Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Δ.Π.Μ.Σ. Επιστήμης Δεδομένων
και Μηχανικής Μάθησης

Παράλληλες Αρχιτεκτονικές για Μηχανική Μάθηση
Εργαστηριακή Άσκηση - FPGAs

July 28, 2021

1 Εισαγωγή

Σκοπός της συγκεκριμένης εργασίας ήταν η ενασχόληση με τους επιταγχυντές τύπου Field Programmable Gate Array (FPGA) μέσω της πλατφόρμας του InAccel.

Σε αυτό το εργαστήριο αρχικά υλοποιήθηκε ένα πολύ μικρό δείγμα λειτουργίας του inaccel μέσω του jupyterhub environment όπου ασχοληθήκαμε με την ανάθεση πόρων για την δημιουργία τεσσάρων διανυσμάτων πάνω στα οποία εφαρμόστηκαν αριθμητικές πράξεις. Ύστερα, έγιναν συγκρίσεις αναφορικά με την χρονική απόδοση του fpga και της αντίστοιχης υλοποίησης πράξεων μέσω numpy. Εν συνεχεία, ασχοληθήκαμε με την πρώτη μας πραγματική εφαρμογή στην οποία η προσοχή μας συγκεντρώθηκε στην *σύγκριση μιας προσαρμοσμένης υλοποίησης του Naive Bayes Classifier σε FPGA σε σχέση με αυτήν του sci-kit learn*, τόσο από άποψη χρόνου αλλά και από άποψη ακρίβειας του ταξινομητή. Τέλος, και προκειμένου να καταστεί αισθητή μια ακόμη χρήση που ενδείκνυται η χρήση των FPGA δημιουργήσαμε ένα notebook στο οποίο εκτελέστηκε η *βελτιστοποίηση υπερπαραμέτρων σε ένα πρόβλημα λογιστικής παλινδρόμησης με την μέθοδο grid-search* μια φορά με την χρήση επιταγχυντών και μία χωρίς.

2 Αρχική Εφαρμογή

Στο συγκεκριμένο κομμάτι της άσκησης παρατηρούμε τον τρόπο που γίνεται η δημιουργία των διανυσμάτων τα οποία ύστερα θα γεμίσουμε με στοιχεία με την χρήση του *inaccel.allocator* και της random (numpy).
Έπειτα βλέπουμε τον τρόπο με τον οποίο γίνονται όλα τα αιτήματα προς το Coral που αποτελεί τον διαχειριστή των πόρων του FPGA.
Παραδείγματος χάριν για την πρόσθεση και την αφαίρεση έχουμε τα αντίστοιχα αιτήματα:

```
inaccel.request("com.inaccel.math.vector.addition")
inaccel.request("com.inaccel.math.vector.subtraction")
```

Κατά την εκτέλεση παρατηρείται πως η *πρώτη φορά που εκτελείται ένα καινούργιο αίτημα στο fpga είναι πολύ πιο αργή καθώς πρέπει να γίνει ο επαναπροσδιορισμός της δομής του* ώστε να συμφωνεί με το bitstream. Λόγω του *μικρού όγκου δεδομένων* καθώς και λόγω της *απλότητας των πράξεων* σε αυτή την περίπτωση δεν υπάρχει βελτίωση στον χρόνο εκτέλεσης της αφαίρεσης και της πρόσθεσης σε σχέση με αυτόν της CPU. Η καθυστέρηση που σημειώνεται αποδίδεται:

I. Μεταφορά δεδομένων στην DDR μνήμη του FPGA

II. Επιστροφή των δεδομένων στο host-OS

3 Ταξινομητής Naive Bayes

Στο συγκεκριμένο κομμάτι της εργαστηριακής άσκησης έγινε σύγκριση του τροποποιημένου GaussianNBC για το InAccel με την υλοποίηση του sci-kit.
Η βάση της σύγκρισης είναι ο χρόνος της πρόβλεψης (inference) για τον εκάστοτε ταξιμνομητή αλλά και η ακρίβεια που επιτυγχάνεται. Αυτός είναι και ο λόγος που κατα την διάρκεια της δημιουργίας και διαχωρισμού δεδομένων κάνουμε ένα μη συνηθισμένο split οπού τα δεδομένα εκπαίδευσης είναι το 10% και τα δεδομένα ελέγχου είναι το 90%.
Για να διευκολύνουμε την ολοκλήρωση της άσκησης δημιουργήσαμε τις κατάλληλες συναρτήσεις για την δημιουργία δεδομένων με είσοδο τις παραμέτρους (samples, features, classes) καθώς και για την εκπαίδευση του inaccel.sklearn.naive_bayes και του sklearn.naive_bayes. Τα αποτελέσματα των πειραμάτων μας παρατίθενται στον πίνακα 1.

Features	Classes	FPGA-Time	CPU-Time	Accuracy	Speedup
500	10	0.18	5.30	99.09	29.45
500	35	0.20	18.87	99.08	94.35
500	60	0.25	34.51	98.96	138.04
1000	10	0.34	10.23	99.06	30.08
1000	35	0.38	35.97	98.97	94.65
1000	60	0.48	67.02	99.00	139.62
2000	10	0.61	20.54	99.07	33.67
2000	35	0.79	70.34	99.02	89.03
2000	60	0.74	128.07	99.05	173.06

Table 1: Naive Bayes FPGA/CPU speedup for different set of features/classes.

Όπως αναμέναμε, το speedup μεγαλώνει τόσο με την αύξηση των features όσο και με αυτή των classes. Το *μεγαλύτερο speedup επετεύχθη για features=2000 και classes=60* καθώς όσο πιο περίπλοκο είναι ένα πρόβλημα και όσο περισσότερο χρόνο απαιτεί για συμβατική εκτέλεση σε CPU τόσο μεγαλύτερο speedup περιμένουμε. Παρατηρούμε ακόμη από τον παραπάνω πίνακα πως οι προβλέψεις που έγιναν με χρήση του FPGA ταξινομητή και του CPU ταξινομητή παράγουν ακριβώς τα ίδια αποτελέσματα.

Επιπλέον παρατηρούμε πως ενώ *το speedup για ίδιο αριθμό features αυξάνεται σημαντικά καθώς αυξάνονται οι classes, δεν ισχύει το ίδιο για σταθερό αριθμό classes και αυξάνοντας τα features* όπου το speedup μένει παρόμοιο σε κάθε περίπτωση με κάποια μικρή μόνο αύξηση για features=2000. Η διαφοροποίηση αυτή στην κλιμάκωση του speedup γίνεται εμφανής και στα παρακάτω γραφήματα 1, 2:

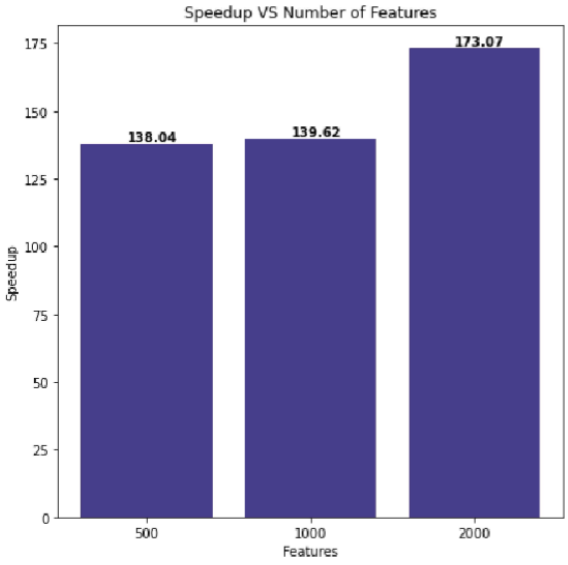

Figure 1: Speedup of FPGA/CPU for different number of features, classes=60.

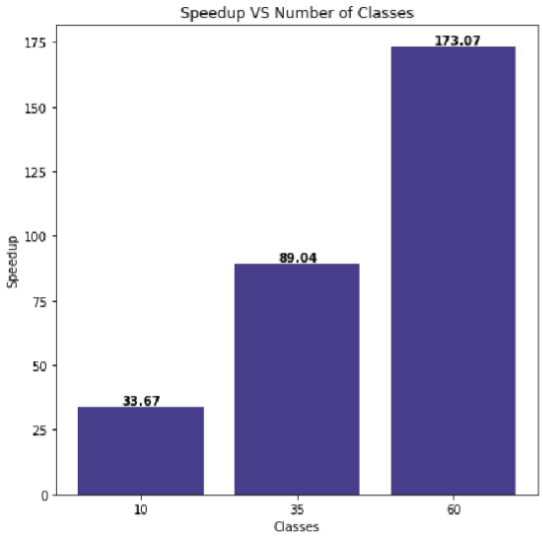

Figure 2: Speedup of FPGA/CPU for different number of classes, features=2000.

Καταληκτικά λοιπόν συμπεραίνουμε πως, αν έπρεπε να διαλέξουμε, *θα προτιμούσαμε να αξιοποιήσουμε αυτόν τον FPGA accelerator σε ένα dataset με περισσότερα classes αντί για ένα με περισσότερα features* καθώς στο πρώτο είναι που θα περιμέναμε να δούμε καλύτερη κλιμάκωση στο speedup.

4 Λογιστική Παλινδρόμηση

Σε αυτό το notebook είδαμε την προσαρμογή ενός Logistic Regression μοντέλου πάνω στο οποίο εκτελέσαμε hyperparameter tuning με την χρήση εξαντλητικής αναζήτησης με πλέγμα (grid-search). Τα δεδομένα που χρησιμοποιούνται σε αυτό το κομμάτι είναι το γνωστό σύνολο δεδομένων Modified MNIST που περιέχει 70.000 χειρόγραφα ψηφία.

Αρχικά, βλέπουμε την δημιουργία του regressor ο οποίος δέχεται μια παράμετρο *n_accel* που καθορίζει τον αριθμό των accelerators που θα χρησιμοποιηθούν. Ύστερα, ξεκινώντας από ένα πολύ μικρό πλέγμα τιμών (που φαίνεται παρακάτω) εκτελούμε ένα GridSearch με cross-validation (3 fold) για να βρούμε τον βέλτιστο συνδυασμό παραμέτρων.

```
param_grid = {'max_iter': (75), 'l1_ratio': (0.5, 0.7)}
```

Η διαδικασία αυτή για το FPGA χρειάζεται **55.59s** και δημιουργεί μοντέλο με απόδοση της τάξης του 0.913% με παραμέτρους

```
Best parameters set:
 l1_ratio: 0.5
 max_iter: 75
```

Αντιστοίχως, εκτέλουμε την διαδικασία προσαρμογής των υπερ-παραμέτρων στο μοντέλο Λογιστικής Παλινδρόμησης σε *CPU*. Σε αυτή την περίπτωση βλέπουμε πάρα πολύ σημαντικές διαφορές καθώς ο χρόνος εκτέλεσης του grid search είναι **24 λεπτά και 11 δευτερόλεπτα**. Φυσικά, τα αποτελέσματα ως προς τις βέλτιστες παραμέτρους συμφωνούν, επομένως τα μοντέλα επιτυγχάνουν ίδια ακρίβεια. Σύμφωνα με τα παραπάνω το *speedup ανέρχεται στο* 26.12.

Τέλος, θέλοντας να υπερτόνισουμε την υπεροχή των FPGA επεκτείναμε το πλέγμα στο παρακάτω και επανεκτελέσαμε την αναζήτηση.

```
param_grid_max = { 'max_iter': (50,75,100),
                   'l1_ratio': (0.3, 0.5, 0.7, 0.9)}
```

Ο χρόνος που χρειάστηκε είναι **4 λεπτά και 20 δευτερόλεπτα** ενώ για την αντίστοιχη αναζήτηση για CPU χρειάστηκαν **65 λεπτά και 30 δευτερόλεπτα**. Το αντίστοιχο accuracy με το βέλτιστο set παραμέτρων είναι 0.916%.

```
Best parameters set:
 l1_ratio: 0.5
 max_iter: 100
```

Παρά το γεγονός ότι δεν έχουμε λάβει πολλαπλές μετρήσεις προκειμένου να έχουμε μια μέση τιμή για τον χρόνο εκτέλεσης σε CPU καθώς και την αντίστοιχη διακύμανση φαίνεται πως ο χρόνος αυτός είναι σημαντικά μεγαλύτερος, επομένως για την *ταχεία εκτέλεση ενός GridSearchCV θα επιλέγαμε την χρήση του FPGA*.