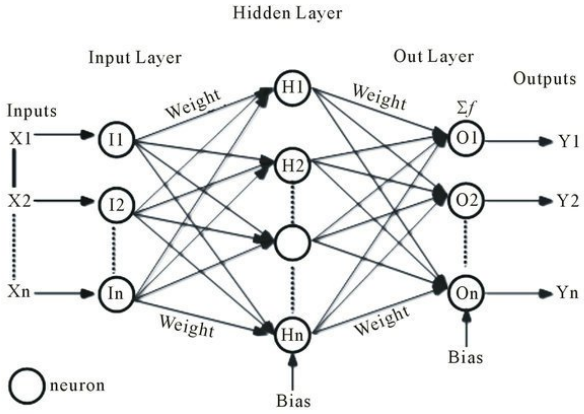
Παράλληλα και διανεμημένα συστήματα
Εργασία 4
Μηχανική Μάθηση
Εκπαίδευση νευρωνικού δικτύου με τον αλγόριθμο Backpropagation και παραλληλοποίηση με Pthread και Cuda

Μπεκιάρης Θεοφάνης ΑΕΜ 8200

Νευρωνικά δίκτυα



Ένα νευρωνικό δίκτυο όπως αυτό της παραπάνω εικόνας αποτελείται κυρίως από :

- Τα διανύσματα εισόδου ή δεδομένων.
- Τα διανύσματα εξόδου που αποτελούν την επιθυμητή έξοδο του νευρωνικού.
- Τους νευρώνες.
- Τα επίπεδα. Υπάρχει ένα επίπεδο εισόδου, ένα επίπεδο εξόδου και τα ενδιάμεσα ονομάζονται κρυφά επίπεδα. Στο παραπάνω σχήμα έχουμε 3 συνολικά επίπεδα.
- Τα διανύσματα συναπτικών βαρών για κάθε νευρώνα ή τους πίνακες βαρών για κάθε επίπεδο.
- Την συνάρτηση ενεργοποίησης (σιγμοειδής συνάρτηση για την εργασία).

<u>Συνοπτική περιγραφή</u>

Οι νευρώνες είναι τα δομικά στοιχεία του δικτύου. Κάθε τέτοιος κόμβος δέχεται ένα σύνολο αριθμητικών εισόδων από διαφορετικές πηγές (είτε από άλλους νευρώνες, είτε από το περιβάλλον) και επιτελεί έναν υπολογισμό με βάση αυτές τις εισόδους και παράγει μία έξοδο. Η εν λόγω έξοδος είτε κατευθύνεται στο περιβάλλον, είτε τροφοδοτείται ως είσοδος σε άλλους νευρώνες του δικτύου. Οι νευρώνες εξόδου διοχετεύουν στο περιβάλλον τις τελικές αριθμητικές εξόδους του δικτύου. Οι νευρώνες πολλαπλασιάζουν κάθε είσοδό τους με το αντίστοιχο συναπτικό βάρος και υπολογίζουν το ολικό άθροισμα των γινομένων. Το άθροισμα οδηγείται στην συνάρτηση ενεργοποίησης,που ανάλογα την με τιμή του αθροίσματος παράγει την έξοδο του νευρώνα. Η έξοδος δείχνει κατά πόσο ο νευρώνας ενεργοποιείται ή καθίσταται ανενεργός από την επίδραση των εισόδους του. Συνάρτηση ενεργοποίησης μπορεί να είναι βηματική , γραμμική ,στοχαστική, μη γραμμική(σιγμοειδής που χρησιμοποιείται και στην εργασία).

Αλγόριθμος Backpropagation

Ένα νευρωνικό δίκτυο πρέπει να έχει την ικανότητα για την παραγωγή συγκεκριμένων τιμών "προτύπων" στην έξοδο του ανάλογα με τις τιμές της εισόδους του. Δηλαδή πρέπει να έχει την ικανότητα να "αντιλαμβάνεται" τις εισόδους και να "αντιδρά" αναλόγως σε αυτές. Το κύριο χαρακτηριστικό των νευρωνικών δικτύων είναι η ικανότητα μάθησης. Για να χρησιμοποιηθεί το δίκτυο θα πρέπει πρώτα να περάσει από μία επαναλαμβανόμενη διαδικασία κατά την οποία προσαρμόζει τα συναπτικά βάρη του έτσι ώστε να παράγει τις αναμενόμενες εξόδους σύμφωνα με τις αντίστοιχες τιμές εισόδου. Αλγόριθμος backpropagation είναι ένας αλγόριθμος για την ανανέωση των βαρών.

Για την συγγραφή του κώδικα της εργασίας στηρίχθηκα στις παρακάτω σημειώσεις.

Backpropagation Algorithm

1. Initialization of weight matrices, $\mathbf{W}^{[1]}(1)$, $\mathbf{W}^{[2]}(1)$, $\mathbf{W}^{[3]}(1)$ at time instant $t = 1$ with small random numbers.

2. For $it = 1, 2, 3, \ldots, N_{it}$

   2.0 Initialize $\left(\frac{\partial J(it)}{\partial w_{ij}^{[k]}}\right) \leftarrow 0, \quad k = \overline{1,3}, \quad j = \overline{1,n_k}, \quad i = \overline{1,n_{k+1}}, \quad J(it) \leftarrow 0$

   2.1 For $n = 1, 2, \ldots, N_{set}$ (we omit writing the time argument (t))

   2.1.0 Read a new element $(\underline{u}^{[1]}(t), d(t))$

   2.1.1 Forward computations "**FORWARD PATH**"

$$\underline{v}^{[1]} = \mathbf{W}^{[1]}\underline{u}^{[1]}, \quad \underline{u}^{[2]} = h(\underline{v}^{[1]})$$
$$\underline{v}^{[2]} = \mathbf{W}^{[2]}\underline{u}^{[2]}, \quad \underline{u}^{[3]} = h(\underline{v}^{[2]})$$
$$\underline{v}^{[3]} = \mathbf{W}^{[3]}\underline{u}^{[3]}, \quad \underline{u}^{[4]} = h(\underline{v}^{[3]})$$
$$\underline{e} = \underline{d}(t) - \underline{u}^{[4]}, \quad J_t = \underline{e}^T\underline{e} = \sum_{j=1}^{n_4} e_j^2$$
$$J(it) \leftarrow J(it) + J_t$$

   2.1.2 Backward computation "**BACKWARD PATH**"

   Compute the generalized errors for all neurons, starting with last layer

$$\delta_i^{[3]} = (d_i(t) - u_i^{[4]})h'(v_i^{[3]}) \quad i = \overline{1,n4}$$
$$\delta_i^{[2]} = h'(v_i^{[2]})\sum_{j=1}^{n_4} w_{ji}^{[3]}\delta_i^{[3]} \quad i = \overline{1,n3}$$
$$\delta_i^{[1]} = h'(v_i^{[1]})\sum_{j=1}^{n_3} w_{ji}^{[2]}\delta_i^{[2]} \quad i = \overline{1,n2}$$

   Compute the elements of the gradients, $\frac{\partial J_t}{\partial w_{ij}^{[1]}}$

$$\left(\frac{\partial J_t}{\partial w_{ij}^{[1]}}\right) = -u_j^{[1]}\delta_i^{[1]} \quad i = \overline{1,n_2}, \quad j = \overline{1,n_1}$$
$$\left(\frac{\partial J_t}{\partial w_{ij}^{[2]}}\right) = -u_j^{[2]}\delta_i^{[2]} \quad i = \overline{1,n_3}, \quad j = \overline{1,n_2}$$
$$\left(\frac{\partial J_t}{\partial w_{ij}^{[3]}}\right) = -u_j^{[3]}\delta_i^{[3]} \quad i = \overline{1,n_4}, \quad j = \overline{1,n_3}$$

   2.1.3 Accumulate the gradient elements $J(it)$

$$\left(\frac{\partial J(it)}{\partial w_{ij}^{[k]}}\right) \leftarrow \left(\frac{\partial J(it)}{\partial w_{ij}^{[k]}}\right) + \left(\frac{\partial J_t}{\partial w_{ij}^{[k]}}\right), \quad k = \overline{1,3}, \quad j = \overline{1,n_k}, \quad i = \overline{1,n_{k+1}}$$

   2.2 If $J(it) < \varepsilon$ **STOP**

   2.3 Modify the wheight values in the direction opposed to gradient vector

$$w_{ij}^{[1]}(it+1) = w_{ij}^{[1]}(it) - \lambda\left(\frac{\partial J}{\partial w_{ij}^{[1]}}\right)_{it}$$
$$w_{ij}^{[2]}(it+1) = w_{ij}^{[2]}(it) - \lambda\left(\frac{\partial J}{\partial w_{ij}^{[2]}}\right)_{it}$$
$$w_{ij}^{[3]}(it+1) = w_{ij}^{[3]}(it) - \lambda\left(\frac{\partial J}{\partial w_{ij}^{[3]}}\right)_{it}$$

Πηγή: Μάθημα 8ου εξαμήνου Ψηφιακά Φίλτρα διάλεξη 11 neural-networks

Περιεχόμενα εργασίας

Μαζί με την αναφορά η εργασία περιλαμβάνει και τους φακέλους με τους κώδικες:

- Της γραμμικής υλοποίησης σε γλώσσα C νευρωνικού δικτύου με χρήση του αλγόριθμου backpropagation για την διαδικασία της εκμάθησης του δικτύου. Η συγγραφή του έγινε με την χρήση των παραπάνω σημειώσεων.
- Παράλληλη υλοποίηση του παραπάνω κώδικα με χρήση Pthread και Cuda.

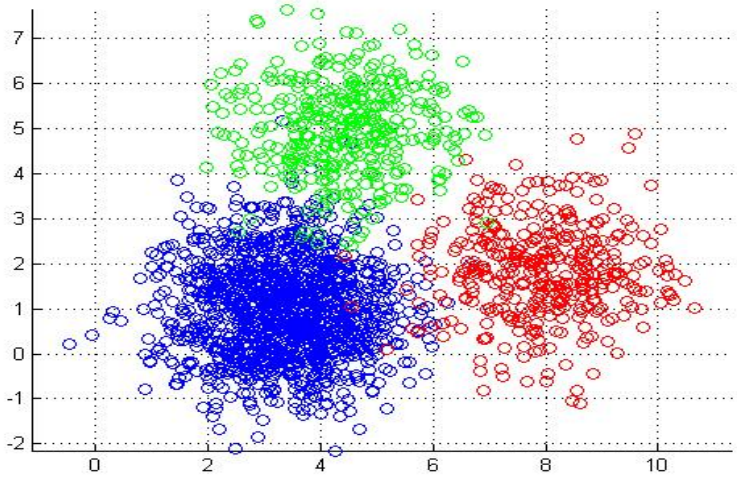
Το νευρωνικό δίκτυο που αναπαριστούν οι παραπάνω υλοποίησεις είναι δίκτυο τεσσάρων επιπέδων, ένα επίπεδο εισόδου ένα επίπεδο εξόδου και δύο κρυφά επίπεδα,για το οποίο γίνεται προσπάθεια παραλληλοποίησης ως προς το μήκος τις εισόδου και το μέγεθος των δεδομένων όπως θα περιγράψουμε στην συνέχεια.

Δομή Κώδικα

Κατά την εκτέλεση του προγράμματος (συνάρτηση main) καλείται η συνάρτηση void *prodData() η οποία παράγει τα δεδομένα εισόδου και τα αντίστοιχα επιθυμητά δεδομένα εξόδου του νευρωνικού. Λεπτομέρειες για την μορφή των δεδομένων θα δοθούν αργότερα. Στην συνέχεια χρησιμοποιούμε τα ΜΙΣΆ από αυτά τα δεδομένα για την εκπαίδευση του δικτύου. Η εκπαίδευση γίνεται με την κλίση της συνάρτησης void* backpropagation_train() στην οποία έχει υλοποιηθεί ο αλγόριθμος backpropagation.Η συνάρτηση υπολογίζει τους πίνακες βαρών (W1,W2,W3) τους οποίους τους επιστρέφει μέσω χρήση pointer. Αφού γίνει η εκπαίδευση του δικτύου καλούμε την συνάρτηση void backpropagation_classify() η οποία χρησιμοποιεί τους παραπάνω πίνακες βαρών και ΌΛΑ τα δεδομένα(τις εισόδους τους μόνο) που δημιουργήσαμε για να υπολογίσει τις εξόδους του δικτύου. Τέλος συγκρίνουμε αυτές τις εξόδους με τις πραγματικές εξόδους του δικτύου και υπολογίζουμε το ποσοστό επιτυχίας εκπαίδευσης του δικτύου.

Μορφή δεδομένων

Για την παραγωγή δεδομένων έχω δημιουργήσει την συνάρτηση void *prodData() η οποία παράγει δεδομένα σε μορφή διανυσμάτων. Τα δεδομένα αναπαριστούν ομάδες ν-διάστατων σημείων γύρο από κάποιο τιμή-πυρήνα διαφορετική για κάθε ομάδα. Για παράδειγμα για δισδιάστατα σημεία τριών ομάδων η τοποθέτησή τους στο επίπεδο θα είναι τις μορφής:



Η συνάρτηση εκτός από συντεταγμένες των σημείων παράγει και τις εξόδους του δικτύου που είναι και αυτές σε μορφή διανυσμάτων. Για κάθε ομάδα δεδομένων παράγεται μία συγκεκριμένη μορφή εξόδου που χαρακτηρίζει την ομάδα. Τα δεδομένα επιστρέφονται από την συνάρτηση ανακατεμένα,δηλαδή είναι τοποθετημένα με τυχαία σειρά στο πίνακα που επιστρέφεται. Το μέγεθος

των δεδομένων στο πρόγραμμα καθορίζεται από τις μεταβλητές που περιέχονται στο αρχείο utils.h και είναι :

- VectLength : Διάσταση διανύσματος δεδομένων.
- VectNumber : Αριθμός δεδομένων ανά ομάδα δεδομένων
- groupsNum : Ομάδες ανεξάρτητων δεδομένων
- outNetLength : Μέγεθος εξόδου

Για παράδειγμα αν:
VectLength : 4 διάσταση διανύσματος εισόδου
VectNumber : 4 δεδομένα ανά ομάδα
groupsNum : 3 διαφορετικές ομάδες
outNetLength : 4 διάσταση διανύσματος εξόδου
ενεργοποιώντας το τελευταίο κομμάτι της main που είναι απενεργοποιημένο σε μορφή σχόλιου θα εκτυπωθούν τα παρακάτω δεδομένα όπου φαίνεται: Πάνω η έξοδος για κάθε ομάδα(άσσοι στην ίδια θέση αντιστοιχούν σε ίδια ομάδα) ,στην μέση οι τιμές των αντίστοιχων διανυσμάτων εισόδου και τέλος κάτω οι αντίστοιχες τιμές που παράγονται από το δίκτυο. Προφανώς στην παρακάτω εικόνα είχαμε 100% επιτυχία γι' αυτό και όλες οι έξοδοι είναι ίδιες με τις πραγματικές εξόδους.

```
-->Elegxoume thn epityxeia ekpaideyshs
Swsta = 12  Sunolo = 12
Epityxeia se pososto 100.00 ths ekato

Pragmatikes times e3odou dedomenwn
0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 1.000000 1.000000 0.000000 0.000000 1.000000
0.000000 0.000000 0.000000 1.000000 1.000000 0.000000 1.000000 0.000000 0.000000 1.000000 0.000000 0.000000
1.000000 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000

Times twn dedomenwn-Dianysmatwn(Eisodoi diktuoy)
-2.938000 -4.000000 -4.000000 2.062000 1.708000 6.062000 1.000000 5.354000 5.708000 2.062000 -4.000000 5.708000
-3.292000 -3.292000 -3.292000 2.062000 1.000000 5.708000 1.000000 5.354000 6.062000 2.062000 -3.292000 5.354000
-3.646000 -3.646000 -4.000000 1.354000 1.354000 5.708000 2.062000 5.354000 6.062000 1.354000 -2.938000 5.708000
-3.646000 -4.000000 -4.000000 1.000000 2.062000 5.000000 1.000000 5.708000 5.000000 1.708000 -3.292000 5.708000

Times e3odou pou prokuptoun apo to diktyo(E3odoi diktuoy)
0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000 1.000000 1.000000 0.000000 0.000000 1.000000
0.000000 0.000000 0.000000 1.000000 1.000000 0.000000 1.000000 0.000000 0.000000 1.000000 0.000000 0.000000
1.000000 1.000000 1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 1.000000 0.000000
0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
```

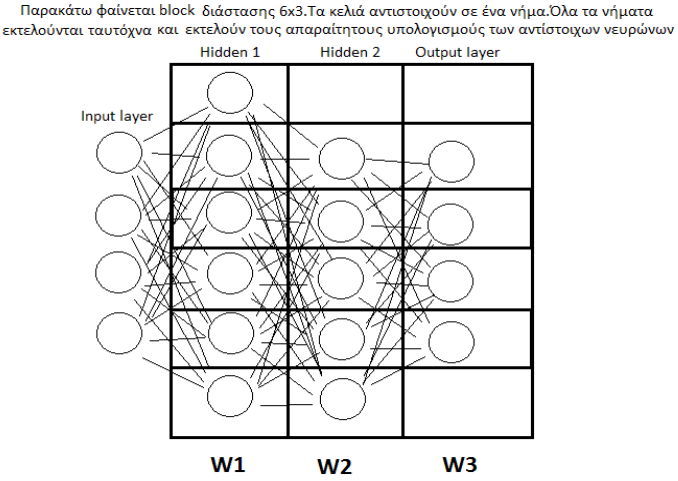
Σύμφωνα με τα παραπάνω το δίκτυο μας στην ουσία θα χρησιμοποιηθεί για την κατηγοριοποίηση ομάδων δεδομένων.

Παραλληλοποίηση με Cuda και Pthread

Σε επίπεδο του backpropagation κώδικα:

Ο αλγόριθμος backpropagation έχει μορφή η οποία είναι αρκετά γραμμική καθώς κάθε επίπεδο για να υπολογίσει τις εξόδους του θα πρέπει πρώτα να έχουν υπολογιστεί οι έξοδοι των προηγούμενων επιπέδων επειδή αποτελούν τις εισόδους του τρέχον επιπέδου. Η παραλληλοποίηση έγινε σε επίπεδο Layer με την χρήση της Cuda και την αξιοποίηση της GPU για ταυτόχρονη εκτέλεση των υπολογισμών που απαιτούνται σε κάθε νευρώνα. Θεώρησα ότι το δίκτυο το αναλαμβάνει ένα block διάστασης (ΔιάστασηΤουΜεγαλύτρουΕπιπέδου X 3) όπου η κάθε στήλη του block αναλαμβάνει ένα επίπεδο του δικτύου(3 στήλες αφού έχουμε 4 επίπεδα δηλαδή 3 πίνακες βαρών W,το επίπεδο εισόδου δεν συμμετέχει).Το καθένα από τα αντίστοιχα νήματα τις εκάστοτε στήλης αναλαμβάνει να εκτελέσει τους υπολογισμούς για έναν νευρώνα και να παράξει την έξοδό του. Η παραπάνω προσέγγιση έγινε με σκοπό να αξιοποιηθεί η Shared Memory που διατίθεται εσωτερικά των block και να αποφευχθεί η χρήση της Global Memory που είναι αρκετά πιο αργή σε σχέση με την προηγούμενη. Έτσι η υλοποίηση μας γίνεται πολύ πιο αποδοτική αφού ο αλγόριθμος backpropagation απαιτεί συνεχής προσπελάσεις στην μνήμη για ανάγνωση και εγγραφή δεδομένων. Το μειονέκτημα τις παραπάνω προσέγγισης είναι ο περιορισμός του μεγέθους του δικτύου που προκύπτει από το περιορισμένο μέγεθος της Shared μνήμης αφού δεν είναι δυνατόν να αποθηκευτούν πίνακες μεγάλων διαστάσεων. Επιπλέον αν ο αριθμός των αντίστοιχων νευρώνων κάθε επιπέδου διαφέρει σημαντικά τότε θα υπάρχουν νήματα που δεν χρησιμοποιούνται. Τα

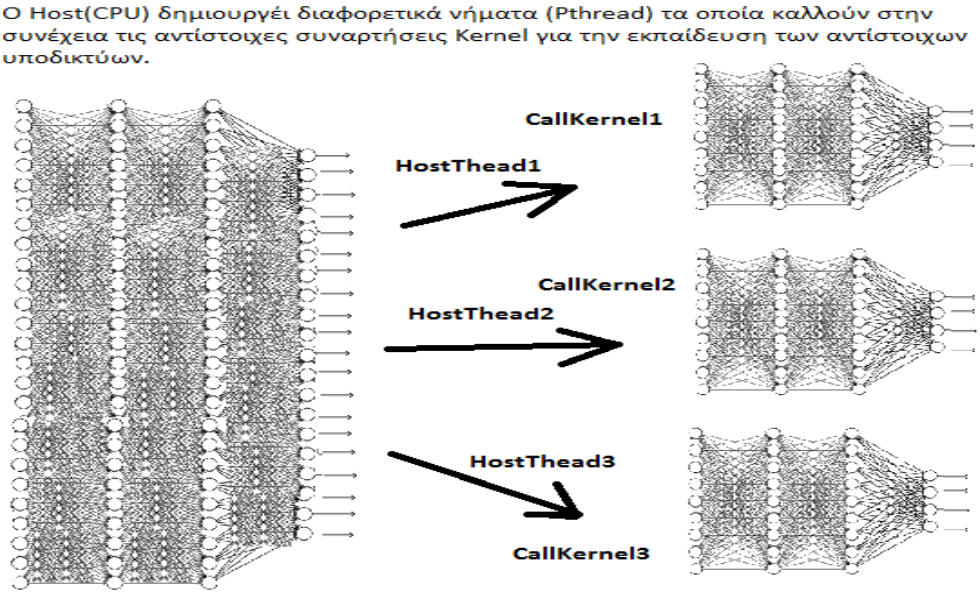
προαναφερθέντα προβλήματα ξεπερνιούνται με τις τεχνικές που θα περιγράψω παρακάτω και τις άλλες μορφές παραλληλοποίησης που χρησιμοποίησα. Τα παραπάνω απεικονίζονται στην παρακάτω εικόνα.



Παραλληλοποίηση σε επίπεδο διάστασης των layer:

Για να ξεπεράσουμε το πρόβλημα των μεγάλων δικτύων(ως προς το μέγεθος των layer και όχι ως προς τον αριθμό τους) και το πρόβλημα της μνήμης Shared ανά block,θεώρησα ότι το δίκτυο χωρίζεται σε κομμάτια και κάθε κομμάτι με την χρήση των **Pthread** ανατίθεται σε ένα νήμα. Τα νήματα του Host τρέχουν παράλληλα,εκτελούν ταυτόχρονα τις αντίστοιχες αρχικοποιήσεις στο πρόγραμμα και καλούν την GPU να εκτελέσει την προηγούμενη υλοποίηση για κάθε υπο-τμήμα ή αλλίως υποδίκτυο του συνολικού νευρωνικού δικτύου. Με την παραπάνω μέθοδο αξιοποιούμε την δυνατότητα που έχει ο Host να καλεί από διαφορετικά streams την GPU για να εκτελέσει ταυτόχρονα περισσότερα από ένα κομμάτια κώδικα(Kernels). Για να συμβεί αυτό,δηλαδή το κάθε νήμα να καλεί την GPU ασύγχρονα θα πρέπει στο αρχείο Makefile που εκτελεί την μεταγλώττιση να προσθέσουμε την δήλωση **--default-stream per-thread**,η οποία δηλώνει κατηγορηματικά ότι θέλουμε κάθε νήμα να χρησιμοποιεί stream ανεξάρτητο από τα υπόλοιπα,άρα οι κλίσεις της GPU να γίνονται ασύγχρονα. Το μειονέκτημα της παραπάνω μεθόδου είναι ότι εν μέρη αλλοιώνουμε την δομή του καθώς το μετατρέπουμε με μερικός συνδεδεμένο νευρωνικό δίκτυο. Δηλαδή χάνονται οι συνδέσεις μεταξύ των υποδικτύων. Παρόλα αυτά η παραπάνω τεχνική είναι πολύ αποδοτική για μεγάλα (σε μέγεθος layer) δίκτυα με μεγάλες εισόδους. Το κατά πόσο μια τέτοια υλοποίηση καθιστά το δίκτυο λειτουργικό εξαρτάται από την εφαρμογή και τα δεδομένα εισόδου. Στην δικιά μας περίπτωση το δίκτυο συνεχίζει να είναι λειτουργικό και έχει την δυνατότητα μάθησης και παραγωγής των σωστών εξόδων ακόμα και σε δεδομένα καινοφανής για το δίκτυο(δηλαδή τα υπόλοιπα δεδομένα που δεν χρησιμοποιήθηκα για εκπαίδευση).

Για τον έλεγχο των μεγεθών του

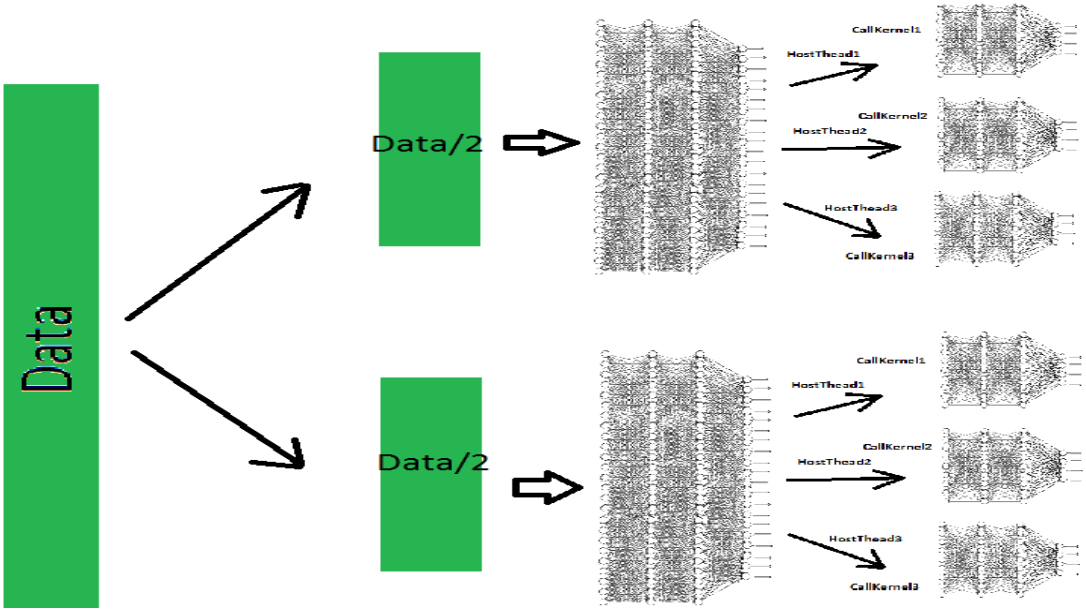


δικτύου,στον φάκελο **cuda** στο αρχείο **utils**.h έχουμε τις μεταβλητές **n1,n2,n3,n4** που καθορίζουν το μέγεθος των layer των υποδικτύων(εισόδου,τα 2 κρυφά και το 1 εξόδου).Με την μεταβλητή **numOfSubNetworks** ορίζουμε πόσα υποδίκτυα έχουμε. Για κάθε υποδίκτυο που ορίζουμε παράγουμε και ένα νήμα Host το οποίο το αναλαμβάνει. Με τις παραπάνω μεταβλητές στην ουσία ορίζουμε έμμεσα το μέγεθος του συνολικού δικτύου αφού το συνολικό δίκτυο είναι τόσο μεγάλο όσα υποδίκτυα ορίσουμε. Με αυτή την υλοποίηση ίσως να μην μπορούμε να ορίσουμε όλους τους δυνατούς συνδυασμούς μεγεθών των στρωμάτων του συνολικού δικτύου,εξάλλου δεν μας ενδιαφέρει η ακριβή υλοποίηση δικτύου αλλά αυτό που θέλουμε είναι να δούμε την απόδοση για τις διάφορες κλίμακες μεγεθών του δικτύου. Για την ακριβή υλοποίηση απλά θα έπρεπε να ορίζουμε περιπτώσεις.

Παραλληλοποίηση ως προς τον αριθμό των δεδομένων εισόδου:

Αυτή η περίπτωση μοιάζει με την προηγούμενη καθώς ακολουθεί την ίδια λογική για την παραγωγή νημάτων Host και ασύγχρονης κλίσης kernel από τα νήματα. Σε αυτή την περίπτωση δημιουργούμε πανομοιότυπα δίκτυα με τα παραπάνω με την λογική των υποδικτύων όπως περιγράψαμε μόνο που τώρα μοιράζουμε και τα δεδομένα στα αντίστοιχο σύνολα υποδικτύων. Η παρακάτω εικόνα περιγράφει καλύτερα την ιδέα.

Ο



αριθμός των υποδικτύων όπως φαίνεται παραπάνω αυξάνεται ανάλογα με τον αριθμό των τμημάτων στα οποία θα χωρίσουμε τα δεδομένα αλλά και τα layer. Για διαχωρισμό των layer σε 3 υποδίκτυα και διαμοιρασμό των δεδομένων στα δύο βλέπουμε ότι απαιτούνται συνολικά 6 νήματα. Ο αριθμός των τμημάτων στα οποία θέλουμε να χωρίσουμε τα δεδομένα διαχειρίζεται από την μεταβλητή **splitData** του αρχείο **utils.h** στον φάκελο **cuda**. Όπως φαίνεται από τα παραπάνω ο αριθμός των συνολικών νημάτων και υποδικτύων είναι ίσος με **splitData* numOfSubNetworks**.

Αποτελέσματα μετρήσεων

Τα παρακάτω αποτελέσματα έχουν προκύψει μετά από εκτέλεση των προγραμμάτων στον σύστημα του Διάδη. Επίσης να υπενθυμίζω ότι η εκπαίδευση του δικτύου γίνεται με τα **μισά** από τα παραγόμενα δεδομένα και στην συνέχεια χρησιμοποιούμε **όλα** τα δεδομένα για να δούμε αν η πίνακες βαρών W1,W2 και W3 που προκύπτουν από την εκπαίδευση είναι σωστή και μπορούν να κάνουν σωστά την κατηγοριοποίηση των δεδομένων. Τέλος,ο έλεγχος των παραμέτρων (μεγέθη δεδομένων,δικτύου και αριθμός νημάτων)γίνεται από το αρχείο utils.h .
Το γραμμικό και το παράλληλο πρόγραμμα τρέχουν για αριθμό εποχών ίσο με 500.

Ως προς την παραλληλοποίηση του backpropagation για εκτέλεση στην GPU.

Αριθμός Δεδομένων	**120**	**600**	**1200**	**12000**	Επιτυχής εκπαίδευση
Διαστάσεις Συνολικού Δικτύου	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Cuda Χρόνος (sec)	0,478	2.186	4.361	43.240	NAI
Γραμμικό Χρόνος (sec)	1,466	7.319	14.620	146.908	NAI

Παρατηρούμε ότι υπάρχει μια αναλογία στην αύξηση του χρόνου των προγραμμάτων με την αύξηση του μεγέθους των δεδομένων(δεκαπλασιασμός δεδομένων προκαλεί δεκαπλασιασμό του χρόνου και στα 2 προγράμματα).Η επιτάχυνση που επιτεύχθηκε είναι ίση με (146.9/43.2)= 3.4 πιο γρήγορη εκτέλεση του παραλληλοποιημένου προγράμματος.

Μεταβάλουμε το μέγεθος του δικτύου.

Αριθμός Δεδομένων	**120**	**600**	**1200**	**12000**	Επιτυχής εκπαίδευση
Διαστάσεις Συνολικού Δικτύου	**50x50x50x50**	**50x50x50x50**	**50x50x50x50**	**50x50x50x50**	
Cuda Χρόνος (sec)	0.564	2.646	5.243	52.434	NAI
Γραμμικό Χρόνος (sec)	2.690	15.147	27.524	271.495	NAI

Επίσης και εδώ βλέπουμε αναλογία ανάμεσα στους χρόνους και στον αριθμό των δεδομένων. Για μεγαλύτερο αριθμό δεδομένων δεν γίνονται μετρήσεις αφενός για το μεγάλο χρόνο εκτέλεσης και αφετέρου λόγο του ότι σύμφωνα με την παραπάνω αναλογία τα αποτελέσματα είναι αναμενόμενα.
Βλέπουμε όμως ότι καθώς μεγαλώσαμε το δίκτυο το παράλληλο πρόγραμμα έγινε ακόμα πιο αποδοτικό και η επιτάχυνση έφτασε περίπου στο 5.2

Για την παραλληλοποίηση σε επίπεδο δεδομένων(μεταβάλουμε την μεταβλητή **splitData**). Το γραμμικό απλά εκτελείτε όπως και πριν με παραμέτρους μόνο το μέγεθος του δικτύου και τον αριθμό των δεδομένων.

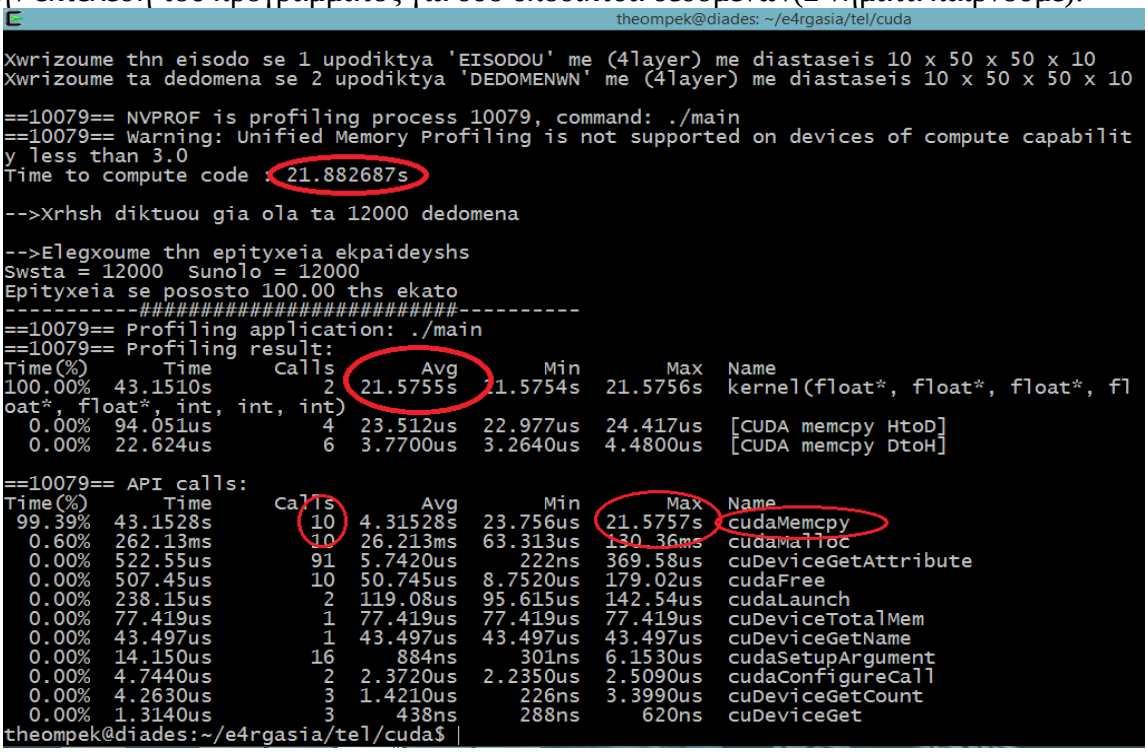
Αριθμός Δεδομένων	**1200**	**1200**	**1200**	**1200**	**1200**	Επιτυχής εκπαίδευση
Διαστάσεις Συνολικού Δικτύου	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Διαστάσεις υποδικτών	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Αριθμός υποδικτύων για διαχωρισμό των layer	1	1	1	1	1	
Αριθμός υποδίκτυων για διαχωρισμό δεδομένων	**1**	**2**	**4**	**8**	**16**	
Συνολικά υποδίκτυα(γινόμενο των 2 παραπάνω)	1	2	4	8	16	
Cuda & Pthread Χρόνος εκτέλεσης	4.361	2.201	3.279	3.813	2.709	NAI
Γραμμικό Χρόνος εκτέλεσης	14.620	14.620	14.620	14.620	14.620	NAI

Αυξάνουμε τον αριθμό των δεδομένων εισόδου και έχουμε:

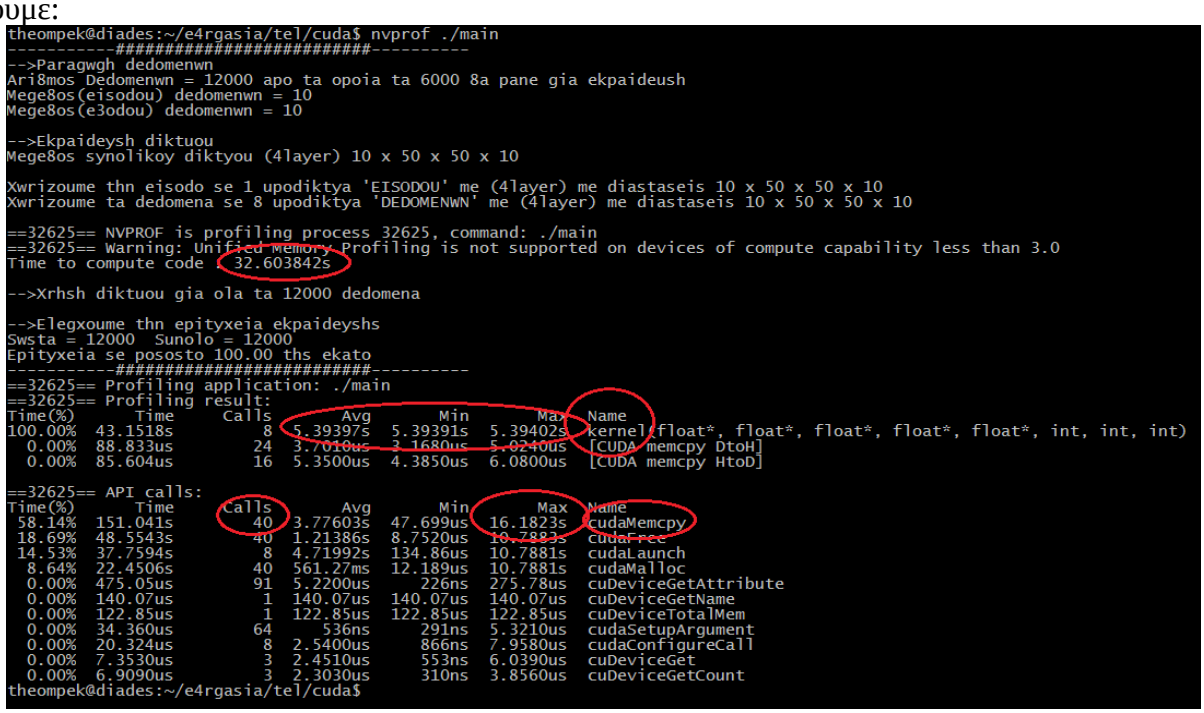
Αριθμός Δεδομένων	**12000**	**12000**	**12000**	**12000**	**12000**	Επιτυχής εκπαίδευση
Διαστάσεις Συνολικού Δικτύου	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Διαστάσεις υποδικτών	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Αριθμός υποδικτύων για διαχωρισμό των layer	1	1	1	1	1	
Αριθμός υποδίκτυων για διαχωρισμό δεδομένων	**1**	**2**	**4**	**8**	**16**	
Συνολικά υποδίκτυα(γινόμενο των 2 παραπάνω)	1	2	4	8	16	
Cuda & Pthread Χρόνος εκτέλεσης	43.240	21.352	32.443	34.855	35.375	NAI
Γραμμικό Χρόνος εκτέλεσης	146.908	146.908	146.908	146.908	146.908	NAI

Από τα παραπάνω δεδομένα παρατηρούμε ότι με την χρήση της παραλληλοποίησης των δεδομένων όπως περιέγραψα προηγουμένος στην αναφορά,έχουμε σαφώς βελτίωση στο χρόνο σε σχέση με την απλή υλοποίηση σε cuda(προφανώς και πολύ καλύτερη από το γραμμικό).Βλέπουμε όμως ότι οι χρόνοι έχουν διακυμάνσεις ειδικά για μικρά δεδομένα και ότι ο καλύτερος χρόνος παρατηρείται στην περίπτωση των 2 νημάτων που χωρίζουμε τα δεδομένα στην μέση. Τα παραπάνω αποτελέσματα πιστεύω ότι οφείλονται στο ότι καθώς αυξάνουμε τα νήματα αυξάνονται κατά μεγάλο βαθμό οι κλίσης για ανταλλαγή δεδομένων μεταξύ της CPU(host) και της GPU(device).Αυτό μπορούμε να το δούμε με την χρήση του **nvprof** μου μας δίνει δεδομένα για τους χρόνους και την χρήση της GPU. Στις παρακάτω εικόνες φαίνονται τα σχετικά αποτελέσματα.

Για την εκτέλεση του προγράμματος για δύο υποδίκτυα δεδομένων(2 νήματα παίρνουμε):



Βλέπουμε ότι για 2 νήματα-υποδίκτυα η συνάρτηση kernel καλείται 2 φορές με χρόνο εκτέλεσης περίπου 21.5 sec(όπου έχουμε παράλληλη εκτέλεση των 2 kernel από την GPU) ενώ έχουμε 10 κλίσεις για ανταλλαγή δεδομένων μεταξύ GPU και CPU. Από ότι φαίνεται όμως οι ανταλλαγή των δεδομένων δεν καθυστερεί το πρόγραμμα καθώς ο μέγιστος χρόνος εκτέλεσης ανταλλαγής ήταν ίσος με τον χρόνο εκτέλεσης των Kernel συναρτήσεων. Αν τώρα αυξήσουμε τα νήματα σε 8 έχουμε:



Δηλαδή βλέπουμε ότι μειώνοντας των αριθμό δεδομένων που διαχειρίζεται το κάθε πρόγραμμα kernel μειώνουμε μεν τον αντίστοιχο χρόνο εκτέλεσης,αυξάνουμε όμως πάρα πολύ των αριθμό των κλίσεων για ανταλλαγή δεδομένων(cudaMemcpy) με αποτέλεσμα όπως φαίνεται να αυξάνεται και ο μέγιστος χρόνος ανταλλαγής (λόγο συμφορήσεις από συνεχής ανταλλαγές). Παραπάνω φαίνεται ότι ο μέγιστος χρόνος για ανταλλαγή δεδομένων ήταν 16.18sec όταν ο μέγιστος χρόνος εκτέλεσης των kernel ήταν 5.39sec. Παρόλα αυτά η ανταλλαγή των δεδομένων είναι απαραίτητη και δεν

μπορεί να παραληφθεί. Ίσως μια λύση στο παραπάνω πρόβλημα να είναι η χρήση περισσότερων από μια κάρτας GPU στις οποίες θα γίνεται ο διαμοιρασμός των συναρτήσεων kernel άρα και των δεδομένων.

Για την παραλληλοποίηση ως προς το μέγεθος του δικτύου(μέγεθος layer) μεταβάλουμε την μεταβλητή **numOfSubNetworks**.

Αριθμός Δεδομένων	1200	1200	1200	1200	Επιτυχής εκπαίδευση
Διαστάσεις Συνολικού Δικτύου	**10x50x50x10**	**20x100x100x20**	**40x200x200x40**	**80x400x400x80**	
Διαστάσεις υποδικτών	10x50x50x10	10x50x50x10	10x50x50x10	10x50x50x10	
Αριθμός υποδικτύων για διαχωρισμό των layer	**1**	**2**	**4**	**8**	
Αριθμός υποδίκτυων για διαχωρισμό δεδομένων	1	1	1	1	
Συνολικά υποδίκτυα(γινόμενο των 2 παραπάνω)	1	2	4	8	
Cuda & Pthread Χρόνος εκτέλεσης	4.361	8.764	13.177	15.352	NAI
Γραμμικό Χρόνος εκτέλεσης	14.620	50.046	171.617	658.211 !	NAI

Από τις μετρήσεις πρώτα από όλα βλέπουμε ότι οι χρόνοι του παράλληλου σε σχέση με το γραμμικό είναι πολύ καλύτεροι. Η επιτάχυνση φτάνει ως και 658/15 = 43.8 φορές πιο γρήγορο να είναι το παράλληλο σε σχέση με το γραμμικό. Επίσης βλέπουμε ότι το πλήθος δεδομένων για ανταλλαγή μεταξύ CPU και GPU δημιουργεί καθυστερήσεις αλλά και πάλι η απόδοση είναι με μεγάλη διαφορά καλύτερη από αυτή του γραμμικού. Η διαφορά αυτή οφείλεται στο ότι θεωρούμε το συνολικό δίκτυο ως συνδυασμό πολλών μικρότερων ανεξάρτητων υποδικτύων που εκτελούνται παράλληλα. Στην περίπτωση των 2 νημάτων δεν βλέπουμε καλύτερο χρόνο σε σχέση με την περίπτωση του ενός όπως στην προηγούμενη περίπτωση με τον διαμοιρασμό δεδομένων επειδή αυτή την φορά δεν μειώνουμε τον αριθμό των δεδομένων στην μέση παρά μόνο το μήκος τους σαν διανύσματα.