί το task-level pipelining, επιτρέποντας οι συναρτήσεις και επαναλήψεις να εκτελεστούν

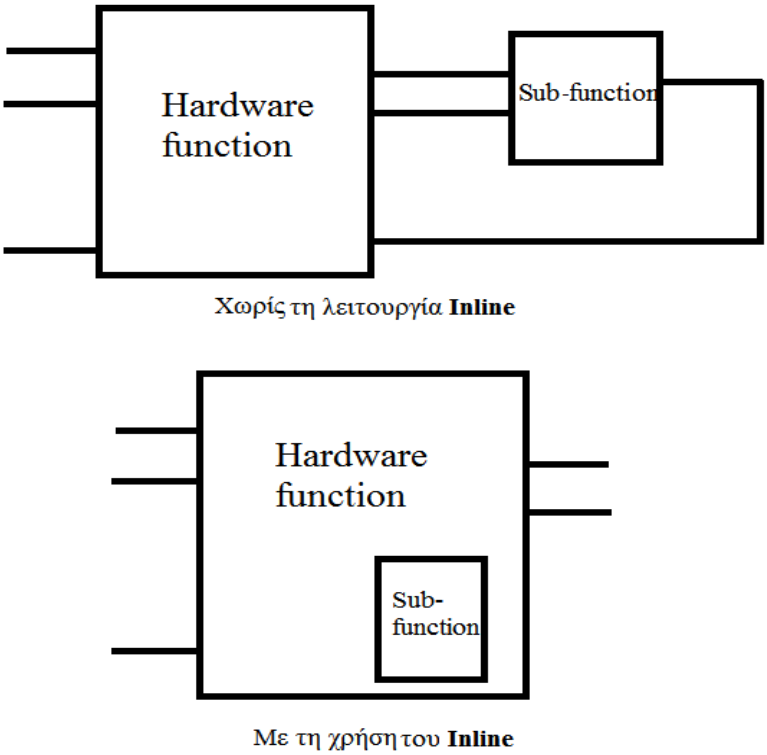
σχεδόν ταυτόχρονα. Μπορεί να χρησιμοποιηθεί το interval όπως στοloop pipeline για τον καθορισμό των κυκλων.

Το Data flow pipelining ενεργοποιεί ταυτόχρονη λειτουργία σε μια ακολουθία **βρόχων ή συναρτήσεων**, που εκτελούνται συνήθως διαδοχικά .Το Data flow pipelining εφαρμόζεται είτε σε συνάρτηση, είτε σε βρόχο είτε σε μια περιοχή που περιέχει όλες τις συναρτήσεις είτε όλους τους βρόχους. Η εντολή δεν ισχύει σε ένα πεδίο που περιέχει βρόχους και συναρτήσεις(δηλαδή ή το ένα ή το άλλο)[26].

5.2.5 Εντολή Inline

Παρόμοια με την inlining Software συνάρτηση, το inline μπορεί να είναι ωφέλιμο στις Hardware συναρτήσεις.

Η συνάρτηση inlining μπορεί να αντικαταστήσει μια κλήση συνάρτησης. Δημιουργεί ένα αντίγραφο του του εσωτερικού της συνάρτησης. Μετά από αυτό, η συνάρτηση που έχει χρησιμοποιήσει το inline δεν εμφανίζεται πλέον ως ξεχωριστό επίπεδο ιεραρχίας. Η inlining συνάρτηση επιτρέπει λειτουργίες εντός της λειτουργίας inline να βελτιωθούν περισσότερο με τις περιβάλλουσες λειτουργίες, με αυτό το τρόπο βελτιώνει τη συνολική πιθανόν λανθάνουσα κατάσταση[27].



Εικόνα 24: Λειτουργία Inline

6. Υλοποίηση του k-NN Αλγορίθμου σε Hardrware

Ο αλγόριθμος k-NN παρακάτω έχει ελεγχθεί στο πρόγραμμα Xilinx SDSoc 2016.3 το οποίο είναι ένα Eclipse-based ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) για ετερογενείς ενσωματωμένα συστήματα Zynq-7000 All Programmable SoC. Η πλακέτα που χρησιμοποιήθηκε είναι το ZedBoard όπου είναι ένα πλήρες περιβάλλον ανάπτυξης χρησιμοποιώντας το Xilinx Zynq-7000 All Programmable SoC.

Η συχνότητα του επεξεργαστή είναι στα 666 MHz. Η συχνότητα του Data Motion είναι στα 142.6 MHz. Το λειτουργικό σύστημα που χρησιμοποιήθηκε είναι LinuxSMP (Zynq 7000) PetaLinux.

Το πρόγραμμα εφόσον εκτελεστεί ο αλγόριθμος περιλαμβάνει κάποιες αναφορές κάποιες από τις οποίες χρησιμοποιήθηκαν (Data Motion Report ,Accelerator Callsites ,Ποσοστό χρήσης πόρων του fpga).

Ο αλγόριθμος k-NN θα εξεταστεί σε διάφορες φάσεις(σε Software και Hardware) οι οποίες εξαρτώνται από μέγεθος των δεδομένων που εισέρχονται στησυνάρτηση. Παρακάτω θα εξεταστεί ποιος είναι ο βέλτιστος τρόπος διαμόρφωσης του αλγορίθμου ανάλογα την περίπτωση και φυσικά θα σύγκριθεί η Hardwareαπόδοση σε σχέση με τη Software.

6.1 Τύποι Ορισμάτων της Hardware Συνάρτησης (Επιλογή Data Mover)

Υπάρχουν διάφοροι τύποι ορισμάτων όπως προαναφέρθηκε .Σε αυτό το υποκεφάλαιο θα διαπιστωθεί στη πράξη ποίος data mover είναι ο καταλληλότερος για να χρησιμοποιηθεί ανάλογα τον όγκο των δεδομένων. Οι data mover ασφαλώς και παίζουν πολύ σημαντικό ρόλο στην απόδοση του αλγορίθμου.

Μια επιλογή ενός λάθους datamover πιθανόν να κοστίσει πολύ στην απόδοση. Αυτό θα διαπιστωθεί και παρακάτω στα παραδείγματα που ακολουθούν.

6.1.1 Μεταφορά Μικρού Σετ Δεδομένων

Παρακάτω ακολουθούν διάφορα παραδείγματα τα οποία σχετίζονται με την επιλογή datamover για μικρό σετ δεδομένων. Τα δεδομένα είναι 100 (100*4*8=3200 bytes) ταξινομημένα δείγματα και 50 (50*4*8=1600 bytes) προς ταξινόμηση. Τα δεδομένα είναι σχετικά λίγα άρα δεν θα υπάρχει κάποιο πρόβλημα σχετικά με τη μνήμη του fpga , δεν υπάρχει κάποιος περιορισμός όσον αφορά τους datamovers η οπότε η επιλογή του datamover σχετίζεται με την απόδοση του datamover και τις προϋποθέσεις που απαιτεί ο καθένας (πχ για να χρησιμοποιηθεί ο AXIDMASIMPLE πρέπει προηγουμένως τα δεδομένα να είναι αποθηκευμένα σε συνεχόμενη μνήμη).

1ος Τρόπος Υλοποίησης της Συνάρτησης (ZERO COPY):

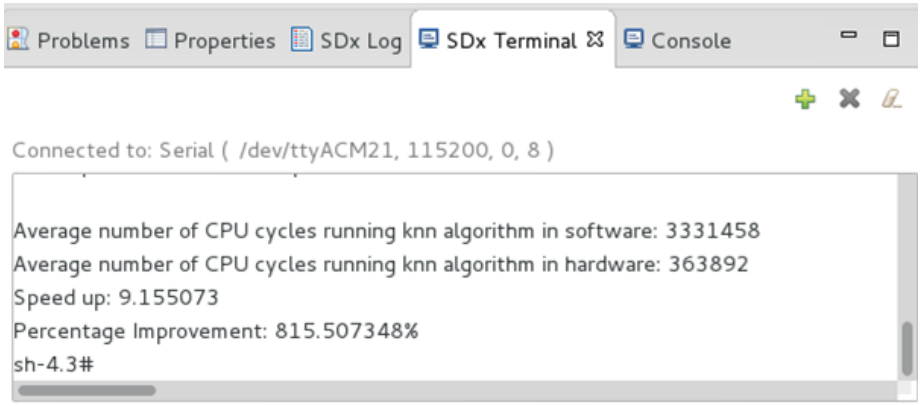
Για ένα πίνακα ο οποίος είναι «read » και «write» (INOUT) στη Hardwarefunction, πρέπει απαραιτήτως να χρησιμοποιηθεί η εντολή **#pragma SDS datazero_copy**(η οποία αντιστοιχίζει το όρισμα σε κοινή μνήμη) ούτως ώστε να πει στον compiler ότι ο πίνακας πρέπει να κρατηθεί σε κοινή μνήμη και όχι απλά να αντιγράφεται. Η εντολή #pragma SDS datazero_copy απαιτεί η μνήμη να είναι **PHYSICAL_CONTIGUOUS** (**#pragma SDS data mem_attribute**: η συγκεκριμένη εντολή εισάγεται διότι ορισμένες φορές το πρόγραμμα αδυνατεί να κατανοήσει τη συνέχεια της μνήμης)άρα ο πίνακας πρέπει να δηλωθεί με **sds_alloc()** (δεσμεύει συνεχόμενο χώρο στη μνήμη όσων bytes

ορίζονται εξ αρχής). Με τα παρακάτω pragmas τοποθετώντας τα ακριβώς πριν τη δήλωση της συνάρτησης επιτυγχάνεται η κοινή μνήμη:

ZERO COPY:

```
1 #pragma SDS data zero_copy (samples)
2 #pragma SDS data mem_attribute (samples: CACHEABLE|PHYSICAL_CONTIGUOUS)
3 voidknn_algorithm_hw(doublesamples [150*4], int a, int b);
```

Η απόδοση στις μεταφορές «burst» (**μεγάλος όγκος δεδομένων**) είναι αρκετά καλή, ωστόσο πρέπει να σημειωθείότι με αυτόν τρόπο η καθυστέρηση στην εξωτερική DDR από το PL είναι αρκετά υψηλότερη απ' ότι είναι στη CPU. Αυτό είναι κάτι ανεπιθύμητο, η απόδοση της Hardware function δεν θα είναι και τόσο καλή όσο θα μπορούσε να είναι με κάποιο άλλο datamover (αυτό θα φανεί στη συνέχεια). Στην παρακάτω εικόνα (**Εικόνα 25**: Απόδοση Συστήματος) περιγράφεται η απόδοση του συστήματος η οποία είναι σχετικά καλή (815% βελτίωση σε σχέση με τη Software συνάρτηση όμως τα δείγματα είναι μόνο 150).



Εικόνα 25:Απόδοση Συστήματος (ZERO COPY)

Στον πρώτο πίνακα παρακάτω(**Πίνακας 4:** DataMotionReport (ZERO COPY)) αναγράφονται αναλυτικά το μέγεθος των δεδομένων που μεταφέρθηκαν, ο τρόπος με τον οποίο μεταφέρθηκαν τα δεδομένα, τα ορίσματα αν λειτουργούν ως in ή out ή inout. Στη δεύτερη στήλη από δεξιά, τη στήλη «Pragmas» αναγράφονται οι εντολές pragmas που εισήχθησαν ,ενώ στη στήλη « Direction Declared

Size (bytes)» αναγράφεται το μέγεθος των δεδομένων που μεταφέρθηκαν για τον πίνακα (150 *4*8).

Στον δεύτερο πίνακα παρακάτω (**Πίνακας 5:** Accelerator Callsites) στη στήλη «CallSite» αναγράφονται οι κλήσεις (1) προς τον επιταχυντή (accelerator) που δημιουργήθηκαν ,ο τρόπος με τον οποίο είναι τοποθετημένα στη μνήμη τα δεδομένα καθώς και πόσοι επιταχυντές υπάρχουν (1).Εδώ δείχνει το μέγεθος μεταφοράς της Hardware function στην στήλη «Transfer Time (CPUcycles)»το οποίο είναι 6991.Στον **Πίνακα 6:** Ποσοστό χρήσης πόρων του fpga (ZERO COPY) αναγράφονται πόσες δεσμεύσεις έγιναν στο χώρο του fpga. Λόγω του τρόπου υλοποίησης της συνάρτησης δημιουργούνται αρκετά LUTs και FF. Αυτό προς το παρόν δεν αποτελεί πρόβλημα εφόσον είναι εντός ορίων στη συνέχεια όμως θα διορθωθεί.

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw1	samples	samples	INOUT	600*8	• mem_attribute:CACHEABLE.PHYSICAL_CONTIGUOUS • data_mover:zero_copy	ps7_S_AXI_ACP:AXIMM:0xC

Πίνακας 4: Data Motion Report (ZERO COPY)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw1	mpliax.cpp:129:3	samples	4800	PHYSICAL_CONTIGUOUS	489	6991

Πίνακας 5: Accelerator Callsites (ZERO COPY)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	36	31	34767	52525
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	36	31	34767	52525
Available	280	220	106400	53200
Utilization (%)	12	14	32	98

Πίνακας 6: Ποσοστό χρήσης πόρων του fpga (ZERO COPY)

2ος Τρόπος Υλοποίησης της Συνάρτησης (AXI_DMA_SG ,AXI_DMA, SIMPLE):

Ο τρόπος με τον οποίο γίνεται η μεταφορά δεδομένων από τη μνήμη εξαρτάται από το τρόπο με τον οποίο δεσμεύτηκαν στη μνήμη (όπως προαναφέρθηκε παραπάνω). Αν πχ δεσμευτεί χώρος στη μνήμη με τη malloc() (δηλαδή όχι απαραίτητα συνεχής) τότε ο κατάλληλος data mover θα ήταν ο DMA τύπου scatter-gather ενώ αν ο χώρος που δεσμεύεται στη μνήμη είναι συνεχής (sds_alloc()) τότε ο πιο κατάλληλος DMA θα είναι τύπου είτε SIMPE είτε 2D (για δισδιάστατους πίνακες).

Στον πίνακα παρακάτω (**Πίνακας 7:** Επιλογή DataMover) αναγράφονται αναλυτικά τα χαρακτηριστικά των παραπάνω datamovers (τύπος της μνήμης και το μέγεθος των ορισμάτων) .

Τοποθετώντας τα παρακάτω pragmas ακριβώς πριν τη δήλωση της συνάρτησης επιτυγχάνεται η μεταφορά δεδομένων με τους DMAs:

Data Mover	Physical Memory Contiguity	Data Size (bytes)
AXIDMA_SG	Either	> 300
AXIDMA_Simple	Contiguous	< 8M
AXIDMA_2D	Contiguous	< 8M
AXI_FIFO	Non-contiguous	< 300

Πίνακας 7: Επιλογή Data Mover

Σημείωση : Από δω και το εξής θα προστεθεί ένα ακόμα όρισμα (group) το οποίο θα λειτουργεί ώς έξοδος ενώ το όρισμα samples πλέον θα είναι μόνο IN διότι πλέον η μνήμη δεν θα είναι κοινή.

Η σύνταξη του κώδικα με DMA scatter gather είναι η εξής:

DMA SCATTER-GATHER:

```
1  #pragma SDS data data_mover (samples: AXIDMA_SG, group: AXIDMA_SG)
2  void knn_algorithm_hw (doublesamples [150*4], doublegroup [50], int a, int b);
```

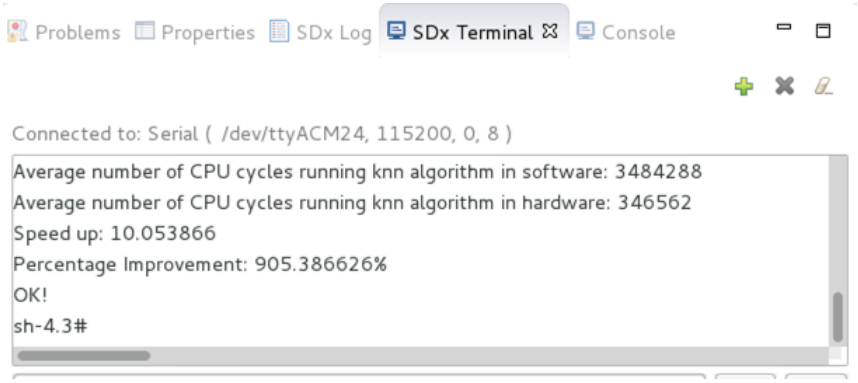
Η σύνταξη του κώδικα με DMA simple είναι η εξής:

DMA SIMPLE:

```
1  #pragma SDS data data_mover (samples: AXIDMA_SIMPLE, group: AXIDMA_SIMPLE)
2  void knn_algorithm_hw (doublesamples [150*4], doublegroup [50], int a, int b);
```

Τα ορίσματα είναι προκαθορισμένα να περνούν από την ACP port (εκτός αν είναι scalar τότε περνούν από τη GP0 port) οπότε αυτό δεν χρειάζεται να καθοριστεί. Παρακάτω αναγράφεται η απόδοση του

συστήματος (**Εικόνα 26:** Απόδοση Συστήματος (DMA SCATTER GATHER και **Εικόνα 27:** Απόδοση Συστήματος (DMA SIMPLE)) ανάμεσα σε Software και Hardware συνάρτηση (905% βελτίωση με τον DMASCATTERGATHER ενώ με τον DMASIMPLE 980%).Οι Hardware συναρτήσεις φαίνονται να έχουν σαφώς καλύτερη απόδοση σε σχέση με τη ZERO COPY εντολή. Ο DMA SCATTER GATHER βέβαια απαιτεί περισσότερα resources σε σχέση με τον DMA SIMPLE (**Πίνακας 10:** Ποσοστό χρήσης πόρων του fpga (DMA SCATTER GATHER)). Στους πίνακες : **Πίνακας 8:** Data Motion Report (DMA SCATTER GATHER) και **Πίνακας 11:** Data Motion Report (DMA SIMPLE) από τη στήλη «pragma» αναφέρονταιοι data movers που επιλέχθηκαν σε κάθε περίπτωση.



Εικόνα 26:Απόδοση Συστήματος (DMA SCATTER GATHER)

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples_PORTA	IN	600*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	a	a	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	b	b	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	group	group_r_PORTA	OUT	50*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG

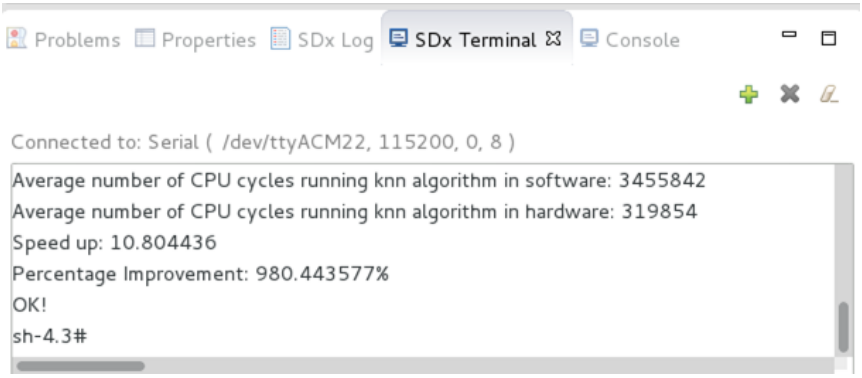
Πίνακας 8: Data Motion Report (DMA SCATTER GATHER)

Πίνακας 9: Accelerator Callsites (DMA SCATTER GATHER)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	sg.cpp:141:3	samples_PORTA	4800	paged	23994	13077
		a	4	paged	0	13
		b	4	paged	0	13
		group_r_PORTA	400	paged	16492	5954

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	32	31	34859	50065
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	32	31	34859	50065
Available	280	220	106400	53200
Utilization (%)	11	14	32	94

Πίνακας 10: Ποσοστό χρήσης πόρων του fpga (DMA SCATTER GATHER)



Εικόνα 27:Απόδοση Συστήματος (DMA SIMPLE)

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples_PORTA	IN	600*8	• data_mover:AXIDMA_SIMPLE	ps7_S_AXI_ACP:AXIDMA_SIMPLE
	a	a	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	b	b	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	group	group_r_PORTA	OUT	50*8	• data_mover:AXIDMA_SIMPLE	ps7_S_AXI_ACP:AXIDMA_SIMPLE

Πίνακας 11: Data Motion Report (DMA SIMPLE)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	mpliax.cpp:135:3	samples_PORTA	4800	contiguous	1113	6438
		a	4	paged	0	13
		b	4	paged	0	13
		group_r_PORTA	400	contiguous	1118	1433

Πίνακας 12: Accelerator Callsites (DMA SIMPLE)

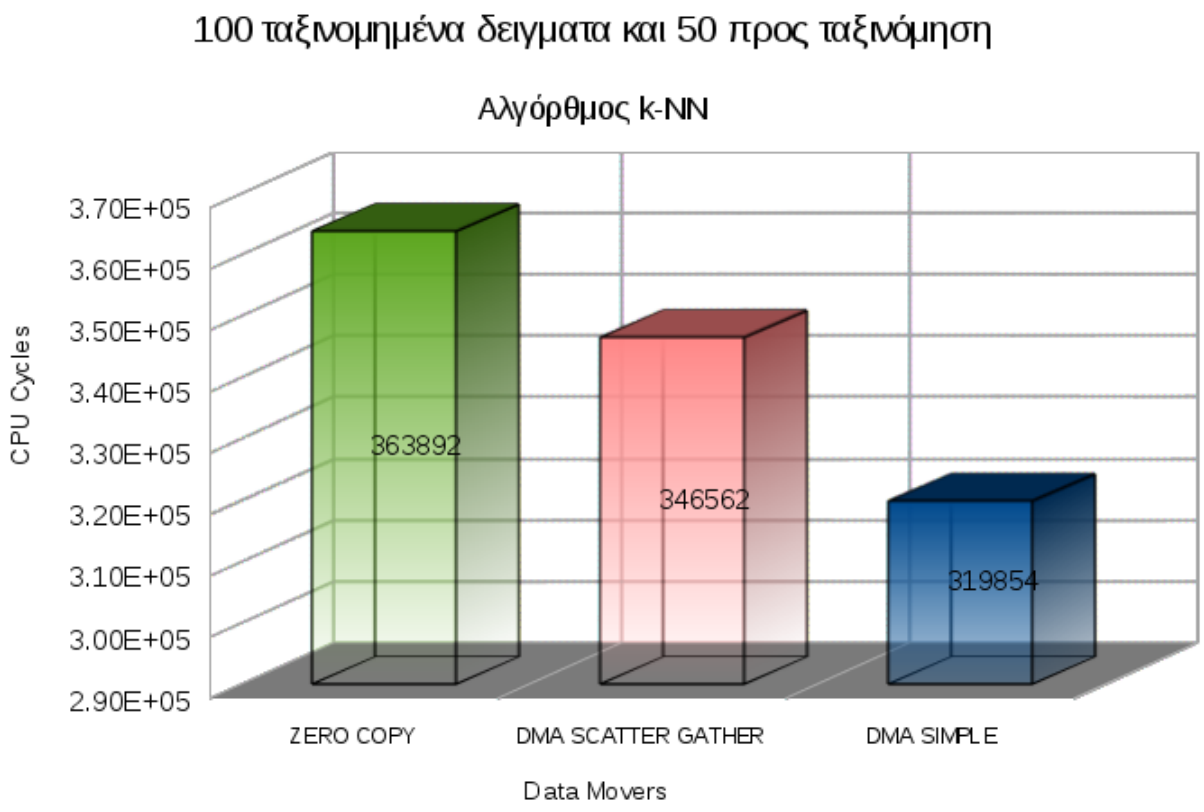
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	32	31	34859	50065
Memory	-	-	-	-
Multiplexer	-	-	-	-
Register	-	-	-	-
Total	32	31	34859	50065
Available	280	220	106400	53200
Utilization (%)	11	14	32	94

Πίνακας 13: Ποσοστό χρήσης πόρων του fpga (DMA SIMPLE)

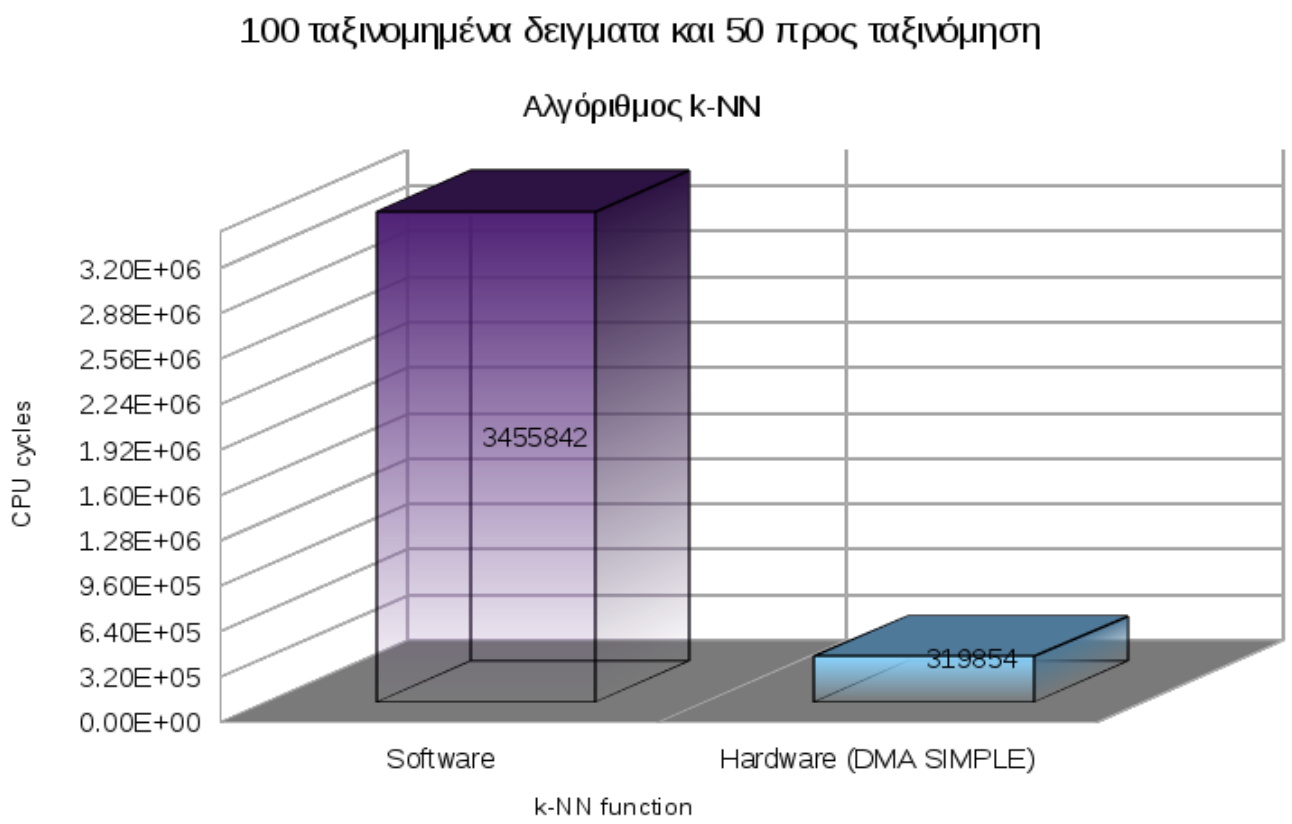
Ένα πλεονέκτημα του DMA είναι ότι παίρνει μαζικά πολλά δεδομένα και κάνει **ένα**hand-shake γι'αυτά και όσο τα διαβάζει ο επιταχυντής υπάρχει η δυνατότητα να παρει ο DMA τα επόμενα δεδομένα. Ο ZERO COPY όταν χρησιμοποιείται χρησμοποιεί ως data mover τον ίδιο τον accelerator. Ο accelerator δημιουργεί **ένα κανάλι** memory map interface για τη μεταφορά των δεδομένων.Η διαφορά του ZERO COPY είναι ότι κάνει πολλά περισσότερα **hand-shake (**4 κύκλους για κάθε **hand-shake)** εν αντιθέσει με τον DMA ο οποίος διαθέτει ένα buffer εσωτερικά,με τον οποίο παίρνει μαζικά κάποια δεδομένα και όσο στέλνονται τα πρώτα δεδομένα υπάρχει η δυνατότητα των επόμενων δεδομένων, κάθε φορά που ο επιταχυντής ζητάει δεδομένα. Όταν ο επιταχυντής ζητάει τα δεδομένα, τα δεδομένα απλα μεταφέρονται από τους buffers μέσω ενός stream καναλιού (2 cpu κυκλους). Ο ZERO COPY για burst μεταφορές.

Παρακάτω στο γράφημα παρατηρείται η διαφορά ανάμεσα στους data movers που χρησιμοποιήθηκαν (**Εικόνα 28:** Σύγκριση μετρήσεων του k-NN αλγορίθμουμε διαφορετικούς data movers (ZERO COPY, DMAS CATER GATHER, DMASIMPLE)). Συγκριτικά η διαφορά στους Data Movers με DMA και στο ZERO COPY είναι μεγάλη συνεπώς για λίγα δεδομένα ο ZERO COPY δεν συνίσταται!

Τέλος στην **Εικόνα 29:** Σύγκριση απόδοσης του του k-NN αλγορίθμου (Software Vs Hardware) απεικονίζεται η βελτιωμένη απόδοση του αλγορίθμου k-NN στο Hardware σε σχέση με το Software. Για τη σύγκριση χρησιμοποιήθηκε η επιλογή data mover με την καλύτερη απόδοση ,ο DMA SIMPLE (980% βελτίωση σε σχέση με τη Software συνάρτηση).



Εικόνα 28: Σύγκριση μετρήσεων του k-NN αλγορίθμουμε διαφορετικούς data movers (ZERO COPY, DMA SCATER GATHER, DMASIMPLE)



Εικόνα 29: Σύγκριση απόδοσης του του k-NN αλγορίθμου (Software Vs Hardware)

6.1.2 Έλεγχος του Όγκου των Δεδομένων που Μεταφέρονται (για Μεγάλο Όγκο Δεδομένων Με Εξάρτηση)

Η μεταφορά μεγάλου μεγέθους δεδομένων αποτελεί περισσότερο από ένα βασικό πρόβλημα στο Hardware λόγω του περιορισμένου αποθηκευτικού χώρου που διαθέτει η προγραμματιζόμενη λογική του ενσωματωμένου συστήματος. Η μεταφορά ενός μεγάλου μεγέθους δεδομένων πίνακα ορισμένες φορές είναι απαραίτητη όπως αυτό συμβαίνει και στη περίπτωση του k-NN αλγορίθμου. Ο αλγόριθμος όπως προαναφέρθηκε μπορεί να είναι ένας απλός αλγόριθμος και εύκολος στη κατανόηση απαιτεί όμως μεγάλη μνήμη ,καθώς πρέπει να αποθηκευτούν όλα τα δεδομένα εκπαίδευσης. Αν τα δεδομένα είναι πολλά καιπρέπει να περάσουν από μέσα από τη Hardware συνάρτηση πιθανόν να υπάρξει πρόβλημα επειδή τα δεδομένα ίσως να μην χωρούν στην τοπική μνήμη του fpga η οποία δεν είναι πολύ μεγάλη (18K). Αυτό αποτελεί πρόβλημα εφόσον όλα τα δείγματα ενδέχεται να παίζουν ρόλο για την κατηγοριοποίηση ενός μη ταξινομημένου δείγματος (αυτό σημαίνει ότι υπάρχει εξάρτηση των δεδομένων) .

Μια λύση σ'αυτό το πρόβλημα η οποία εφαρμόζεται παρακάτω είναι η αλλαγή της δομής του αλγορίθμου για να επεξεργάζονται τα δεδομένα λίγα-λίγα (π.χ. ανά 1000 εφόσον πλέον το training set αποτελείται από 10.000 δείγματα) καθώς και η τοποθέτηση των κατάλληλων εντολών ώστε να μπορεί να «διαβάζει» και να «γράφει» (read/write) δεδομένα σωστά και γρήγορα. Στην περίπτωση όπου τα δεδομένα είναι πολλά επίσης δεν μπορούν να χρησιμοποιηθούν όλοι οι data movers εφόσον υπάρχει ένα όριο για τον κάθε ένα (για το μέγεθος δηλαδή που μπορεί να μεταφέρει). Για παράδειγμα για πολλά δείγματα (πχ 10.000 δηλαδή 32000 bytes μαζί με τα μη ταξινομημένα που είναι 50 δηλαδή 1600 bytes >16K όπου στη συγκεκριμένη πλακέτα μέχρι τόσα δεδομένα μπορούν να μεταφερθούν) . Οι data movers που μπορούν να χρησιμοποιηθούν είναι ο copy μαζί με fifo interface (να παίρνονται τα δεδομένα σειριακά) και ο zero copy(απαιτεί η μνήμη να είναι contiguous και αντιστοιχίζει το όρισμα σε κοινή μνήμη). Παρακάτω αναφέρονται τρόποι υλοποίησης της Hardware συνάρτησης ενός μεγάλου πίνακα (10.000 δειγμάτων ως σετ εκπαίδευσης και 50δειγμάτων ως δείγματα προς ταξινόμηση).

<u>1ος Τρόπος Υλοποίησης της Συνάρτησης (ZERO COPY και COPY &DMA 2D στην υποσυνάρτηση):</u>

Όπως προαναφέρθηκε παραπάνω με το ZERO COPY η καθυστέρηση έχει κάποια καθυστέρηση. Πρέπει να ληφθεί υπόψιν ότι με αυτόν το τρόπο το AXI protocol handshake διαρκεί τουλάχιστον 4 κύκλους ανά λειτουργία ανάγνωσης / εγγραφής εκτός από το χρόνο που χρειάζεται ο ελεγκτής μνήμης (memory controller) PS να ανταποκριθεί με τα δεδομένα προς ανάγνωση. Μπορεί να προστεθούν 20+ κύκλοι καθυστέρησης σε κάθε λειτουργία ανάγνωσης. Ωστόσο η απόδοση στις μεταφορές «burst» (μεγάλος όγκος δεδομένων) είναι καλή.

Το AXI Memmory Mapped interface δίνει τη δυνατότητα στον πυρήνα να εκτελεί ανάγνωση / εγγραφή απευθείας στη μνήμη PS (DDR ή cache) χρησιμοποιώντας την εντολή: #pragma δεδομένα SDS zero_copy (samples) (samples είναι το όνομα του ορίσματος) , τοποθετώντας την ακριβώς πριν τη δήλωση της συνάρτησης όπου μπαίνουν τα ορίσματα (οι πίνακες) σε κοινή μνήμη:

Το ZERO COPY λόγω του ότι τα ορίσματα πρέπει να είναι inout συγχέονται όλα τα ορίσματα σε ένα πίνακα διότι μόνο με αυτόν το τρόπο θα λειτουργήσει. Με τα παρακάτω pragmas τοποθετώντας τα ακριβώς πριν τη δήλωση της συνάρτησης επιτυγχάνεται η κοινή μνήμη:

ZERO COPY στην Hardware function:

```
1 #pragma SDS data zero_copy (samples)
2 #pragma SDS data mem_attribute (samples: CACHEABLE|PHYSICAL_CONTIGUOUS)
3 void knn_algorithm_hw (doublesamples [10050*4], int a, int b);
```

Τα δεδομένα πρέπει να αντιγράφονται στη τοπική μνήμη bram , λόγω του ότι δεν επαρκεί όμως ο χώρος της bram για τα 10.000 δειγματα, τα δειγματα διασπώνται και περνούν μέσα στην εσωτερική συνάρτηση (η οποία δημιουργήθηκε γι'αυτόν το σκοπό) ανά 1000άδες.Στη πρώτη κλήση της συνάρτησης αφου αντιγράφουν τοπικά τα δεδομένα θα εισαχθούν τα πρώτα 1000 δεδομένα για επεξεργασία και στη συνέχεια τα υπόλοιπα (ανα χιλιάδες). Για την υποσυνάρτηση θα πρέπει να οριστούν data movers όπως και για τη Hardwareσυνάρτηση. Οι data movers που χρησιμοποιήθηκαν είναι οι εξής:

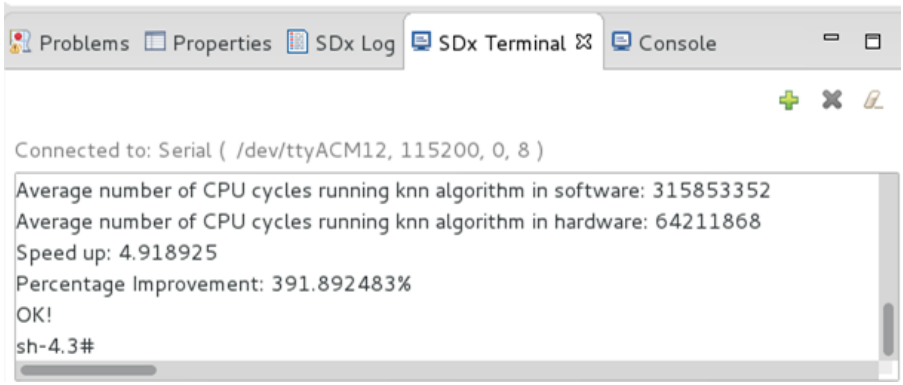
COPY και DMA 2D στην υποσυνάρτηση της Hardware function:

```
1 #pragma SDS data data_mover (
  deigma: AXIDMA_2D, deigma1:AXIDMA_2D, best_voters:AXIDMA_2D)
2 #pragma SDS data dim (deigma [MAT][W],deigma1[MAT][W],best_voters[K*MAT][W])
3 #pragma SDS data access_pattern (
  deigma: SEQUENTIAL, deigma1: SEQUENTIAL, best_voters: SEQUENTIAL)
4 #pragma SDS data copy (
  deigma[0:MAT][0:W],deigma1[0:MAT][0:W],best_voters[0:K*MAT [0:W])
5 void eukleideia_best_voters (
  doubledeigma [MAT][W], double deigma1[MAT][W],double best_voters[K*MAT][W]);
```

Ο DMA 2D data mover για να χρησιμοποιηθεί πρέπει να εισαγχθεί και ο COPY data mover για τον λόγο που σε προηγούμενο κεφάλαιο (**5.1.1.1. Data Movers**). Η εντολή η οποία καθορίζει τον τρόπο με τον οποίο διαβάζονται/γραφονται τα δεδομένα πρέπει να εισαχθεί και εκείνη και να οριστεί ως SEQUENTIALεπειδή τα δεδομένα είναι πολλά (1000*4*8=3200) (**5.1.1.1. Data Movers**) .

Στην παρακάτω εικόνα (**Εικόνα 30:** Απόδοση Συστήματος (ZERO COPY και COPY&DMA 2D)) αναγράφεται τη απόδοση του συστήματος με Data Mover ZERO COPY στην Hardware function και στην υποσυνάρτηση όπου παίρνονται λίγα-λίγα τα δεδομένα COPY και DMA 2D. Η βελτίωση δεν είναι τόσο υψηλή όσο ήταν προηγουμένως (391% βελτίωση σε σχέση με τη Software συνάρτηση) διοτι όσο αυξάνονται τα δείγματα (ακόμα και αν εκμεταλλευτούμε περισσότερο τους πόρους του fpga) επειδή δεν μπορούν να μεταφερθούν απευθείας στην μνήμη bram. Το ότι χρησιμοποιείται ο ZERO

COPY ο οποίος χρησιμοποιεί AXI Memmory Mapped interface φαίνεται και στον **Πίνακα 14:** Data Motion Report (ZERO COPY) στη στήλη «connection» όπου με αυτόν τον τρόπο το AXI protocol handshake θα διαρκέσει τουλάχιστον 4 κύκλους ανά λειτουργία ανάγνωσης / εγγραφής εκτός από το χρόνο που χρειάζεται ο ελεγκτής μνήμης (memory controller) PS να ανταποκριθεί με τα δεδομένα προς ανάγνωση. Αυτό μπορεί να προσκαλέσει μια μικρή καθυστέρηση. Στον επόμενο πίνακα (**Πίνακας 15:** Accelerator Callsites (ZERO COPY)) στη στήλη «transfersize(bytes)» αναγράφεται ο συνολικός αριθμός bytes που μεταφέρονται (10.000*4*8 τα ταξινομημένα δείγματα και 50*4*8 τα δείγματα προς ταξινόμηση), στήλη «Paged or Contoguous» φαίνεται ότι τα δεδομένα είναι συνεχόμενα στη μνήμη (PHYSICAL_CONTIGUOUS)αυτό διασφαλίζεται και από την εντολή mem_attribute καθώς και από την sds_alloc() όπως προαναφέρθηκε στο προηγούμενο κεφάλαιο. Στον επόμενο πίνακα (**Πίνακας 16:** Ποσοστό χρήσης πόρων του fpga (ZERO COPY)) είναι εμφανης η πτώση δεσμευσης πολλών πόρων που υπήρχε στο προηγούμενο παράδειγμα η οποία οφειλόταν στη διαμόρφωση που είχε ο κώδικας. Η διαφορά σε σχέση με πριν είναι ότι τώρα οικ γείτονες αναζητούνται στην υποσυνάρτηση και αφαιρέθηκαν κάποιες εντολές hlsοι οποίες είχαν βελτιστοποιήσει την απόδοση του αλγόριθμου όμως απαιτούσαν πολλά resources με αποτέλεσμα το design να είναι πολύ μεγάλο και να μην χωράει στο board.



Εικόνα 30:Απόδοση Συστήματος(ZERO COPY και COPY&DMA 2D)

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples	INOUT	40200*8	• mem_attribute:CACHEABLE.PHYSICAL_CONTIGUOUS • data_mover:zero_copy	ps7_S_AXI_ACP:AXIMM:0xC
	a	a	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	b	b	IN	4		ps7_M_AXI_GP0:AXILITE:0x14

Πίνακας 14: Data Motion Report (ZERO COPY)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	zerocopy.cpp:176:2	samples	321600	PHYSICAL_CONTIGUOUS	526	378735
		a	4	paged	0	13
		b	4	paged	0	13

Πίνακας 15: Accelerator Callsites (ZERO COPY)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	15
FIFO	0	-	15	132
Instance	32	31	11241	13375
Memory	0	-	256	8
Multiplexer	-	-	-	6
Register	-	-	6	-
Total	32	31	11518	13536
Available	280	220	106400	53200
Utilization (%)	11	14	10	25

Πίνακας 16: Ποσοστό χρήσης πόρων του fpga (ZERO COPY και COPY & DMA 2D)

2ος Τρόπος Υλοποίησης της Συνάρτησης (COPY&SEQUENTIAL και COPY&DMA 2Dστηνυποσυνάρτηση):

Σ' αυτήν την υλοποίηση θα χρησιμοποιηθεί ο COPYdatamover. Λόγω του ότι τα δεδομένα είναι πολλά απαιτείται να είναι και SEQUENTIAL (να μεταφέρονται δηλαδη σειριακά). Γενικά μπορεί να χρησιμοποιηθεί μόνο του το SEQUENTIAL χωρίς datamover (εφόσον το COPYείναι προκαθορισμένο από το πρόγραμμα) εκτός αν κάποιο όρισμαπίνακας περνάει ώς pointer (όπως συμβαίνει παρακάτω) όπου πρέπει να εισαχθεί υποχρεωτικά για να δηλώσει το μέγεθος του πίνακα. Αν τα δεδομένα διαβάζονται μόνο μία φορά και σε σειρά (διαδοχικά διαβάζοντας κάθε στοιχείο), τότε ορίζεται ένα FIFO interface χρησιμοποιώντας την εντολή pragma: #pragma SDS data access_pattern (samples: SEQUENTIAL) όπου το «samples» είναι το όνομα του ορίσματος. Το pragma τοποθετείται ακριβώς πάνω από τη συνάρτηση .Τα δεδομένα στη μνήμη είναι αποθηκευμένα με συνέχεια δηλαδή το ένα μετά το άλλο αλλά επειδή όπως προαναφέρθηκε επειδή είναι πολλά τα δεδομένα καλό θα ήταν να εισαχθεί και η αντίστοιχη εντολή για να εξασφαλιστεί (mem_attribute).

Με αυτήτην υλοποίηση θα χρησιμοποιηθεί ο AXI DMA Simple ως data mover εφόσον τα δεδομένα είναι αποθηκευμένα με sds_alloc() (**5.1.1.1. Data Movers**). Η υλοποίηση του αλγορίθμου πρέπει να αλλάξει και πάλι εφόσον αυτή τη φορά τα ορίσματα θα είναι είτε IN είτε OUT.Η σύνταξη του κώδικα είναι η εξής:

COPYκαιSEQUENTIAL:

```
1 #pragma SDS data copy (samples [0: h*w])
2 #pragma SDS data mem_attribute (samples: PHYSICAL_CONTIGUOUS|CACHEABLE)
3 #pragma SDS data access_pattern (
  samples: SEQUENTIAL, samplesu: SEQUENTIAL, group: SEQUENTIAL)
4 void knn_algorithm_hw (
  double *samples, doublesamplesu [U*4],double group [U], int h,int w);
```

Αν κάποιος τύπος δεν καθοριστεί για ένα όρισμα πίνακα ,είναι προκαθορισμένος ο COPY Data Mover σε αυτήν την περίπτωση το πρόγραμμα πρώτα ελέγχει τον τύπο του ορίσματος. Αν ο τύπος του ορίσματος έχει compile-time μέγεθος πίνακα, τότε ο compiler χρησιμοποιεί αυτό σαν μέγεθος μεταφοράς δεδομένων. Ειδάλλως, το πρόγραμμα αναλύει τον κώδικα για να καθορίσει το μέγεθος μεταφοράς βάσει τη δέσμευση χώρου μνήμης του πίνακα (για παράδειγμα, malloc ή sds_alloc) και συμπεριφέρεται αναλόγως. Για παράδειγμα αν υλοποιηθεί με δείκτες τότε χρησιμοποιεί τον DMA SCATTER GATHER (**Πίνακας 17:** Data Motion Network Report (COPY&SEQUENTIAL) στη στήλη «Connection») ακόμα και αν τα δεδομένα είναι αποθηκευμένα με sds_alloc() ενώ αν περαστεί ο πίνακας κατευθείαν μαζί και το μέγεθος του τότε χρησιμοποιεί τον DMA SIMPLE εφόσον βέβαια τα δεδομένα είναι αποθηκευμένα με sds_alloc().

Παρακάτω η υλοποίηση της υποσυνάρτησης δεν έχει αλλάξει είναι ακριβώς η ίδια όπως και στο προγούμενο παράδειγμα και η υλοποιηση της είναι η ακόλουθη:

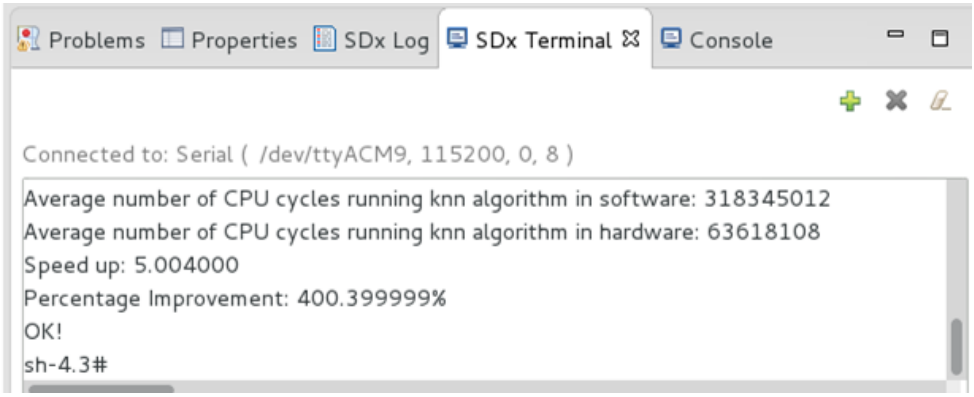
COPY και DMA 2D στην υποσυνάρτηση της Hardware function:

```
1 #pragma SDS data data_mover (
  deigma: AXIDMA_2D, deigma1:AXIDMA_2D, best_voters:AXIDMA_2D)
2 #pragma SDS data dim (deigma [MAT][W], deigma1 [MAT][W],best_voters[K*MAT][W])
3 #pragma SDS data access_pattern (
  deigma: SEQUENTIAL, deigma1: SEQUENTIAL, best_voters: SEQUENTIAL)
4 #pragma SDS data copy (
  deigma[0:MAT][0:W],deigma1[0:MAT][0:W],best_voters[0:K*MAT [0:W])
5 void eukleideia_best_voters (
  doubledeigma [MAT][W],double deigma1[MAT][W],double best_voters[K*MAT][W]);
```

Σε περίπτωση που δεν οριστεί κάποια εντολή πάνω από τη Hardware συνάρτηση τότε χρησιμοποιείται το COPY (αντιγραφή δεδομένων). Κατά συνέπεια, τα ορίσματα της Hardware συνάρτησης χρησιμοποιούνται είτε ως είσοδος είτε ως έξοδος, αλλά όχι και τα δύο.

Στους πίνακες (**Πίνακας 17:** Data Motion Network Report (COPY & SEQUENTIAL) και **Πίνακας 18:** Accelerator Callsites (COPY & SEQUENTIAL)) παρακάτω φαίνεται διαφορετικά το μέγεθος των δεδομένων (0*8 στηστήλη «Declared size(bytes)» (h*w)*8 στη στήλη «transfer size(bytes)» αντίστοιχα) αυτό οφείλεται στο γεγονός ότι γι'αυτήν την υλοποίηση χρειάστηκε οπίνακας samples να περαστεί ως pointer και το μέγεθος του δεν αναγράφεται κατά την δήλωση της συνάρτησης όπως γινόταν στα προηγούμενα παραδείγματα καθώς επίσης γι’ άυτό το λόγο επιλέγει να χρησιμοποιήσει τον DMA SCATTER GATHER παρόλο που τα δεδομένα είναι αποθηκευμένα με sds_alloc(). Η απόδοση του συστήματος (**Εικόνα 31:** Απόδοση Συστήματος (COPY & SEQUENTIAL και COPY & DMA 2D) δεν έχει και πολύ μεγάλη διαφορά σε σχέση με προηγούμενο παραδειγμα ωστόσο είναι καλή (391% στο πρώτο παραάδειγμα 400% σε αυτό το παράδειγμα).

Πρέπει να επισημανθεί ότι με αυτή την υλοποίηση έχουν μειωθεί τα resources και η μικρή διαφορά με τα resources του ZERO COPY προηγουμένως ευθύνεται στο ότι σ’ άυτό το τρόπο υλοποίησης χρησιμοποιείται ο DMA SCATTER GATHER (**Πίνακας 19:** Ποσοστό χρήσης πόρων του fpga (COPY&SEQUENTIAL και COPY & DMA 2D)).



Εικόνα 31:Απόδοση Συστήματος (COPY & SEQUENTIAL και COPY & DMA 2D)

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS,CACHEABLE	ps7_S_AXI_ACP:AXIDMA_SG
	samplesu	samplesu	IN	200*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	group	group_r	OUT	50*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14

Πίνακας 17: Data Motion Network Report (COPY & SEQUENTIAL)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	mpliax.cpp:196:2	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	9188	9859282
		samplesu	1600	contiguous	1118	2711
		group_r	400	contiguous	1118	1433
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13

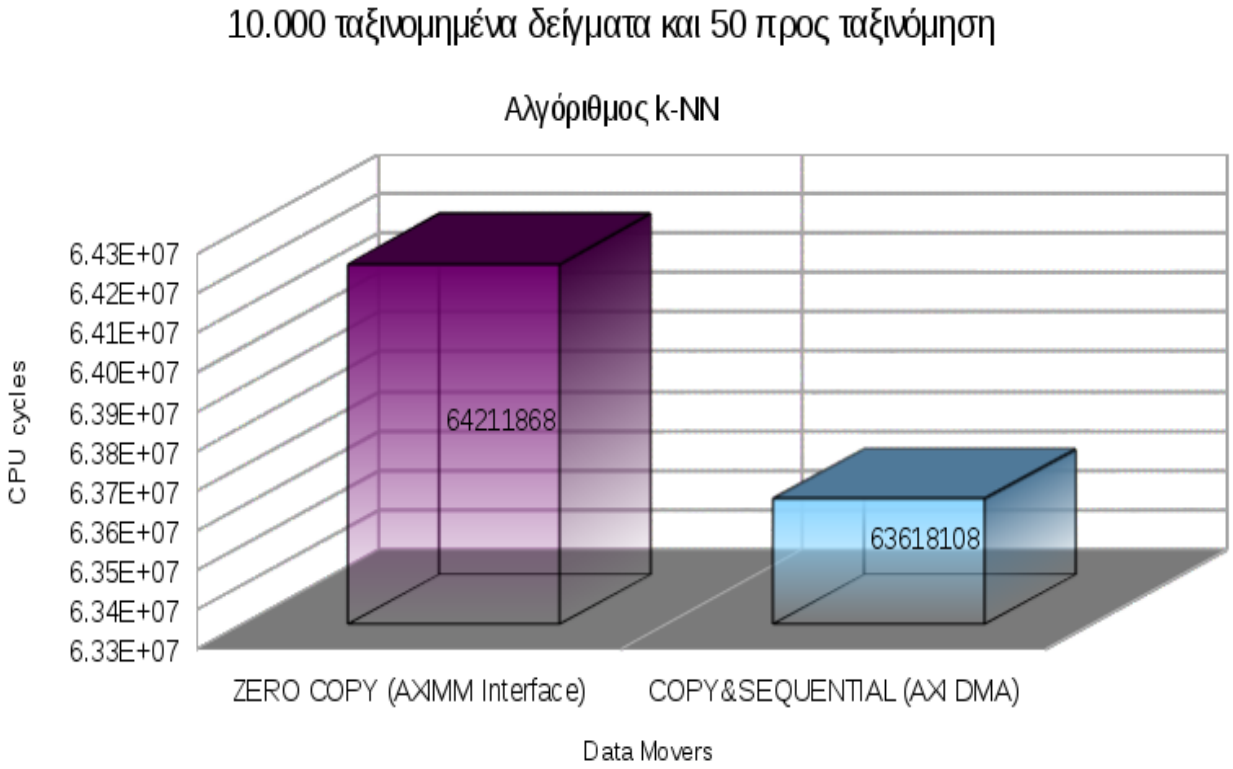
Πίνακας 18: Accelerator Callsites (COPY & SEQUENTIAL)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	27
FIFO	0	-	10	88
Instance	20	34	11305	13632
Memory	16	-	256	8
Multiplexer	-	-	-	11
Register	-	-	11	-
Total	36	34	11582	13766
Available	280	220	106400	53200
Utilization (%)	12	15	10	25

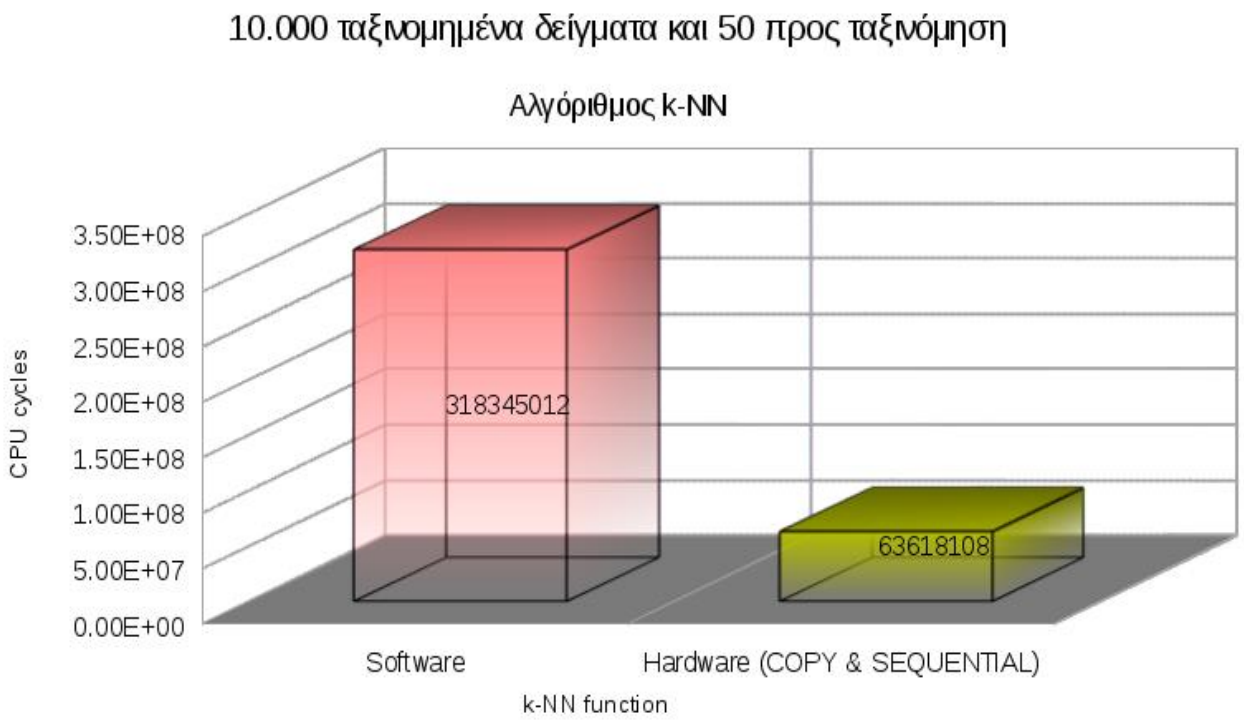
Πίνακας 19: Ποσοστό χρήσης πόρων του fpga (COPY & SEQUENTIAL και COPY & DMA 2D)

Στην εικόνα παρακάτω υπάρχει μικρή διαφορά στην απόδοση ανάμεσα στην υλοποίηση της συνάρτηση με τον data mover ZERO COPY (391%) και τον COPY(400%)με FIFO Interface (SEQUENTIAL). Στην δεύτερη περίπτωση χρησιμοποιούνται οι AXI DMA SIMPLE και AXI DMA SCATTER GATHER. Το AXI Memmory Mapped interface όπου το AXI protocol handshake θα διαρκέσει τουλάχιστον 4 κύκλους ανά λειτουργία ανάγνωσης / εγγραφής εκτός από το χρόνο που χρειάζεται ο ελεγκτής μνήμης (memory controller) PS να ανταποκριθεί με τα δεδομένα προς ανάγνωση. Αυτό μπορεί να προσκαλεί μια μικρή καθυστέρηση[13]. Πρέπει να σημειωθεί πάντως ότι για burst μεταφορές έχει καλή απόδοση και δεν έχει ουσιαστική διαφορά σε σχέση με το DMA , συνεπώς σε αυτή τη περίπτωση και αν τα δεδομένα χρειαστεί να είναι inout θα ήταν μια καλή επιλογή.

Παρακάτω στο γράφημα παρατειρείται αυτή η διαφορά ανάμεσα στους data movers που χρησιμοποιήθηκαν (**Εικόνα 32:** Σύγκριση μετρήσεων του k-NN αλγορίθμου με διαφορετικούς data movers (ZERO COPY, COPY & SEQUENTIAL)). Τέλος στο γράφημα παρακάτω **Εικόνα 33:** Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)) απεικονίζεται η βελτιωμένη απόδοση του αλγορίθμου k-NN στο Hardware σε σχέση με το Software χρησιμοποιώντας τη βέλτιστη λύση με τον COPY & SEQUENTIAL data mover (400% βελτίωση).



Εικόνα 32: Σύγκριση μετρήσεων του k-NN αλγορίθμου με διαφορετικούς data movers (ZERO COPY, COPY&SEQUENTIAL)



Εικόνα 33: Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)

6.2 Τεχνικές για Παραλληλισμό του Συστήματος

Όταν δεν υπάρχει εξάρτηση των δεδομένων (δηλαδή δεν χρειάζεται όλα τα δεδομένα να περάσουν από τη Hardware συνάρτηση) τότε υπάρχει η δυνατότητα του παραλληλισμού είτε αυτό έχει να κάνει με παραλληλισμό των κλήσεων μιας συνάρτησης είτε με παραλληλισμό των κλήσεων δύο διαφορετικών συναρτήσεων. Υπάρχουν διάφοροι τρόποι υλοποίησης για να πετύχει αυτό (ο παραλληλισμός των κλήσεων και accelerators) όμως και πάλι εξαρτάται από τον όγκο των δεδομένων. Παρακάτω ο αλγόριθμος υλοποιείται και με λίγα και με πολλά δεδομένα.

6.2.1 Παραλληλισμός Δύο Επιταχυντών (για Λίγα Δείγματα)

Με βάση τον αποδοτικότερο τρόπο υλοποίησης του προηγούμενου κεφαλαίου θα δοκιμαστεί η εφαρμογή δύο επιταχυντών αντί για έναν, επιδιώκοντας καλύτερη απόδοση χρόνου και καθώς και παραλληλισμό του συστήματος.

Παρακάτω εφαρμόζονται δύο τρόποι αύξησης του παραλληλισμού του συστήματος:

1ος Τρόπος Υλοποίησης της Συνάρτησης (παραλληλισμός δύο επιταχυντών με τη χρήση Task-Level Pipeline):

Το**async** pragma καθορίζεται αμέσως πριν από μια κλήση μιαςHardwareσυνάρτηση, καθοδηγώντας τον μεταγλωττιστή να μην παράγει αυτόματα την αναμονή βάσει της ανάλυσης ροής δεδομένων.

Το πρόγραμμα πρέπει να συγχρονιστεί σωστά με την κλήση της συνάρτησης εισάγοντας ένα αντίστοιχο pragma **wait** με το ίδιο id σε ένα κατάλληλο σημείο του προγράμματος.

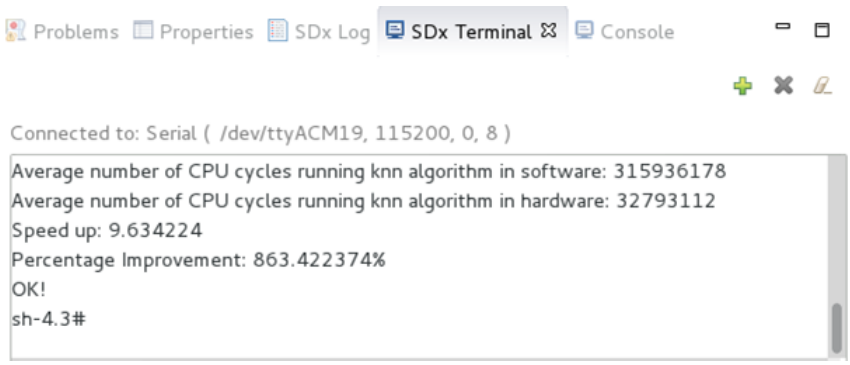
Χρησιμοποιώντας τα παρακάτω pragmas, τοποθετώντας τα ακριβώς πριν τη κλήση της συνάρτησης μαζί με το id των ορισμάτων του accelerator δημιουργείται Task-Level Pipelining. Για το συγκεκριμένο τρόπο υλοποίησης Task-Level Pipeline και για βελτίωση της απόδοσης του αλγορίθμου θα δημιουργηθεί ένας ακόμα επιταχυντής (κάθε επιταχυντής έχει διαφορετικό id) . Η σύνταξη του κώδικα γράφεται ως εξής:

TASK-LEVELPIPELINE:

```
1  #pragmaSDSasync (1)
2      knn_algorithm_hw (samples, samplesu, group1, H, W, U, u1);
3  #pragma SDS async (2)
4      knn_algorithm_hw1 (samples, samplesu1, group, H, W, U, u1);
5  #pragmaSDSwait (1)
6  #pragmaSDSwait (2)
```

Στην παρακάτω εικόνα (**Εικόνα 34:** Απόδοση Συστήματος(Task-Level Pipeline)) παρατηρείται ότι ότι οι κύκλοι στη CPU έχουν πέσει γύρω στο 50% (απόδοση χωρίς παραλληλισμό του συστήματος : 400% απόδοση με παραλληλισμό 863%). Αυτό συμβαίνει διότι παράγεται ένας δεύτερος επιταχυντής (τοποθετόντας δύο id ,ένα για την κάθε κλήση). Με αυτόν τον τρόπο μοιράζονται τα μη ταξινομημένα (τα μισά σε κάθε accelerator) και επιτυγχάνεται καλύτερη απόδοση.

Ωστόσο αυτή η εντολή ίσως να ήταν πιο χρήσιμη αν υπήρχαν δύο διαφορετικές συναρτήσεις και όχι τόσο σε αυτή τη περίπτωση όπου οι δύο accelerators κάνουν το ίδιο πράγμα.



Εικόνα 34:Απόδοση Συστήματος (Task-LevelPipeline)

Στον πίνακα παρακάτω (**Πίνακας 20:** Data Motion Report (Task-Level Pipeline)) στη στήλη pragmas αντί για DMA SIMPLE χρησιμοποιήθηκε το DMA SCATTER-GATHER ως datamoverγια τη μεταφορά όλων των ορισμάτων (εκτός των κατηγοριοποιημένων που χρησιμοποιούν το COPY). Αυτό συνέβη διότι η εντολή async-wait υλοποιείται μόνο με DMA SCATTER-GATHER. Για βελτίωση χρόνου εκτέλεσης η δέσμευση μνήμης των πινάκων εξακολουθεί να γίνεται με sds_alloc() (δηλαδή να είναι συνεχόμενη) οπότε, η απόδοση με DMA SCATTER-GATHER (σε σχέση με τον DMA SIMPLE) δεν θα έχει σημαντική διαφορά. Επίσης τα δεδομένα προς ταξινόμηση θα περάσουν από την HP port αυτό θα γίνει επειδή τα όλα τα δεδομένα της συνάρτησης δεν χωρούν να περάσουν από την ACP port (εξαιτίας της δημιουγίας και του δεύτερου επιταχυντή) συνεπώς κάποιο από τα ορίσματα θα έπρεπε να περάσει από αυτήν. Επιλέχθηκε ο πίνακας samples επειδή διαβάζεται μια μόνο φορά. Στον παρακάτω πίνακα (**Πίνακας 21:** Accelerator Callsites (Task-Level Pipeline)) σημαντική παρατήρηση αποτελεί ο δεύτερος επιταχυντής που δημιουργείται στη στήλη accelearator.

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.NON_CACHEABLE	ps7_S_AXI_HP0:AXIDMA_SG
	samplesu	samplesu	IN	200*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	group	group_r	OUT	50*8		ps7_S_AXI_ACP:AXIDMA_SG
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14
	u1	u1	IN	4		ps7_M_AXI_GP0:AXILITE:0x18
knn_algorithm_hw1_1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.NON_CACHEABLE	ps7_S_AXI_HP1:AXIDMA_SG
	samplesu	samplesu	IN	200*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	group	group_r	OUT	50*8		ps7_S_AXI_ACP:AXIDMA_SG
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14
	u1	u1	IN	4		ps7_M_AXI_GP0:AXILITE:0x18

Πίνακας 20: Data Motion Report (Task-Level Pipeline)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	async_wait.cpp:190:2	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	5672	9851923
		samplesu	1600	contiguous	3597	4798
		group_r	400	contiguous	3299	4734
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
		u1	4	paged	0	13
knn_algorithm_hw1_1	async_wait.cpp:192:2	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	5672	9851923
		samplesu	1600	contiguous	3597	4798
		group_r	400	contiguous	3299	4734
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
		u1	4	paged	0	13

Πίνακας 21: Accelerator Callsites (Task-Level Pipeline)

Παρακάτω στον πρώτο πίνακα (**Πίνακας 22:** Ποσοστό χρήσης πόρων του fpga (Task-Level Pipeline) αναγράφονται τα «resources» που δεσμεύτηκαν για τον πρώτό accelerator (δηλαδή για την πρώτη Hardware συνάρτηση) και έπειτα για τον δεύτερο. Με αυτή την εντολή και με τη δημιουγία και του δεύτερου επιταχυντή ,τα resources στην ουσία διπλασιάζονται οπότε και το block designμεγαλώνει. Αυτό δεν είναι και τόσο θετικό διότι μπορεί σε κάποιο άλλο παράδειγμα το block design να είναι τόσο μεγάλο ώστε να μην χωράει στη πλακέτα.

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	27
FIFO	0	-	20	176
Instance	20	34	11472	13752
Memory	16	-	256	8
Multiplexer	-	-	-	11
Register	-	-	11	-
Total	36	34	11759	13974
Available	280	220	106400	53200
Utilization (%)	12	15	11	26

Πίνακας 22: Ποσοστό χρήσης πόρων του fpga για hw1 (Task-Level Pipeline)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	27
FIFO	0	-	20	176
Instance	20	34	11472	13752
Memory	16	-	256	8
Multiplexer	-	-	-	11
Register	-	-	11	-
Total	36	34	11759	13974
Available	280	220	106400	53200
Utilization (%)	12	15	11	26

Πίνακας 23: Ποσοστό χρήσης πόρων του fpga για hw2 (Task-Level Pipeline)

2ος Τρόπος Υλοποίησης της Συνάρτησης (παραλληλισμός δύοεπιταχυντώνμε τη χρήσητου Resource):

Υπάρχει και ένας άλλος τρόπος με τον οποίος αυξάνεται ο παραλληλισμός και ο συγχρονισμός του συστήματος, χρησιμοποιώντας τον sdscc compiler. Κατευθύνοντας τον compiler δημιουργεί πολλαπλά «instances» της Hardware συνάρτησης εισάγοντας την ακόλουθη εντολή ακριβώς πριν τη κλήση της συνάρτησης (το ακόλουθο pragma δημιουργεί instances από το id):

RESOURCE:

```
1  #pragma SDS resource (1)
2     knn_algorithm_hw (samples, samplesu, group, H, W, U, u1);
3  #pragmaSDS resource (2)
4     knn_algorithm_hw (samples, samplesu1, group, H, W, U, u1);
```

Με αυτόν τον τρόπο επιδιώκεται παραλληλισμός και συγχρονισμός του συστήματος. Δημιουργείται ένας ακόμα accelerator συνεπώς τα δεδομένα μοιράζονται άρα η απόδοση του συστήματος αυξάνεται πάνω από το διπλάσιο (χωρίς παραλληλισμό του συστήματος η απόδοση είναι 400% ενώ με παραλληλισμο (Resource) 887%), όπως φαίνεται και στην εικόνα παρακάτω(**Εικόνα 35:** Απόδοση Συστήματος (Resource)). Η διαφορά σε σχέση με το προηγούμενο παράδειγμα είναι ότι σε αυτό το παράδειγμα δεν χρειάζεται να υπάρχει ένας δεύτερος accelerator ,το πρόγραμμα τον παράγει από μόνο του μέσω αυτής της εντολής.

Εικόνα 35:Απόδοση Συστήματος (Resource)

Στους πίνακες παρακάτω (**Πίνακας 24:** Data Motion Report (Resource) και **Πίνακας 25:** Accelerator Callsites (Resource)) φαίνεται σαν να υπάρχει και δεύτερος accelerator , είναι το πρώτο και το δεύτερο instance που δημιουργήθηκε μέσα από την εντολή resource. Αυτή τη φορά περνούν όλα τα δείγματα από την ACP port εφόσον με τη resource εντολή δεν χρειάστηκε ο dmascatter-gather. Οι πόροιτου fpga φαίνεται να παραμένουν στα ίδια επίπεδα (**Πίνακας 26:** Ποσοστό χρήσης πόρων του fpga (Resource)) και μοιάζει σαν να δημιουργεί ένα μόνο accelerator στην πραγματηκότητα όμως όπως φαίνεται και στον παρακάτω πίνακα (**Πίνακας 24:** Data Motion Report (Resource)) στη στήλη «Accelerator» υπάρχουν δύο επιταχυντές άρα και τα resourcestου fpgaδιπλασιάζονται. Όπότε στον πίνακαπαρακάτω **(Πίνακας 26:** Ποσοστό χρήσης πόρων του fpga (Resource)) είναι οι ποροι που δεσμεύονται είναι επί δύο.

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw2	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE	ps7_S_AXI_ACP:AXIDMA_SG
	samplesu	samplesu	IN	0*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	group	group_r	OUT	0*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14
	u1	u1	IN	4		ps7_M_AXI_GP0:AXILITE:0x18
knn_algorithm_hw1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE	ps7_S_AXI_ACP:AXIDMA_SG
	samplesu	samplesu	IN	0*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	group	group_r	OUT	0*8		ps7_S_AXI_ACP:AXIDMA_SIMPLE
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14
	u1	u1	IN	4		ps7_M_AXI_GP0:AXILITE:0x18

Πίνακας 24: Data Motion Report (Resource)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw2	recource.cpp:187:2	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	9188	9659282
		samplesu	1600	contiguous	1118	2711
		group_r	400	contiguous	1118	1433
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
		u1	4	paged	0	13
knn_algorithm_hw1	recource.cpp:185:2	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	9188	9659282
		samplesu	1600	contiguous	1118	2711
		group_r	400	contiguous	1118	1433
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
		u1	4	paged	0	13

Πίνακας 25: Accelerator Callsites (Resource)

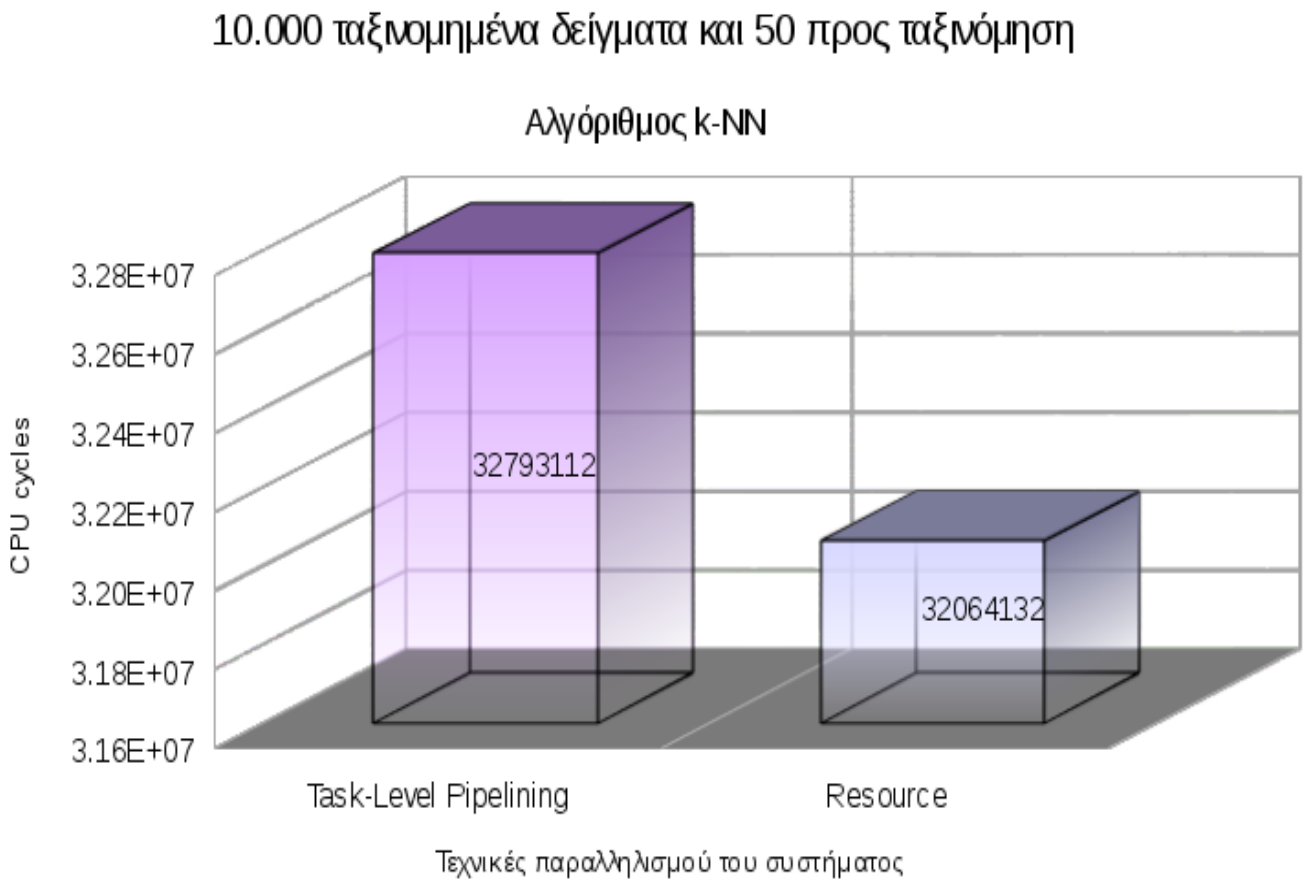
Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	27
FIFO	0	-	20	176
Instance	20	34	11472	13752
Memory	16	-	256	8
Multiplexer	-	-	-	11
Register	-	-	11	-
Total	36	34	11759	13974
Available	280	220	106400	53200
Utilization (%)	12	15	11	26

Πίνακας 26: Ποσοστό χρήσης πόρων του fpga (Resource)

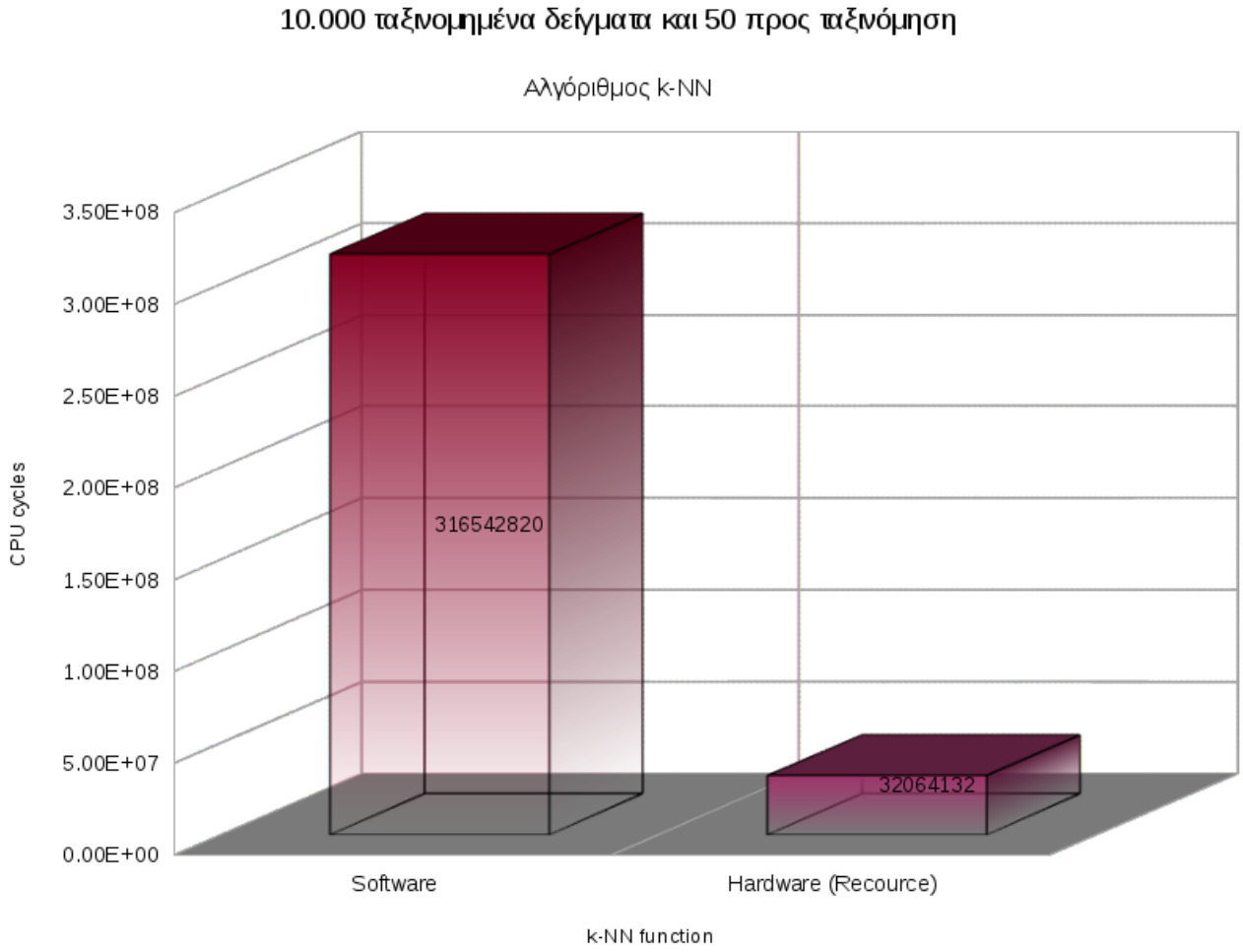
Η διαφορά ανάμεσα στην resource και την και async-wait εντολή είναι ότι το resource παραλληλίζει τους δύο accelerators ενώ το με την async-wait γίνεται pipeline στους accelerators.

Επίσης με τον τρόπο που χρησιμοποιείται σε αυτή περίπτωση η Task-Level Pipeline εντολή καθιστά αναγκαία τη δημιουργία μιας ακόμα Hardware συνάρτησης (δηλαδή accelerator) κάτι το οποίο μπορεί να αποφευχθεί με το resource που στην ουσία δημιουργεί acceleratorαπό μονο του.Στην **Εικόνα 36:** Σύγκριση απόδοσης του k-NNαλγορίθμου με διαφορετικές τεχνικές παραλληλοποίησης (Task-Level Pipeline , Resource) δεν έχει μεγάλη διαφορά στην απόδοση (με Task-Level Pipeline : 863%βελτίωση ενώ με Resource:887% βελτίωση)αλλά επειδή στο συγκκριμένο τρόπο υλοποίησης η Resource εντολή θεωρείται ειδανικότερη λόγω του ότι οι επιταχυντές στην ουσία αντιπροσωπεύουν την ίδια ακριβώς συνάρηση στη συγκεκριμένη περίπτωση η εντολή resource είναι η βέλτιστη λύση.

Στο γράφημα παρακάτω (**Εικόνα 37:** Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)) απεικονίζεται η βελτιωμένη απόδοση του αλγορίθμου k-NN στο Hardware σε σχέση με το Software η οποία είναι αρκετά καλή. Στην περίπτωση όμως που τα δείγματα αυξηθούν (π.χ. στα 20.000) τότε το blockdesignθα γίνει τόσο μεγάλο που δεν θα χωράει στη πλακέτα άρα αυτή η μέθοδος (resource) είναι καλή αλλά για λίγα δείγματα.



Εικόνα 36: Σύγκριση απόδοσης του k-NN αλγορίθμου με διαφορετικές τεχνικές παραλληλοποίησης (Task-Level Pipeline, Resource)



Εικόνα 37: Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)

6.2.2 Παραλληλισμός των Κλήσεων ενός Επιταχυντή (για Μεγάλο Όγκο Δεδομένων Χωρίς Εξάρτηση)

Παραπάνω αυξήθηκαν τα δεδομένα εκπαίδευσης από 100 σε 10.000 το ίδιο θα συμβεί και στα δείγματα προς ταξινόμηση δηλαδή από 50 θα γίνουν 10.000,οπότε τα δείγματα στο σύνολο θα αυξηθούν στα 20.000 (20.000*4*8=640.000 bytes). Θα εφαρμοστεί παραλληλισμός των κλήσεων ενός επιταχυντή. Επιδιώκεται καλή απόδοση χρόνου , περισσότερο παραλληλισμό του συστήματος και όσο το δυνατόν καλύτερη εκμετάλευση των πόρων. Παρακάτω εφαρμόζονται δύο τρόποι αύξησης του παραλληλισμού του συστήματος:

1ος ΤρόποςTask-LevelPipeline(2 κλήσειςστονaccelerator)

Αρχικά η διαμόρφωση του κώδικα θα αλλάξει και τα δεδομένα θα περνούν ανά 1000 αλλά αυτό θα γίνει εξωτερικά της συνάρτησης (δηλαδή τα δεδομένα προς ταξινόμηση θα εισέρχονται στη Hardwareσυνάρτηση ανά χιλιάδα). Ο λόγος είναι ότι αν όπως και προηγουμένως περαστούν όλα τα δεδομένα η τοπική μνήμη του accelerator δεν θα χωρέσει όλα αυτά τα δεδομένα. Δημιουργείται λοιπόν ένας πίνακας δεικτών όπου ανά κλήση της συνάρτησης θα περνάει από όλους τους δείκτες όπου ο κάθε δείκτης δείχνει σε ένα σετ χιλίων δειγμάτων προς ταξινόμηση.

Το **async** pragma μεταφέρει τοπικά τον πίνακα samplesu και έπειτα τον samplesu+1. Μόλις ολοκληρωθεί η μεταφορά των δεδομένων αρχίζει το computation των hardware κλήσεων πρώτα της πρώτης κλήσης και έπειτα της δεύτερης (**wait**) και όλα αυτά με pipeline. Το πρόγραμμα πρέπει να συγχρονιστεί σωστά με την κλήση της συνάρτησης εισάγοντας ένα αντίστοιχο pragma **wait** με το ίδιο id σε ένα κατάλληλο σημείο του προγράμματος.

Στο παράδειγμα παρακάτω δημιουργούνται 2 κλήσειςπρος τον ίδιο επιταχυντή. Με αυτό τον τρόπο επιδιώκεται η παραλληλοποίηση των κλήσεων (pipeline). Η σύνταξη του κώδικα θα γραφει ως εξής:

TASK-LEVELPIPELINE:

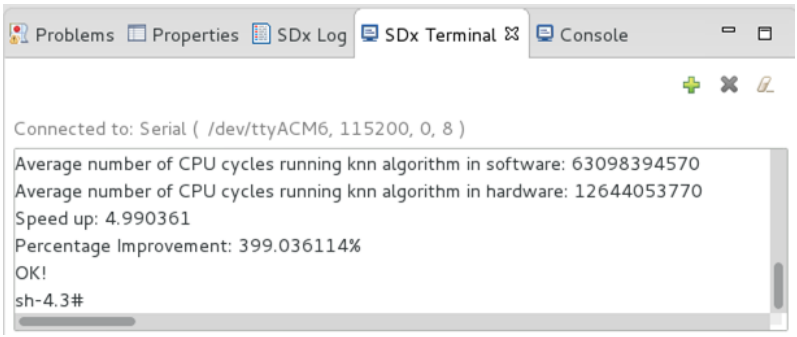
```
1     for (i=0; i<10; i+=2)
2     {
3  #pragma SDS async (1)
4             knn_algorithm_hw (samples, samplesu[i], group1 [i], H, W, U);
5  #pragma SDS async (2)
6             knn_algorithm_hw (samples, samplesu [i+1], group1 [i+1], H, W, U);
7   #pragma SDS wait (1)
8   #pragma SDS wait (2)
9     }
```

Στηνπαρακάτω δήλωση συνάρτησης πρέπει να εισαχθεί *2 φορές η εντολή #pragma SDS data data_mover η οποία έχει διαφορετικό id για κάθε κλήση. Η δήλωση της συνάρτησης πρέπει να γραφεί ως εξής:

DATA MOVERS:

```
1 #pragma SDS data data_mover (samplesu: AXIDMA_SG:1, group: AXIDMA_SG:1)
2 #pragma SDS data data_mover (samplesu: AXIDMA_SG:2, group: AXIDMA_SG:2)
3 #pragma SDS data mem_attribute (
            samples: PHYSICAL_CONTIGUOUS|CACHEABLE,
            samplesu: PHYSICAL_CONTIGUOUS|CACHEABLE,
            group: PHYSICAL_CONTIGUOUS|CACHEABLE)
4 #pragma SDS data copy (samples [0: h*w])
5 #pragma SDS data access_pattern (samples: SEQUENTIAL)
6 void knn_algorithm_hw (
  double *samples, doublesamplesu [MAT*4], doublegroup [MAT], int h, int w, int u);
```

Στην παρακάτω εικόνα (**Εικόνα 38:** Απόδοση Συστήματος (Task-Level Pipeline)) η pipeline απόδοση των 2 κλήσεων σε έναν επιταχυντή(accelerator) είναι περίπου 400% βελτιωμένη σε σχέση με το Software. Η απόδοση θεωρείται σχετικά καλή εφόσον τώρα τα δείγματα είναι 20.000 και κάθε μη ταξινομημένο δείγμα (10.000) πρέπει να συγκρινει την απόσταση του με όλα τα ταξινομημένα δείγματα (10.000*10.000=100.000 μετρήσεις!).



Εικόνα 38: Απόδοση Συστήματος (Task-Level Pipeline)

Στονεπόμενο πίνακα (**Πίνακας 27:** Data Motion Report (Task-Level Pipelining) στη στήλη «pragma» εμφανίζονται οι εντολές (pragma) που χρησιμοποιήθηκαν ενώ στη στήλη «declared size» φαίνεται η αυξηση των δειγμάτων από 50 σε 1000 (1000*4*8) εφόσον ο αρχικός πίνακας είναι 10000 και τα δεδομένα μεταφέρονται ανά χιλιάδες . Στον επόμενο πίνακα (**Πίνακας 28:** Accelerator Callsites (Task-Level Pipeline)) φαίνονται οι δύο κλήσεις προς τον acceleratorστη στήλη «Callsite» που ήταν και το ζητούμενο. Στον **Πίνακας 32:** Ποσοστό χρήσης πόρων του fpga (Task-Level Pipeline) δεν υπήρξε καμία αύξηση στη δέσμευση των πόρων του fpga εφόσον εξακολουθεί να υφίσταται μόνο ένας επιταχυντής.

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE	ps7_S_AXI_ACP:AXIDMA_SG
	samplesu	samplesu_PORTA	IN	4000*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	group	group_r_PORTA	OUT	1000*8	• data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14

Πίνακας 27: Data Motion Report (Task-Level Pipeline)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	async_buff_2calls.cpp:211:2	samples	(h*w)*8	PHYSICAL_CONTIGUOUS	9188	9859282
		samplesu_PORTA	32000	paged	55596	46059
		group_r_PORTA	8000	paged	28267	17379
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
	async_buff_2calls.cpp:209:2	samples	(h*w)*8	PHYSICAL_CONTIGUOUS	9188	9859282
		samplesu_PORTA	32000	paged	55596	46059
		group_r_PORTA	8000	paged	28267	17379
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13

Πίνακας 28: Accelerator Callsites (Task-Level Pipeline)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	21
FIFO	-	-	-	-
Instance	20	34	11337	13725
Memory	144	-	256	8
Multiplexer	-	-	-	8
Register	-	-	8	-
Total	164	34	11601	13762
Available	280	220	106400	53200
Utilization (%)	58	15	10	25

Πίνακας 29: Ποσοστό χρήσης πόρων του fpga (Task-Level Pipeline)

2ος Τρόπος Task-Level Pipeline (με BUFFER)

Εφαρμογή πολλαπλών κλήσεων ενός επιταχυντή για μεγαλύτερη απόδοση του συστήματος και καλύτερη διαχείριση του μεγάλου όγκου δεδομένων των μη ταξινομημένων δειγμάτων.

Το **async (asynchronous)** pragma υλοποιεί-εκτελεί το Task-Level Pipeliningόπως και προηγουμένως. Για την συγκεκριμένη υλοποίησης της εντολή Task-Level Pipeline χρειάζονται πολλαπλές κλήσεις προς τον επιταχυντή για να δοκιμαστεί απόδοση του συστήματος και χρόνου εκτέλεσης του αλγορίθμου με στόχο τη βελτίωση του Hardware συστήματος(στη συγκεκριμένη περίπτωση υλοποιούνται δέκα κλήσεις).

1. Ο πίνακας samplesu μεταφέρεται από τη κύρια μνήμη στις τοπικές μνήμες του accelerator (ο πίνακας samples πηγαίνει και αυτός στις τοπικές μνήμες αλλά εσωτερικά της συνάρτησης).
2. Οaccelerator εκτελεί (κάνει τους υπολογισμούς).
3. Το αποτέλεσμα το οποίο αποθηκεύεται στο πίνακα group μεταφέρεται από τον accelerator πίσω στη κύρια μνήμη.

TASK-LEVEL PIPELINE με BUFFER:

```
1    for (i=0; i< pipeline_depth; i++)
2      {
3  #pragma SDS async (1)
4            knn_algorithm_hw (samples, samplesu[i], group1 [i], H, W, U);
5      }
6    for (i=pipeline_depth; i<10; i++)
7      {
8  #pragma SDS wait (1)
9  #pragma SDS async (1)
10           knn_algorithm_hw (samples, samplesu[i], group1 [i], H, W, U);
11     }
12   for (i=0; i< pipeline_depth; i++)
13   {
14 #pragmaSDSwait(1)
15     }
```

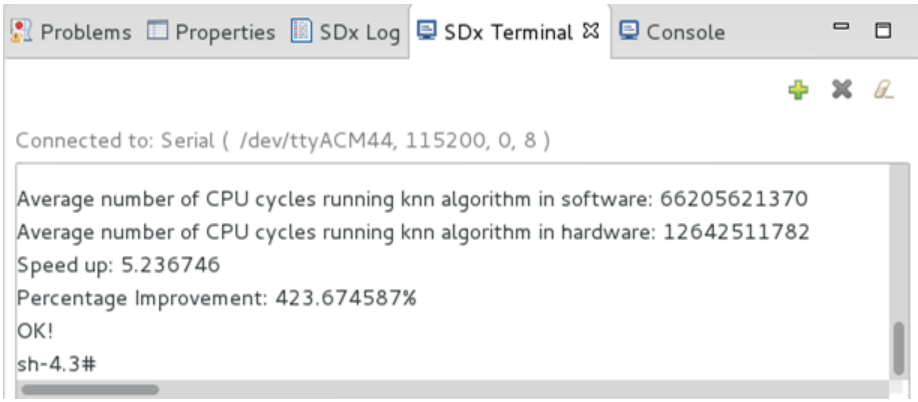
Το Hardware buffer depth (όπου ορίζεται με την εντολή #pragma SDS data buffer_depth και στη συγκεκριμένη περίπτωση είναι 3) θα πρέπει να ρυθμιστεί στην ίδια τιμή με το pipeline_depth. Ο στόχος αυτού του pipeline είναι η μεταφορά δεδομένων στον επιταχυντή για την επόμενη εκτέλεση, ενώ η τρέχουσα εκτέλεση δεν έχει ολοκληρωθεί!

Παρακάτω στον κώδικα προστέθηκαν δύο ακόμα εντολές: το #pragma SDS data buffer_depth με βάση το οποίο όπως προαναφέρθηκε καθορίζεται το βάθος του buffer καθώς και το #pragma SDS data data_mover όπου εισάγει τον data mover AXI DMA SG ο οποίος είναι απαραίτητος για να εφαρμοστεί η εντολή Task-Level Pipeline. Η σύνταξη του κώδικα γράφεται ως εξής:

DATA MOVERS:

```
1 #pragmaSDSdatabuffer_depth (samplesu: 3, group: 3)
2 #pragma SDS data data_mover (samplesu: AXIDMA_SG:1, group: AXIDMA_SG:1)
3 #pragma SDS data mem_attribute (
        samples: PHYSICAL_CONTIGUOUS|CACHEABLE,
        samplesu: PHYSICAL_CONTIGUOUS|CACHEABLE,
        group: PHYSICAL_CONTIGUOUS|CACHEABLE)
4 #pragma SDS data copy (samples [0: h*w])
5 #pragma SDS data access_pattern (samples: SEQUENTIAL)
6  void knn_algorithm_hw (
  double *samples, doublesamplesu [MAT*4], doublegroup [MAT], int h, int w, int u);
```

Στην παρακάτω εικόνα(**Εικόνα 39:** Απόδοση Συστήματος (Task-Level Pipeline με BUFFERS)) η απόδοση της pipeline τεχνικής σε ένα επιταχυντή (accelerator) είναι κατά ελάχιστα καλύτερη σε σχέση με την προγουμενη τεχνική (προηγουμένως η απόδοση ήταν περίπου 400% ενώ σε αυτό το παράδειγμα είναι 423%). Στην ουσία η απόδοση είναι όσο ήταν όταν τα δείγματα ήταν 50 (υποκεφάλαιο **5.1.2**.) αντί για 10.000 που είναι σε αυτό το παράδειγμα.



Εικόνα 39:Απόδοση Συστήματος (Task-LevelPipeline με BUFFERS)

Στον πίνακα παρακάτω (**Πίνακας 30:** Data Motion Report (Task-Level Pipeline με BUFFERS)), εξακριβώνεται ότι υπάρχει ένας επιταχυντής. Στη στήλη «pragmas» αναγράφονται οι εντολές pragma που χρησιμοποιήθηκαν. Στη στήλη «declared size (bytes)» στον πίνακα samplesu το μέγεθος φαίνεται να είναι 4000*8 (1000*4*8) αυτό συμβαίνει διότι έχει καθοριστεί ένας πίνακας δεικτών 10 θέσεων όπου κάθε δείκτης δείχνει σε 1000*4*8 , συνεπώς τα μη κατηγοριοποιημένα δείγματα (στα οποία δεν υπάρχει εξάρτηση) επεξεργάζονται ανά χιλιάδες.

Στον επόμενο πίνακα (**Πίνακας31:** Accelerator Callsites (Task-Level Pipeline με BUFFERS)) αναγράφονται οι κλήσεις που έγιναν στον επιταχυντή φαίνονται 2 διότι 2 φορές έχει γίνει κλήση της συνάρτησης όμως και οι 2 κλήσεις βρίσκονται μέσα σε βρόγχους όπου το σύνολο των επαναλήψεων είναι 10 όσο δηλαδή και ο πίνακας δεικτών που στο σύνολο είναι 10.000 δείγματα.

Τέλος στον **Πίνακα 32:** Ποσοστό χρήσης πόρων του fpga (Task-Level Pipeline με BUFFERS) πρέπει να σημειωθεί ότι δεν υπήρξε καμία αύξηση στη δέσμευση των πόρων του fpga εφόσον εξακολουθεί να υφίσταται μόνο ένας επιταχυντής.

Accelerator	Argument	IP Port	Direction	Declared Size(bytes)	Pragmas	Connection
knn_algorithm_hw_1	samples	samples	IN	0*8	• length:(h*w) • mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE	ps7_S_AXI_ACP:AXIDMA_SG
	samplesu	samplesu_PORTA	IN	4000*8	• mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE • buffer_depth:3 • data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	group	group_r_PORTA	OUT	1000*8	• mem_attribute:PHYSICAL_CONTIGUOUS.CACHEABLE • buffer_depth:3 • data_mover:AXIDMA_SG	ps7_S_AXI_ACP:AXIDMA_SG
	h	h	IN	4		ps7_M_AXI_GP0:AXILITE:0xC
	w	w	IN	4		ps7_M_AXI_GP0:AXILITE:0x10
	u	u	IN	4		ps7_M_AXI_GP0:AXILITE:0x14

Πίνακας 30: Data Motion Report (Task-Level Pipeline με BUFFERS)

Accelerator	Callsite	IP Port	Transfer Size(bytes)	Paged or Contiguous	Datamover Setup Time(CPU cycles)	Transfer Time(CPU cycles)
knn_algorithm_hw_1	async_buff.cpp:161:3	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	9188	9859282
		samplesu_PORTA	32000	PHYSICAL_CONTIGUOUS	5285	43808
		group_r_PORTA	8000	PHYSICAL_CONTIGUOUS	5176	15222
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13
	async_buff.cpp:156:3	samples	(h*w) * 8	PHYSICAL_CONTIGUOUS	9188	9859282
		samplesu_PORTA	32000	PHYSICAL_CONTIGUOUS	5285	43808
		group_r_PORTA	8000	PHYSICAL_CONTIGUOUS	5176	15222
		h	4	paged	0	13
		w	4	paged	0	13
		u	4	paged	0	13

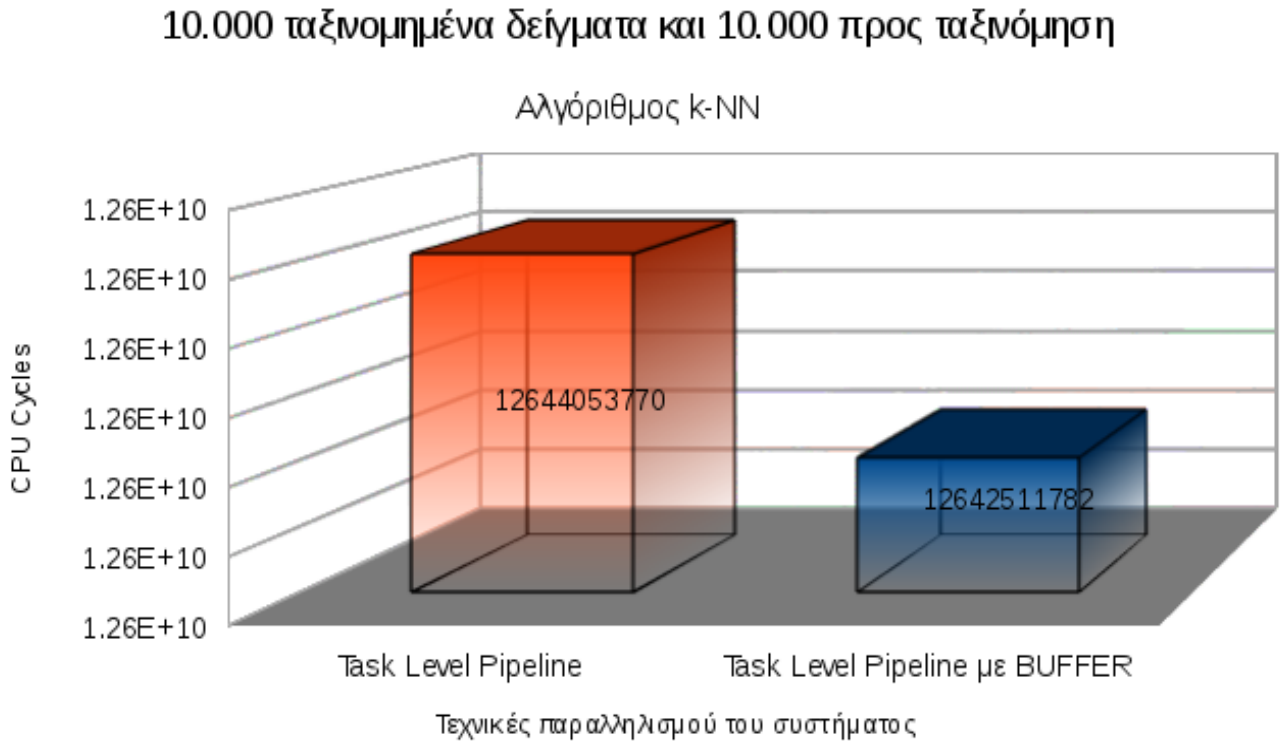
Πίνακας 31: Accelerator Callsites (Task-Level Pipeline με BUFFERS)

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	0	21
FIFO	-	-	-	-
Instance	20	34	11337	13725
Memory	144	-	256	8
Multiplexer	-	-	-	8
Register	-	-	8	-
Total	164	34	11601	13762
Available	280	220	106400	53200
Utilization (%)	58	15	10	25

Πίνακας 32: Ποσοστόχρήσηςπόρωντουfpga (Task-LevelPipeline με BUFFERS)

Σε αυτό το παράδειγμα δεν χρησιμοποιήθηκε προηγούμενη βέλτιστη λύση (η τεχνική resource) και ο λόγος είναι ότι απαιτεί πολλά resources (με την τροποποίηση του έγινε στον κώδικα και την αύξηση των δειγμάτων) με αποτέλεσμα το block design να είναι τόσο μεγάλο ώστε να μην χωράει στην πλακέτα. Γι'αυτό το λόγο στην περίπτωση όπου ο αλγόριθμος απαιτεί πολλές κλήσεις προς μια Hardware συνάρτηση συνίσταται η τεχνική Task-Level Pipeline.

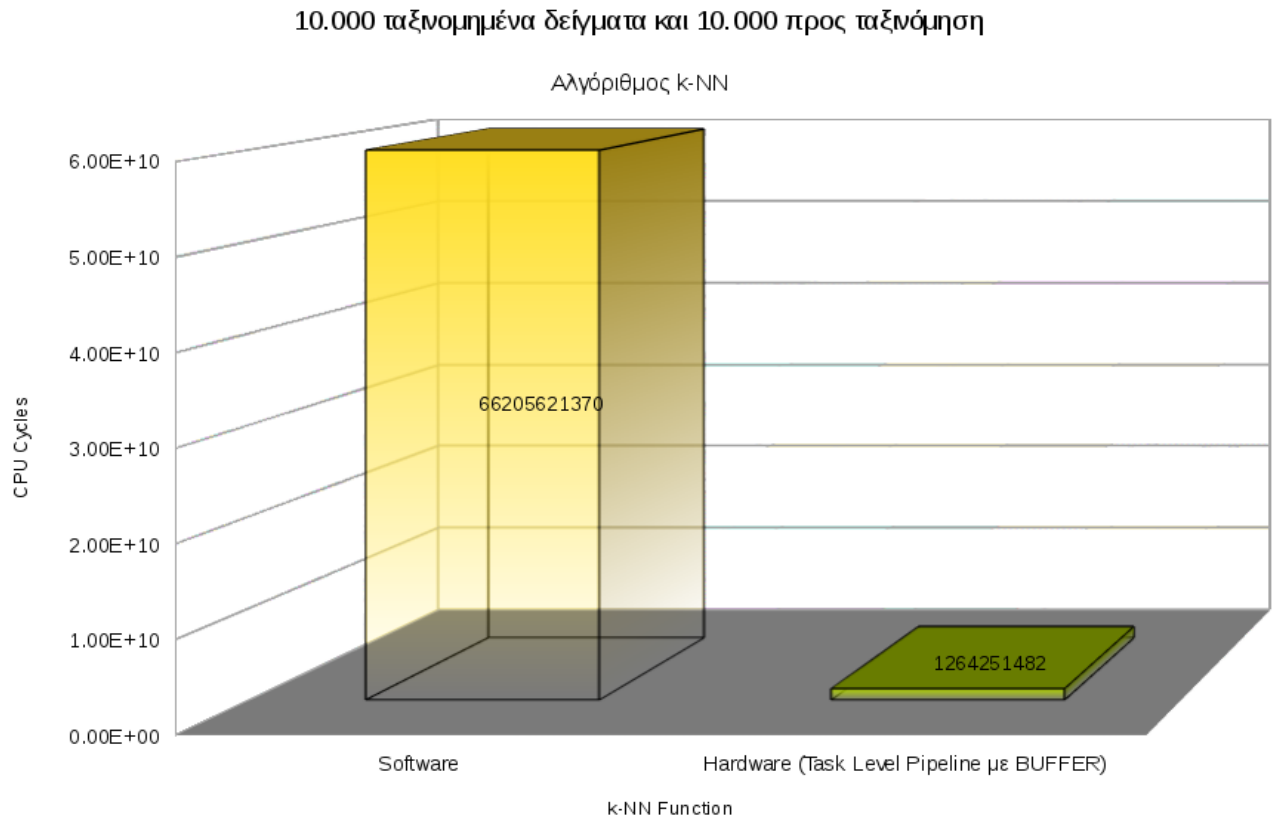
Η Task-Level Pipeline με buffer τεχνική σε αυτό το παράδειγμα δεν φαίνεται να έχει μεγάλη διαφορά σε σχέση με την τεχνική Task-Level Pipeline χωρίς buffer (**Εικόνα 40:**Σύγκρισηαπόδοσης του k-NNαλγορίθμου με διαφορετικές τεχνικές παραλληλοποίησης (Task-Level Pipeline, Task-Level Pipeline με BUFFER)) ο λόγος είναι ότι με την τεχνική αυτή η διαφορά στην απόδοση φαίνεται μόνο όταν οι κλήσεις προς την Hardware συνάρτηση ξεπερνούν τις 100 ενώ σε αυτό το παράδειγμα είναι μόνο 10. Οπότε η βελτίωση 423% θεωρείται μια καλή απόδοση για 20.000 δειγματα.Η διαφορά που έχουν αυτοι οι δύο τρόποι υλοποίησης αυτής της εντολής είναι ότι το Task-Level Pipeline με buffer χρησιμοποιεί buffers για να μπορέσει να φερεί τοπικά περισσότερους πίνακες(το πόσοι καθορίζεται από το buffer_depth) όποτε , αν υπάρχουν πολλοί πίνακες όπως σε αυτό το παράδειγματα ο δεύτερος τρόπος υλοποίησης της συνάρτησης είναι καλύτερος.



Εικόνα 40:Σύγκριση απόδοσης του k-NN αλγορίθμου με διαφορετικές τεχνικές παραλληλοποίησης (Task-Level Pipeline, Task-Level Pipeline με BUFFER)

Στο παρακάτω γράφημα (**Εικόνα 41:** Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)) απεικονίζεται η διαφορά ανάμεσα στην συνάρτηση Hardware και Software. Στο Hardware οι κύκλοι στη CPU είναι εμφανώς πιο μειωμένοι σε σχέση με το Software. Για τη σύγκριση χρησιμοποιήθηκε η τεχνική Task-Level Pipeline με BUFFER η οποία είναι κατά ελάχιστα πιο βελτιωμένη σε σχέση με την Task-Level Pipeline τεχνική για τον λόγο που προαναφέρθηκε. Ωστόσο η

απόδοση εξακολουθεί να είναι καλή παρόλο που τα δείγματα αυξήθηκαν. Η πτώση της απόδοσης σε σχέση με τα προηγούμενα παραδείγματα οφείλεται στο γεγονός ότι τα δείγματα αυξήθηκαν και δεν χωρούν να μεταφερθούν όλα απευθείας στη μνήμη B-RAM.



Εικόνα 41: Σύγκριση απόδοσης του k-NN αλγορίθμου (Software Vs Hardware)

6.3 Τεχνικές Διάσπασης Πινάκων και Τοποθέτησης τους στη Block-RAM

Τα δεδομένα αντιγράφονται τοπικά μέσα στη συνάρτηση ούτως ώστε να αποθηκευτούν στη block-RAM. Με αυτό το τρόπο δεν χρειάζεται επιστρέφει στη DDR και να δημιουργεί καθυστέρηση.

Με την παρακάτω υλοποίηση της partition εντολής η διάσταση 2 του πίνακα διασπάται ολόκληρη με τη μια φορά , δηλαδή, όλα τα στοιχεία της κάθε στήλης θα διαβάζονται ή θα γράφονται παράλληλα.

Σε αυτή τη περίπτωση είναι προκαθορισμένο το complete.

Με την επόμενη εντολη resource δεσμεύονται πόροι μόνο από τη B-RAM. Αυτό είναι χρήσιμο διότι οι περισσότερες εντολές χρησιμοποιούν πολλά resources από το fpga κυρίως LUTS και FFμε αποτέλεσμαπολλές φορές το designναξεπερνά το προκαθορισμένο όριο των πόρων του fpga(το design δηλαδή δεν χωράει στο board) ενώ η B-RAM να διαθέτει αρκετό χώρο προς εκμετάλευση.

ARRAY_PARTITION &RESOURCE:

```
#pragma HLS ARRAY_PARTITION variable=deigma dim=2
#pragma HLS RESOURCE variable=deigma core=RAM_2P_BRAM
```

6.4 Τεχνικές για Παραλληλισμό Λειτουργιών Μέσα σε Βρόγχο

Με αυτή την εντολή επιδιώκεται ο παραλληλισμός τωνεπαναλήψεων ενός βρόγχου. Η loop pipeline δίνει τη δυνατότητα στις επαναλήψεις να γίνουν με ταυτόχρονο τρόπο όπως φαίνεται και στον ακόλουθο κώδικα. Στο επόμενο παράδειγμα με το II (Initiation Interval) καθορίζεται ο αριθμός των κύκλων όπου θα ξεκινήσει η επόμενη λειτουργία καθιστώντας το πιο αποδοτικό, ωστόσο πιθανόν να μην μπορεί να υλοποιήσει τους κύκλους που τυχόν ορίστούν. Σε αυτή τη περίπτωση το πρόγραμμα κρίνει μόνο του μετά από πόσους κύκλους θα ξεκινήσει η επόμενη επανάληψη αυτό θα συμβεί επίσης και στη περίπτωση που δεν εισαχθεί το II. Η εντολή εισάγεται ακριβώς πάνω από το βρόγχο.

PIPELINE:

```
for (intq=0; q<H; q+=1000)
{
	for (int h=0;h<1000;h++)
	{
		for(int m=0;m<W;m++)
		{
		#pragma HLS PIPELINE
			deigma[h][m] =samples [q + h * W + m];
		}
	} eukleideia_best_voters (deigma, deigma1, best_voters, u1);
}
```

PIPELINE II:

```
 for (i=0; i<u1; i++)
{
	for(int m=0;m<W;m++)
	{
	#pragma HLS PIPELINE II=1
		deigma1 [i][m]=  samplesu[ i * W + m ];
	}
}
```

Με την εντολή loopunrollπαραλληλίζονται οι επαναλήψεις του βρόγχου μεταξύ τους. Δημιουργεί

πολλαπλά αντίγραφα των λειτουργιών του βρόγχου και ρυθμίζοντας τον «factor» μετρητή ανάλογα με τις επαναλήψεις που ο προγραμματιστής επιθυμεί να παραλληλιστούν. Δημιουργεί όσα block του αναθέτουν (από το factor) αν χωράνε στο fpga φυσικά και τα παραλληλίζει. Στο παρακάτω παράδειγμα ξετυλίγει τη loop (βρόγχο) ανά 10 επαναλήψεις .

UNROLL:

```
for (i=0; i<u1; i++)
{
        for (int m=0; m<W; m++)
        {
        #pragma HLS PIPELINE II=1
                deigma1 [i][m] = samplesu[ i * W + m ];
        }
}
```

Μια τεχνική για μεγαλύτερο παραλληλισμό αποτελεί ο παρακάτω κώδικας διότι εκτός της unroll εντολής εσωτερικά υπάρχει και η pipeline εντολή. Διαμορφώνοντας λοιπόν με αυτό το τρόπο τον κώδικα γίνεται το εξής:

ξετυλίγονται δέκα μη ταξινομημένα δείγματα, δηλαδή δημιουργούνται 10 blocks όπου μέσα στα blocks οι λειτουργίες (δηλαδή ο εσωτερικός βρόγχος) παραλληλίζονται με την pipeline εντολή. Με αυτόν τον τρόπο επιτυγχάνεται μεγαλύτερη βελτίωση απόδοσης χρόνου της Hardware συνάρτησης.

UNROLL & PIPELINE:

```
 for (i=0; i<u1; i++)
{
        #pragma HLS unroll factor =10
                for (sampleIndex=0; sampleIndex<1000; sampleIndex++)
                {
                #pragmaHLSPIPELINE
                        .
                        .
                        .
                }
}
```

6.5 Τεχνικές για Παραλληλισμό Βρόγχων

Με αυτήν την εντολή παραλληλοποιούνται οι βρόγχοι που βρίσκονται κάτω από αυτή την εντολή. Η εντολή αυτή πρεπει να εισαχθεί στην αρχή της Hardware συνάρτησης ούτως όλοι οι βρόγχοι που βρίσκονται κάτω από αυτή την εντολή να τις παραλλήλοποιήθούν. Αντί να γίνει σειριακά η εκτέλεση των βρόγχων (ο ένας δηλαδή μετά τον άλλο) , γίνονται pipeline (πριν προλάβει να ολοκληρωθεί ο προηγούμενος βρόγχος δηλαδή να ξεκινήσει ο επόμενος βρόχος).

DATAFLOW:

```
#pragma HLS dataflow
loop1
   loop2
      loop3
```

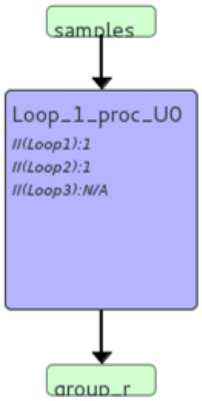
Η τεχνική αυτή σε συνδυασμό με looppipeline (εκτελώντας δηλαδή τη pipeline εντολή για το βρόγχο για παραλληλισμό των επαναλήψεων) ή με unroll ή και τα δύο μαζί φέρει καλύτερα αποτελέσματα.

DATAFLOW& UNROLL &PIPELINE:

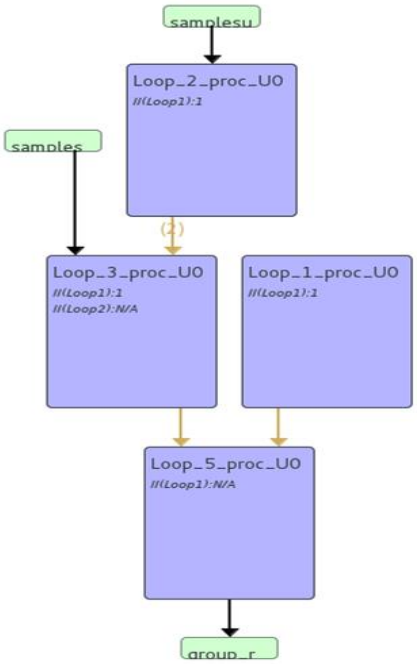
```
#pragma HLS dataflow

loop1:for (i=0; i<H; i++)
      {
            for (int m=0; m<W; m++)
            {
            #pragma HLS PIPELINE II=1
                  deigma [i][m]=  samples[ i * W + m ];
            }
      }
loop2:   for(i=0; i<u1; i++)
      {
            for (int m=0; m<W; m++)
            {
            #pragma HLS PIPELINE II=1
                  deigma1 [i][m]=  samplesu[ i * W + m ];
            }
```

```
        }
loop3:   for (i=0; i<u1; i++)
         {
         #pragma HLS unroll factor =10
              for (sampleIndex=0; sampleIndex<1000; sampleIndex++)
              {
              #pragmaHLSPIPELINE
                        .
                        .
}
```

Ανάλογα με την εισαγωγή των δεδομένων στα προηγούμενα παραδείγματα η εντολή dataflow συμπεριφέρεται διαφορετικά. Στην παρακάτω εικόνα (**Εικόνα 42:** Συμπεριφορά dataflow στον κώδικα με μια κλήση προς τον επιταχυντή) αναγράφονται τρείς βρόγχοι (δημιουργία πίνακα groups, αντιγραφή στην τοπικη μνήμη του samplesu, πολλαπλή αντιγραφη στην τοπική μνήμη το samples ανα 1000 δείγματα και στην ίδιο βρόγχο η επανάληψη της υποσυνάρτησης) αυτοί παραλληλίζονται. Στην επόμενη εικόνα (**Εικόνα 43:** Συμπεριφορά dataflow στον κώδικα με πολλαπλές κλήσεις προς τον επιταχυντή) υπάρχουν παραπάνω βρόχοι (κάθε 1000 samplesu υλοποιείται ένας βρόγχος (loop2 proc) ,το ίδιο θα συμβεί και για κάθε 1000 samples(loop 3 proc) και ο βρόγχος της υποσυνάρτησης , στην loop 1 proc είναι τα groups και στην τελευταία είναι υπολογίζεται το group του δείγματος).



Εικόνα 42: Συμπεριφορά dataflow στον κώδικα με μια κλήση προς τον επιταχυντή



Εικόνα 43: Συμπεριφορά dataflow στον κώδικα με πολλαπλές κλήσεις προς τον επιταχυντή

6.6 Τεχνικές για Υποσυναρτήσεις

Η εντολή inline τοποθετείται μέσα στη συνάρτηση ως πρώτη εντολή. Η εντολή inline ζητάει από τον compiler να αντικαταστήσει κάθε κλήση τη συνάρτησης στο πρόγραμμα με ένα αντίγραφο του κώδικα της συνάρτησης. Τοποθετώντας αυτή την εντολή μειώνεται η επιβάρυνση που γίνεται με την κλήση της συνάρτησης. Αποτελεί καλή περίπτωση για συχνά καλούμενες συναρτήσεις. Αυτή η εντολή μπορεί να συνδυαστεί με τις παραπάνω τεχνικές κάτω από την inlineεντολή και να φέρει πολύ καλά αποτελέσματα.

INLINE:

```
Void eukleideia_best_voters (double deigma [1000][4],double deigma1[U][4],double
best_voters [K*U][4] , int u1){

        #pragma HLS inline

                .
                .
                }
```

7. Αποτελέσματα

Σε αυτό το κεφάλαιο αναγράφονται συγκεντρωτικά οι καλύτερες μετρήσεις του αλγορίθμου ανάλογα τον αριθμό των δειγμάτων. Επίσης αναφέρονται τρόποι για τη βελτίωση ενός αλγορίθμου και αναφέρονται λύσεις το πρόβλημα του μεγάλου όγκου δεδομένων όσον αφορά τη διαχείρηση του.

7.1 Μετρήσεις

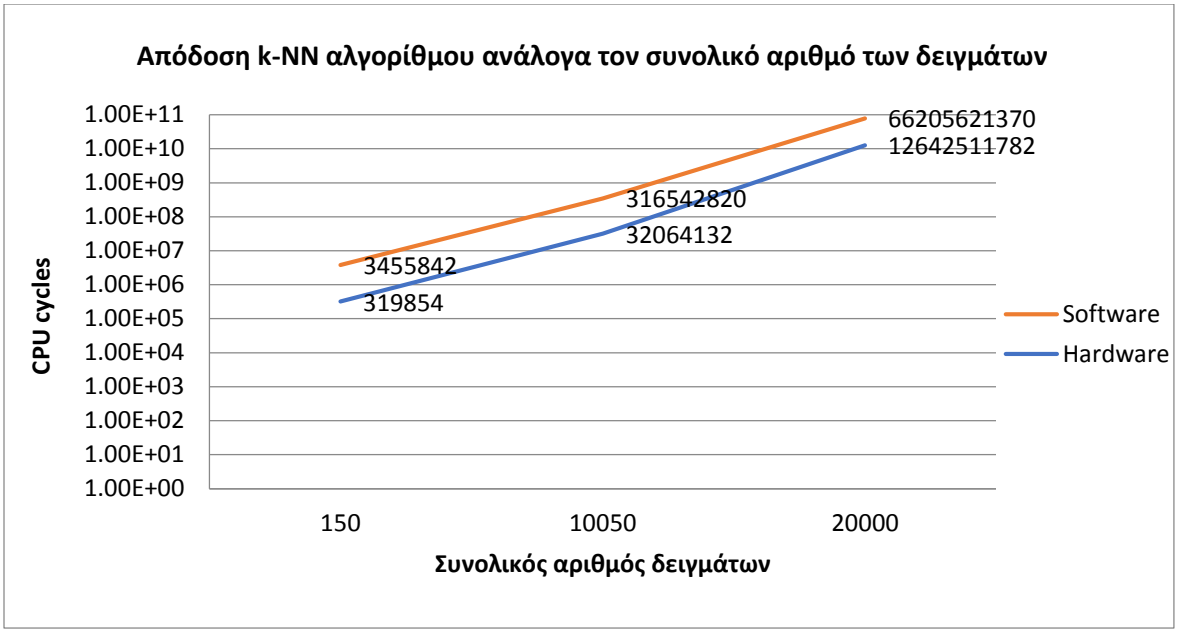
Στο προηγούμενο κεφάλαιο αναλύθηκαν διάφορα παραδείγματα τα οποία είχαν στόχο την προσέγγιση της καλύτερης απόδοσης του αλγορίθμου ανάλογα με τον όγκο των δεδομένων.

Με βάση τις παραπάνω μετρήσεις από τα παραδείγματα που αναλύθηκαν προκύπτουν τα γραφήματα παρακάτω (**Εικόνα 44:** Συνολικό γράφημα μετρήσεων απόδοσης αλγορίθμου σε Hardware και **Εικόνα 45:** Γράφημα ποσοστιαίων μετρήσεων απόδοσης αλγορίθμου σε Hardware). Στο πρώτο γράφημα απεικονίζονται οι βέλτιστες αποδόσεις σε CPUκύκλους του κάθε παραδείγματος της συνολικής εργασίας ενώ το δεύτερο γράαφημα απεικονίζονται οι βέλτιστες αποδόσεις σε ποσοστά βελτίωσης τουHardware σε σχεση με το Software.

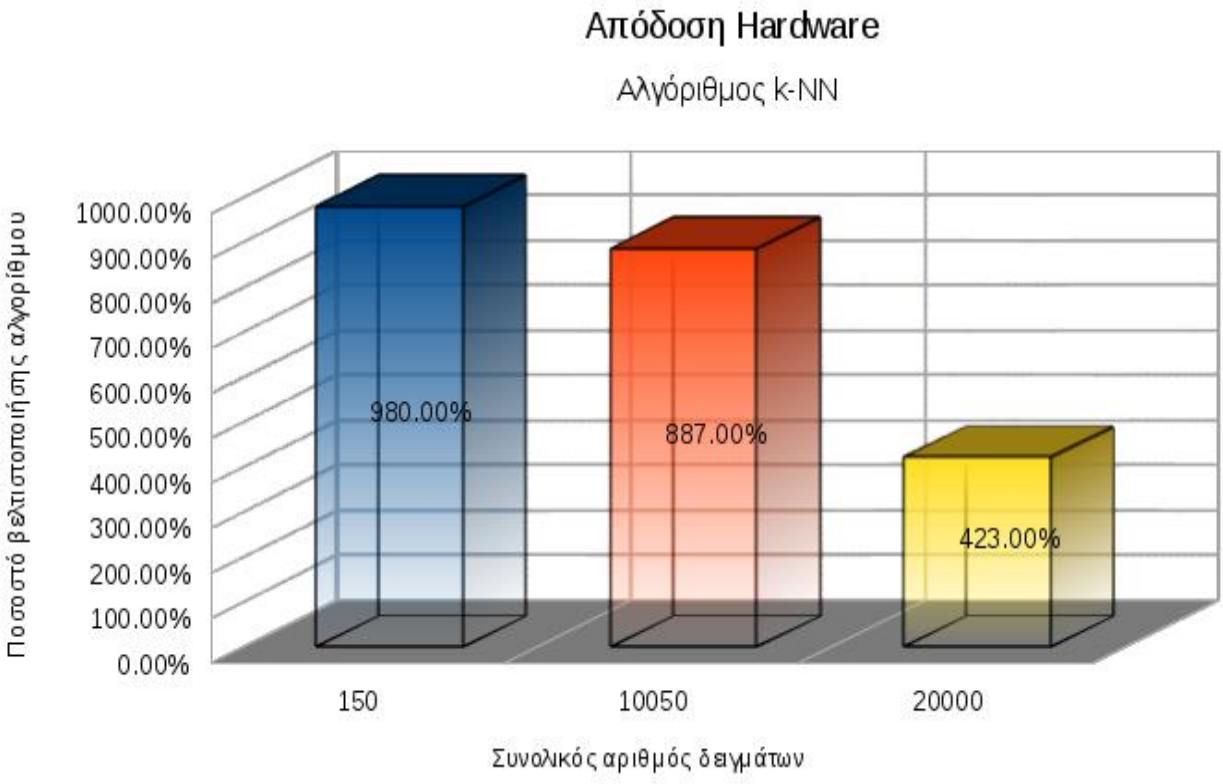
Στο πρώτο γράφημα η μπλε γραμμη που απεικονίζεται αναφέρεται στις μετρήσεις της απόδοσης του Hardware ενώ η κόκκινη γραμμη αναφέρεται στις Software μετρήσεις.

Στην πρώτη σύγκρισηανάμεσα σε Hardwareκαι Softwareαπόδοση, ο αριθμός των δειγμάτων είναι 150 (100 κατηγοριοποιημένα και 50 μη κατηγοριοποιημένα) ,στη δεύτερη σύγκριση 10.000 (10.000 κατηγοριοποιημένα και 50 μη κατηγοριοποιημένα) και στη τρίτη 20.000 (10.000 κατηγοριοποιημένα και 10.000 μη κατηγοριοποιημένα). Στο δεύτερο γράφημα για 150 δείγματα η βελτίωση του Hardwareσε σχέση με το Software είναι 980%. Στην δεύτερη στήλη το ποσοστό βελτίωσης ανέρχεται περίπου στα 887% για 10.050 δείγματα. Στην τρίτη στήλη το ποσοστό βελτίωσηςείναι 423% για 20.000 .

Στα 20.000 δείγματα υπάρχει μια πτώση της απόδοσης του αλγορίθμου σε Hardware. Αυτό συνέβη λόγω της αύξησης των δειγμάτων. Οι πόροι του fpga είναι περιορισμένοι με αποτέλεσμα αν δεν μπορούν όλα τα δεδομένα να μεταφερθούν στη τοπική μνήμη απευθείας να υπάρχει κάποια καθυστέρηση η οποία στερεί την βελτιστοποίηση που θα είχε ο αλγόριθμος σε αντίθετη περίπτωση.



Εικόνα 44:Συνολικό γράφημα μετρήσεων απόδοσης αλγορίθμου σε Hardware



Εικόνα 45:Γράφημα ποσοστιαίων μετρήσεων απόδοσης αλγορίθμου σε Hardware

8. Συμπεράσματα

Το Hardware είναι πολύ ωφέλιμο και πολύ αποδοτικότερο σε σχέση με το Software αρκεί να εκμεταλευτει σωστά.

Για τη βελτίωση ενός αλγορίθμου σημαντικό ρόλο παίζει η μετακίνηση των δεδομένων (data mover), η διαχείρηση πολλαπλών κλήσεων , η παραλληλοποιήση εσωτερικών λειτουργιών του αλγορίθμου , διαχείσηση μεγάλου όγκου δεδομένων καθώς και η όσο το δυνατόν καλύτερη Software προσέγγιση του αλγορίθμου στα μέτρα του Hardware.

Το βασικό όμως πρόβλημα στο οποίο εστιάζει η εργασία είναι ο μεγάλος όγκος δεδομένων όπου για να επιτευχθεί μια καλή απόδοση τα δεδομένα θα πρέπει να μετακινηθούν στην τοπική μνήμη του FPGA.

Στην περίπτωση που τα δεδομένα δεν χωρούν στη μνήμη B-RAM:

1. **Όταν δεν υπάρχει εξάρτηση δεδομένων (μη ταξινομημένα δείγματα)**μια λύση που εφαρμόστηκε είναι η διάσπαση των δεδομένων. Το αρχικό σετ δειγμάτων κατατμείται σε μικρότερους πίνακες οι οποίοι περνούν μέσα από τη Hardware συνάρτηση. Αρχικά αυτό γίνεται για να μεταφερθούν τα δεδομένα λίγα-λίγα και να τοποθετηθούν στη μνήμη B-RAM. Το κάθε σετ των χιλίων μόλις επεξεργαστεί πλήρως αντικαταστείται με το επόμενο σετ.

   Αυτό είναι απαραίτητο προκειμένου τα δεδομένα να βρίσκονται κοντά στο υπολογιστικό μέρος (computation).Επίσης με τον τρόπο αυτό υλοποιείται και η δυνατότητα του παραλληλισμού ανάμεσα στις κλήσεις των συναρτήσεων όπου κάθε φορά η συνάρτηση θα καλείται με θα υλοποιείται με διαφορετικά δεδομένα (δηλαδή με τοεπόμενο σετ πίνακα).

2. **Όταν υπάρχει εξάρτηση των δεδομένων (ταξινομημένα δείγματα)** όλα τα δεδομένα πρέπει να περάσουναπό τη Hardware συνάρτηση (στο συγκεκριμένο αλγόριθμο κάθε μη ταξινομημένο δείγμα πρέπει να μετρήσει την απόσταση σε σχέση με όλα τα ταξινομημένα συνεπώς όλα τα ταξινομημένα δείγματα πρέπει να περάσουν από τη Hardware συνάρτηση) τότε το αρχικό σετ δειγματων κατατέμνεται και σε αυτή την περίπτωση σε μικρότερους πίνακες οι οποίοι περνούν μέσα από μια Hardware υπόσυνάρτηση. Με αυτόν τον τρόπο η υποσυνάρτηση μπορεί να βελτιωθεί όπως και η Hardware συνάρτηση.

Σε καμία περίπτωση δεν εξυπηρετεί να μεταφερονται πολλά δεδομένα (δηλαδή τόσα ώστε να μη χωρούν στη B-RAM) μέσα από μια Hardware συνάρτηση εφόσον δεν υπάρχει εξάρτηση. Το fpga είναι αποδοτικό επειδή η μνήμη B-RAM βρίσκεται δίπλα από υπολογιστικό μέρος. Αν τα δεδομένα βρίσκονται στη DDR θα υπάρξει μεγάλη καθυστέρηση και ο χρόνος δεν θα είναι αποδοτικός για την ακρίβεια με αυτόν το τρόπο δεν θα διαφοροποιείται από το Software. Συνεπώς οι πόροι του fpga θα πρέπει να διαχειρίζονται σωστά για να υπάρξει απόδοση.

Ωστόσο υπάρχουν περιπτώσεις όπου αν τα δεδομένα είναι υπερβολικά πολλά η απόδοση του συστήματος πιθανόν να μην είναι και τόσο καλή όσο θα ήταν με λιγότερα. Επίσης είναι πιθανόν το block που προκύπτει να είναι τόσο μεγάλο ώστε να μην χωράει στη πλακέτα ή τα δεδομένα να είναι τόσα πολλά που να μην χωρούν σε συνεχόμενη μνήμη. Επίσης η επιλογή των data movers περιορίζεται (εξαιτίας της πλακέτας) και οι διαθέσιμοidata movers δεν είναι τόσο αποδοτικοί όσο κάποιοι άλλοι. Αυτά είναι κάποια μειονεκτήματα. Το συμπέρασμα που προκύπτει είναι ότι μετά από ένα σημείο όσο αυξάνονται τα δεδομένα τόσο μειώνεται η απόδοση. Αυτό θα μπορούσε φυσικά να διορθωθεί επιλέγοντας κάποια άλλη πλακέτα με ένα fpga του οποίου οι διαθέσιμοι πόροι θα ήταν περισσότεροι.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] k-nearest neighbors algorithm, https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

[2] https://github.com/fahadmuslim/KNN

[3] https://en.wikipedia.org/wiki/Nonparametric_statistics

[4] https://en.wikipedia.org/wiki/Instance-based_learning

[5] Saravanan Thirumuruganathan, "A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm", 2010, https://saravananthirumuruganathan.wordpress.com/2010/05/17/a-detailed-introduction-to-k-nearest-neighbor-knn-algorithm/

[6] Y. Lee, "Handwritten Digit Recognition Using K Nearest-Neighbor, Radial-Basis Function, and Back-propagation Neural Networks", Neural Computation, Fall 1991, Vol. 3, No. 3, Pages 440-449, (doi: 10.1162/neco.1991.3.3.440)

[7] G. Kornaros, "Application Specific Customizable Embedded Systems", Book Chapter 2, in "Multi-Core Embedded Systems", CRC Press/Taylor & Francis Group, Apr. 2010, ISBN: 978-1-4398-1161-0.

[8] http://scikit-learn.org/stable/modules/neighbors.html

[9] http://www.saedsayad.com/k_nearest_neighbors.htm

[10] https://www.quora.com/What-are-industry-applications-of-the-K-nearest-neighbor-algorithm

[11] http://www.hertasecurity.com/en

[12] https://el.wikipedia.org/wiki/Μηχανική_μάθηση

[13] https://forums.xilinx.com/t5/SDSoC-Development-Environment/HW-function-interface-maximum-array-size/td-p/713670

[14] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/data-motion-network-gen/concept-data-motion-network-gen.html

[15] https://en.wikipedia.org/wiki/Field-programmable_gate_array

[16] https://el.wikipedia.org/wiki/Άμεση_πρόσβαση_μνήμης

[17] https://www.xilinx.com/support/documentation/white_papers/wp459-data-mover-IP-zynq.pdf

[18] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_3/ug1235-sdsoc-optimization-guide.pdf

[19] SDSoC Environment User Guide, https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2_1/ug1027-intro-to-sdsoc.pdf

[20] AXI Reference Guide, UG761 (v13.1) March 7, 2011, https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/v13_4/ug761_axi_reference_guide.pdf

[21] I. Christoforakis, O. Tomoutzoglou, D. Bakoyiannis and G. Kornaros, "Dithering-Based Power and Thermal Management on FPGA-Based Multi-Core Embedded Systems.", In Proc. of the 13th IEEE International Conference on Embedded and Ubiquitous Computing (EUC), 2015, pp. 173-177

[22] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/sdsoc_doc/topics/calling-coding-guidelines/reference-hls-function-arguments.html

[23] https://www.xilinx.com/products/intellectual-property/axi_fifo.html

[24] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/ref-pragma_SDS_resource.html

[25] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/ref-pragma_HLS_unroll.html

[26] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/ref-pragma_HLS_dataflow.html

[27] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/sdsoc_doc/topics/calling-coding-guidelines/concept_function_inlining.html

[28] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/ref-pragma_HLS_pipeline.html

[29] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/ref-pragma_HLS_array_partition.html

[30] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/concept-Intro_to_HLS_pragmas.html

[31] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/concept-Intro_to_SDS_pragmas.html

[32] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/pragmas/concept-hardware-buffer-depth-pragma.html

[33] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/tutorials/concept_lab-task-level-pipelining.html

[34] https://www.xilinx.com/html_docs/xilinx2017_1/sdsoc_doc/topics/data-motion-network-gen/concept-data-motion-network-gen.html

[35] https://www.xilinx.com/products/design-tools/software-zone/sdnet.html

[36] https://www.xilinx.com/products/design-tools/software-zone/sdsoc.html

[37] https://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[38] https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug1027-intro-to-sdsoc.pdf

[39] M. Coppola, B. Falsafi, J. Goodacre and G. Kornaros, “From embedded multi-core SoCs to scale-out processors”, In Proceedings of the Conference on Design, Automation and Test in Europe, pp. 947-951, 2013

[40] G. Kornaros, (Ed.), “Multi-core embedded systems”, CRC Press/Taylor & Francis Group, 07-April-2010, ISBN: 978-1-4398-1161-0

[41] O. Tomoutzoglou, D. Bakoyannis, G. Kornaros and M. Coppola, "Efficient communication in heterogeneous SoCs with unified address space," In Proc. of the 11th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC), Tallinn, 2016, pp. 1-6