Εθνικο Μετσοβιο Πολυτεχνειο

Επιστημη Δεδομενων και Μηχανικη Μαθηση

Παράλληλες Αρχιτεκτονικές Υπολογισμού για Μηχανική Μάθηση

Εργαστηριακή Άσκηση - FPGAs

Βλάχος Ιωάννης	AM: 03400087	ioannisvlahos@mail.ntua.gr
Ορφανουδάκης Φίλιππος Σκόβελεφ	AM: 03400107	philipposorfanoudakis@mail.ntua.gr
Πανόπουλος Ιωάννης	AM: 03003050	ioannispanop@mail.ntua.gr
Παπαγεωργίου Δημήτριος	AM: 03400088	dimitriospapageorgiou@mail.ntua.gr

Αθήνα, Ιούλιος 2021

Περιεχόμενα

1. Running the first FPGA accelerated application

Στο πρώτο κομμάτι της εργαστηριακής άσκησης μας ζητείται να τρέξουμε δυο βασικές πράξεις, την **πρόσθεση** και την **αφαίρεση**, πάνω σε δύο vectors μεγέθους 1024×1024 τόσο στη CPU όσο και στην υποδομή FPGA που μας παρέχεται μέσω του InAccel Studio.

Αρχικά, παρατηρούμε ότι έχουμε το ίδιο αποτέλεσμα και στις 2 εκτελέσεις των πράξεων. Στη συνέχεια, το ζητούμενο της άσκησης είναι να σημειώσουμε το χρόνο που χρειάστηκε το αίτημα της πρόσθεσης και της αφαίρεσης ώστε να εξυπηρετηθεί στη CPU και στο FPGA.

Όπως αναφέρεται και στην εκφώνηση, δεν λαμβάνουμε υπ'όψιν την πρώτη εκτέλεση του κώδικα καθώς πραγματοποιείται η επαναδιαμόρφωση του FPGA με το συγκεκριμένο bitstream. Συνεπώς τα αποτελέσματα που λάβαμε για την δεύτερη εκτέλεση, είναι τα εξής :

Second Execution - Time in ns			
	FPGA	CPU	**SpeedUp**
Addition	26575699	3439598	0.1294
Subtraction	14602421	3891210	0.2665

Στο σημείο αυτό να αναφέρουμε πως ο τύπος που χρησιμοποιούμε για το SpeedUp είναι ο εξής :

$$S = \frac{T_{cpu}}{T_{FPGA}}$$

Επομένως, βλέπουμε ότι δεν έχουμε κάποια επιτάχυνση, πόσο μάλλον η CPU φέρνει πολύ καλύτερους χρόνους. Για να αιτιολογήσουμε αυτό το φαινόμενο θα παρατηρήσουμε το Monitor το οποίο μας δείχνει τον χρόνο που απαιτείται στο Read, Execution και Write κάθε αιτήματος που στέλνουμε στο FPGA.

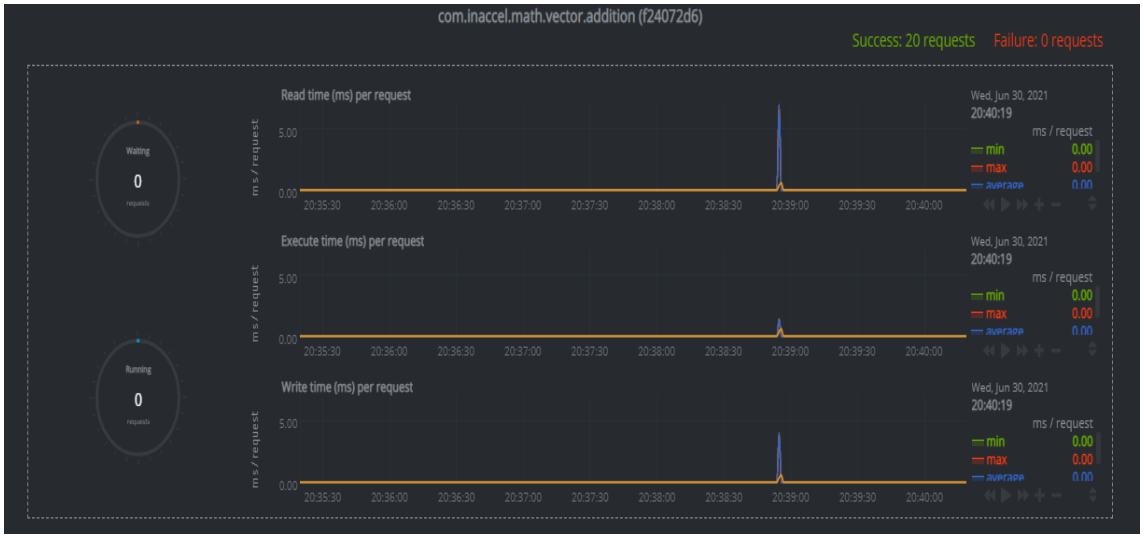

Figure 1.1: Addition

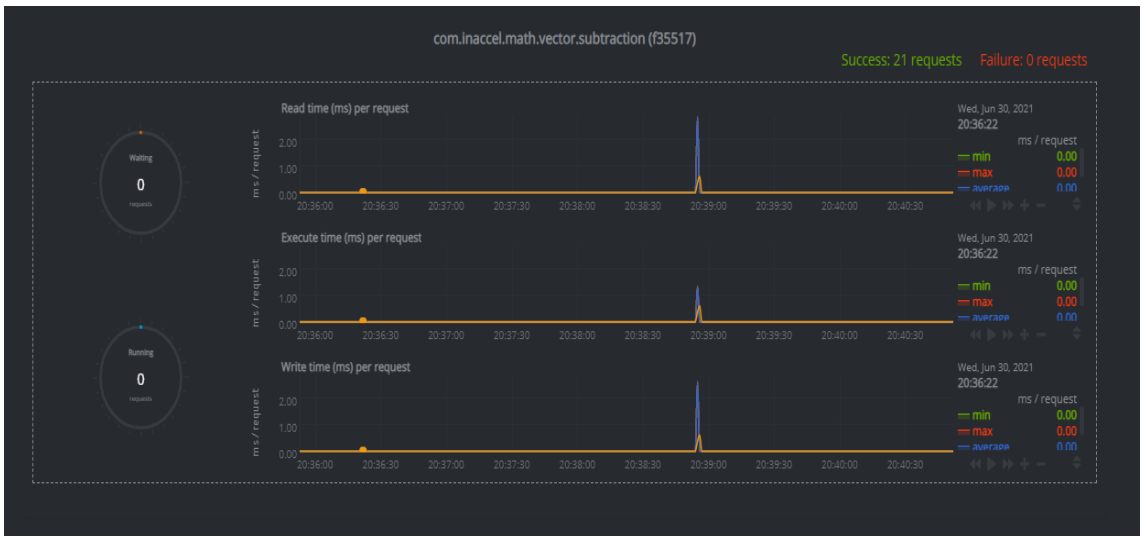

Figure 1.2: Subtraction

Είναι ξεκάθαρο ότι ο περιοριστικός παράγοντας είναι ο χρόνος του Read και Write, δηλαδή ο χρόνος που απαιτείται για την μεταφορά των δεδομένων από την CPU στην DRAM του FPGA και το αντίστροφο. Τέλος το μέγεθος των vectors είναι αρκετά μικρό, έτσι ώστε να μην έχει μεγάλες υπολογιστικές απαιτήσεις, για αυτό και η CPU επιτυγχάνει καλύτερο χρόνο.

2. Scikit-Learn on FPGAs

2.1 Naive Bayes Example

Στο δεύτερο κομμάτι της εργαστηριακής άσκησης, μας ζητείται να χρησιμοποιήσουμε την βιβλιοθήκη sklearn, για να συγκρίνουμε τις εκτελέσεις ενός Naive Bayes Classifier που προορίζεται για την CPU και ενός Naive Bayes Classifier που προορίζεται για FPGA. Ο δεύτερος έχει βελτιστοποιηθεί στο κομμάτι του inference, δηλαδή της πρόβλεψης.

Θα δοκιμάσουμε τον αλγόριθμο του Naive Bayes για διαφορετικής μορφής Dataset τη φορά. Οι δοκιμές που θα γίνουν θα είναι οι εξής :

- samples: 100000
- features: 500, 1000, 2000
- classes: 10, 35, 60

Συνολικά έχουμε 9 πειράματα , και ακόμα ένα αρχικό με samples: 10000, features: 500, classes: 10 (10 στο σύνολο). Αρχικά να αναφέρουμε πως για όλες τις δοκιμές που πραγματοποιήσαμε, παρατηρήσαμε την ίδια ακρίβεια στη προβλεπτική ικανότητα τόσο στον αλγόριθμο εκπαιδευμένο στη CPU όσο και στο FPGA. Παρακάτω παραθέτουμε τα αποτελέσματα :

Samples	**Features**	**Classes**	**FPGA Time (sec)**	**CPU Time (sec)**	**SpeedUp**	**Accuracy**
10000	500	10	0.03	0.55	18.33	99.13%
100000	500	10	0.22	5.25	23.86	99.09%
100000	500	35	0.24	26.33	109.7	99.08%
100000	500	60	0.23	36.43	158.39	98.96%
100000	1000	15	0.35	14.57	41.62	99.06%
100000	1000	35	0.41	34.34	83.75	98.97%
100000	1000	60	0.39	65.88	**168.92**	99.0%
100000	2000	10	0.72	19.73	27.4	99.07%
100000	2000	35	0.7	173.21	104.58	99.02%
100000	2000	60	0.73	99.05	165.47	99.05%

Όπως παρατηρούμε και από τον πίνακα το μεγαλύτερο SpeedUp το λαμβάνουμε για :

- samples: 100000
- features: 1000
- classes: 60

Διαισθητικά αναμέναμε το μεγαλύτερο SpeedUp να εμφανίζεται όταν το πρόβλημα μας απαιτεί περισσότερους υπολογισμούς. Αυτό συμβαίνει για $classes$: 60 και $features$: 2000, καθώς πρέπει να υπολογιστούν οι πιθανότητες $P(feature_i|class_k)$, δηλαδή $features \times classes$ υπολογισμοί. Στη συνέχεια από $\#classes$ πολλαπλασιασμούς με $\#features$ όρους θέλουμε να βρεθεί το μέγιστο με χρονική πολυπλοκότητα $O(\#classes)$.

Παρόλα αυτά παρατηρούμε μια στασιμότητα στο SpeedUp καθώς αυξάνουμε τα features. Όταν αυξάνουμε τα features προσθέτουμε έξτρα στήλες στα δεδομένα μας, δηλαδή περισσότερο όγκο. Καθώς η CPU δεν μοιράζεται την ίδια μνήμη με το FPGA, πρέπει να γίνει και μεταφορά περισσότερων δεδομένων, δηλαδή έχουμε αύξηση στον Read χρόνο. Στην περίπτωση των 2000 features έχουμε διπλασιασμό του όγκου συγκριτικά με τα 1000 features. Παραθέτουμε τους αντίστοιχους χρόνους που σημειώθηκαν σε ένα kernel, στην περίπτωση των $classes$: 60 και $features$: 2000.

Figure 2.1: Naive Bayes: 60 classes & 2000 features

Βλέπουμε ότι ο Read χρόνος είναι μεγαλύτερος από τον Execution Time. Μια υπόθεση μας είναι ότι με την αύξηση των όρων που απαιτούνται στον πολλαπλασιασμό για τον υπολογισμό κάθε $P(feature_i|class_k)$ μειώνουμε το operational intensity, δηλαδή για την ίδια πράξη θέλουμε περισσότερα δεδομένα. Αυτό έχει ως αποτέλεσμα να απαιτείται μεγαλύτερη μεταφορά δεδομένων από τον δίαυλο CPU → FPGA.

Συνεπώς υποθέτουμε ότι με συνεχόμενη μείωση του operational intensity η αρχιτεκτονική μπορεί να περιορίζεται από τις δυνατότητες του διαύλου, δηλαδή ο δίαυλος της δεν μπορεί να υποστηρίξει βέλτιστα την μεταφορά των δεδομένων.

Για να μπορέσουμε να απαντήσουμε με σιγουριά θα έπρεπε να γνωρίζουμε το optimization που έχει γίνει στον επιταχυντή για τον συγκεκριμένο αλγόριθμο του Naive Bayes. Με αυτό τον τρόπο θα καθορίζαμε το operational intensity κάθε εκτέλεσης και θα μπορούσαμε να τοποθετηθούμε πάνω στο roofline μοντέλο της συγκεκριμένης αρχιτεκτονικής. Για τον καθορισμό του roofline μοντέλου χρειαζόμαστε το bandwidth του διαύλου CPU→FPGA (το οποίο δεν μας είναι γνωστό), όπως επίσης το kernel speed, το οποίο μας είναι γνωστό καθώς μας δίνονται οι επιταχυντές που χρησιμοποιούνται (2 Xilinx Alveo [U250 and U280] και 2 Intel PAC A10 FPGA).

Για καλύτερη εποπτεία του παραπάνω φαινομένου σχεδιάζουμε 2 διαγράμματα. Το πρώτο δείχνει το SpeedUp με την αύξηση των classes και το δεύτερο το SpeedUp με την αύξηση των features.

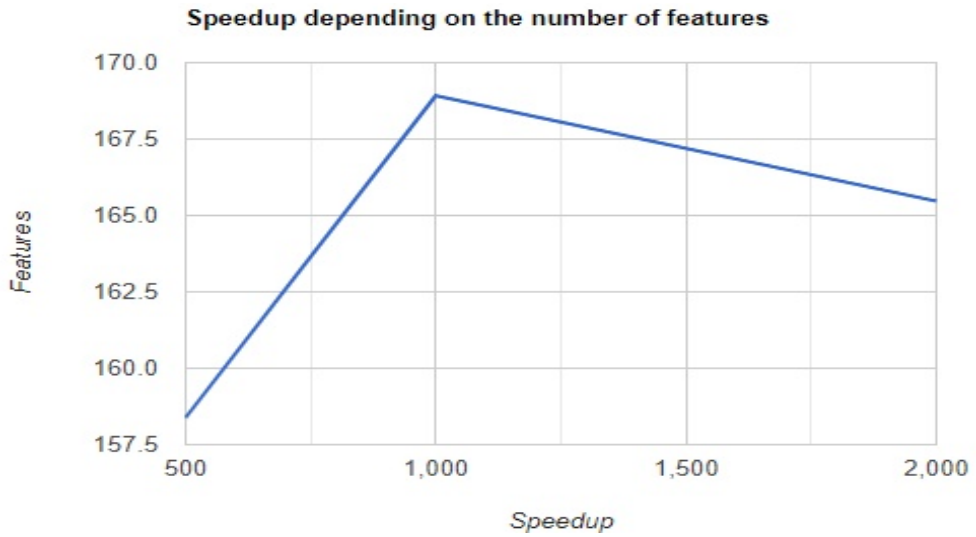

Figure 2.2: Features vs SpeedUp

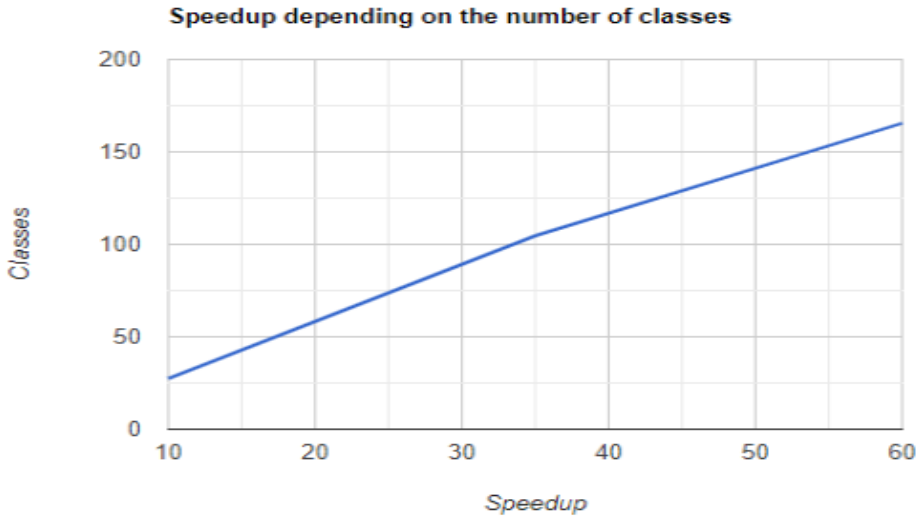

Figure 2.3: Classes vs SpeedUp

Συνεπώς αν έπρεπε να επιλέξουμε να χρησιμοποιήσουμε τον συγκεκριμένο FPGA accelerator στον συγκεκριμένο αλγόριθμο θα προτιμήσουμε δεδομένα που η πολυπλοκότητα τους οφείλεται στις περισσότερες κλάσεις και όχι στα περισσότερα features.

2.2 Logistic Regression Example

Στο τελευταίο κομμάτι της εργαστηριακής άσκησης, μας ζητείται να εφαρμόσουμε hyperparameter tuning στον αλγόριθμο Logistic Regression, και να μελετήσουμε τους χρόνους εκτέλεσης τόσο σε CPU όσο και σε FPGA. Συνεπώς, ενώ προηγουμένως εξετάζαμε την επιτάχυνση στο prediction part, τώρα εστιάζουμε στο training part.

Το hyperparamater tuning θα πραγματοποιηθεί με GridSearchCV, δηλαδή εφαρμογή της τεχνικής CrossValidation πάνω σε όλους του συνδυασμούς ενός grid. Θα δοκιμάσουμε 2 grid τα οποία είναι τα εξής :

Grid #1 - 2 combinations

- max iter: (75)

- l1 ratio: (0.5,0.7)

Grid #2 - 12 combinations

- max iter: (75, 50, 100)
- l1 ratio: (0.3,0.5, 0.7,0.9)

Τα αποτελέσματα που λάβαμε είναι τα εξής:

FPGA				
	Time (sec)	**Best Score**	**max iter**	**l1 ratio**
Grid #1	25.83	0.91561	75	0.5
Grid #2	62.49	0.91687	100	0.5

CPU				
	Time (sec)	**Best Score**	**max iter**	**l1 ratio**
Grid #1	1354.23	0.91300	75	0.5
Grid #2	4326.85	0.91643	50	0.3

Και τα αντίστοιχα SpeedUp :

- Grid #1: 52.42
- Grid #2: 69.24

Αρχικά, παρατηρούμε ότι συγκριτικά, το FPGA με την CPU πετυχαίνουν παρόμοιο Best Score σε κάθε Grid, αν και η τελική επιλογή των υπερπαραμέτρων στο Grid #2 είναι διαφορετική. Τέλος, βλέπουμε αύξηση στο SpeedUp, γεγονός που μας ενθαρρύνει να χρησιμοποιήσουμε τον συγκεκριμένο FPGA επιταχυντή με αυτόν τον αλγόριθμο και σε αυτό το ζητούμενο. Ο λόγος είναι ότι δεν παρατηρείται κάποια σταθεροποίηση με την αύξηση του μεγέθους του grid όπως είδαμε στο προηγούμενο κομμάτι της άσκησης για τα features.