



ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

**ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ
ΥΠΟΛΟΓΙΣΤΩΝ**

**ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΕΡΓΑΣΤΗΡΙΟ ΕΠΙΒΛΕΠΟΝΤΟΣ**

**Αναζήτηση και Εξόρυξη Πληροφορίας σε Μεγάλες Βάσεις
Αδόμητων Δεδομένων με Μεθόδους Παράλληλης Επεξεργασίας**

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΟΛΥΜΠΙΑΣ ΤΣΑΜΟΥ

ΕΠΙΒΛΕΠΩΝ: ΕΥΑΓΓΕΛΟΣ ΔΕΡΜΑΤΑΣ

ΠΑΤΡΑ - ΜΗΝΑΣ ΕΤΟΣ

Δερματάς Ευάγγελος , Αναπληρωτής Καθηγητής, Τμήμα Μηχανικών Η/Υ & Πληροφορικής

Αντωνικόπουλος Θεόδωρος, Καθηγητής, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας
Υπολογιστών

Εισαγωγή

Ορισμός του Προβλήματος

Εφαρμογή τεχνικών βελτιστοποίησης σε πρόβλημα ταξινόμησης εικόνων κάνοντας χρήση ενός Συνελικτικού Νευρωνικού Δικτύου [\[cs61c\]](#)

Στόχοι της Διπλωματικής Εργασίας

Γίνεται έρευνα σχετικά με τις διαφορετικές αρχιτεκτονικές νευρωνικών δικτύων και τις μεθόδους εκπαίδευσης, ωστόσο, μια άλλη κρίσιμη πτυχή των νευρωνικών δικτύων είναι, δεδομένου ενός εκπαιδευμένου δικτύου, η γρήγορη και ακριβής ταξινόμηση εικόνων. Σε αυτό το project, δίνετε ένα εκπαιδευμένο νευρωνικό δίκτυο που ταξινομεί εικόνες 32x32 RGB σε 10 κατηγορίες. Οι εικόνες ανήκουν στο dataset CIFAR-10. Μας δίνετε ο αλγόριθμος εμπρόσθιας διάδοσης του νευρωνικού δικτύου και τα τελικά βάρη του δικτύου. Σκοπός είναι η βελτίωση της ταχύτητας της εμπρόσθιας διάδοσης ώστε να γίνει ταξινόμηση με πιο γρήγορο ρυθμό.

Χρήσεις και εφαρμογές

[<https://catalog.ngc.nvidia.com/orgs/nvidia/collections/imageclassification>]

Χρήσεις και εφαρμογές

Κατηγοριοποίηση εικόνων σε μεγάλες βάσεις οπτικών δεδομένων

Οι επιχειρήσεις με βάσεις οπτικών δεδομένων μπορεί να συσσωρεύουν μεγάλο όγκο εικόνων από τις οποίες λείπουν ετικέτες ή meta-data. Αν δεν υπάρχει κάποιος αποτελεσματικός τρόπος οργάνωσης αυτών των εικόνων, μπορεί να μην έχουν κάποια χρησιμότητα. Επιπλέον καταλαμβάνουν πολύτιμο αποθηκευτικό χώρο. Αυτοματοποιημένοι αλγόριθμοι ταξινόμησης εικόνων μπορούν να ταξινομήσουν εικόνες χωρίς ετικέτες σε προκαθορισμένες κατηγορίες.

Αντίστοιχο πρόβλημα είναι αυτό της οργάνωσης των εικόνων σε έξυπνες συσκευές όπως τα κινητά τηλέφωνα. Με τεχνικές ταξινόμησης εικόνων οι εικόνες και τα βίντεο οργανώνονται ώστε να είναι πιο εύκολα προσβάσιμα.

Οπτική Αναζήτηση

Η οπτική αναζήτηση ή αναζήτηση βάσει εικόνας έχει γίνει αρκετά δημοφιλής τα τελευταία χρόνια. Πολλές διακεκριμένες μηχανές αναζήτησης παρέχουν ήδη τη δυνατότητα αναζήτησης οπτικού περιεχομένου παρόμοιο με μια παρεχόμενη εικόνα. Αυτό έχει πολλές εφαρμογές στον τομέα του ηλεκτρονικού εμπορίου και της λιανικής πώλησης, όπου οι χρήστες μπορούν να τραβήξουν μια φωτογραφία και να ανεβάσουν μια εικόνα ενός προϊόντος που τους ενδιαφέρει να αγοράσουν.

Ιατρική

Η ιατρική απεικόνιση αφορά τη δημιουργία οπτικών εικόνων των εσωτερικών τμημάτων του σώματος για κλινικούς σκοπούς. Αυτό περιλαμβάνει την παρακολούθηση της υγείας, την ιατρική διάγνωση, τη θεραπεία και την τήρηση οργανωμένων αρχείων. Οι αλγόριθμοι ταξινόμησης εικόνων μπορούν να διαδραματίσουν κρίσιμο ρόλο στην ιατρική απεικόνιση, βοηθώντας τους επαγγελματίες του ιατρικού κλάδου να ανιχνεύσουν την παρουσία ασθενειών και έχοντας συνέπεια στην κλινική διάγνωση.

Θεωρητικό Υπόβαθρο

Κατηγοριοποίηση Εικόνων με χρήση Συνελικτικών Νευρωνικών Δικτύων

Αναγνώριση Εικόνων

Πώς ένας υπολογιστής αναγνωρίζει εικόνες;

Η ταξινόμηση εικόνων περιγράφει ένα πρόβλημα στο οποίο σε ένα υπολογιστή δίνεται μία εικόνα και πρέπει να καταλάβει τι απεικονίζει (από ένα σύνολο πιθανών κατηγοριών)

Σήμερα, τα Συνελικτικά Νευρωνικά Δίκτυα (CNNs) αποτελούν μια πολύ καλή προσέγγιση αυτού το προβλήματος. Γενικά, τα νευρωνικά δίκτυα υποθέτουν πως υπάρχει κάποια συνάρτηση από την είσοδο (π.χ. εικόνες) σε μία έξοδο (π.χ. ένα σύνολο κατηγοριών εικόνων). Ενώ οι κλασσικοί αλγόριθμοι προσπαθούν να κωδικοποιήσουν κάποια πληροφορία του πραγματικού κόσμου στη συνάρτηση τους, τα CNN μαθαίνουν την συνάρτηση δυναμικά από ένα σύνολο ταξινομημένων εικόνων (labelled images)—αυτή η διαδικασία ονομάζεται εκπαίδευση. Μόλις καταλήξει σε μια σταθερή συνάρτηση (δηλαδή σε μια προσέγγιση αυτής), μπορεί να εφαρμόσει τη συνάρτηση σε εικόνες που δεν έχει ξαναδεί.

Τι μπορεί να κάνει ένα νευρωνικό δίκτυο;

Ένα νευρωνικό δίκτυο αποτελείται από πολλαπλά επίπεδα. Κάθε επίπεδο λαμβάνει έναν πολυδιάστατο πίνακα αριθμών ως είσοδο και παράγει έναν άλλο πολυδιάστατο πίνακα αριθμών ως έξοδο (ο οποίος στη συνέχεια γίνεται η είσοδος του επόμενου επιπέδου). Κατά την ταξινόμηση εικόνων, η είσοδος του πρώτου επιπέδου είναι η εικόνα εισόδου (π.χ. $32 \times 32 \times 3$ αριθμοί για εικόνες 32×32 pixel με 3 κανάλια χρώματος), ενώ η έξοδος του τελευταίου επιπέδου αποτελείται ένα σύνολο πιθανοτήτων των διαφόρων κατηγοριών (π.χ., $1 \times 1 \times 10$ αριθμοί αν υπάρχουν 10 κατηγορίες).

Κάθε επίπεδο έχει ένα σύνολο από βάρη που σχετίζονται με αυτό — αυτά τα βάρη είναι που “μαθαίνει” το νευρωνικό όταν του δοθούν δεδομένα εκπαίδευσης. Ανάλογα με το επίπεδο, τα βάρη έχουν διαφορετικές ερμηνείες, αλλά δεν είναι αντικείμενο μελέτης της συγκεκριμένης διπλωματικής, φτάνει να γνωρίζουμε ότι κάθε επίπεδο λαμβάνει μία είσοδο, εκτελεί κάποια διεργασία σε αυτή, που εξαρτάται από τα βάρη και παράγει μια έξοδο. Αυτό το βήμα ονομάζεται εμπρόσθια διάδοση: παίρνουμε μία είσοδο και την προωθούμε στο δίκτυο, παράγοντας το επιθυμητό αποτέλεσμα ως έξοδο. Η εμπρόσθια διάδοση είναι το μόνο που χρειάζεται για την ταξινόμηση εικόνων σε ένα ήδη εκπαιδευμένο CNN.

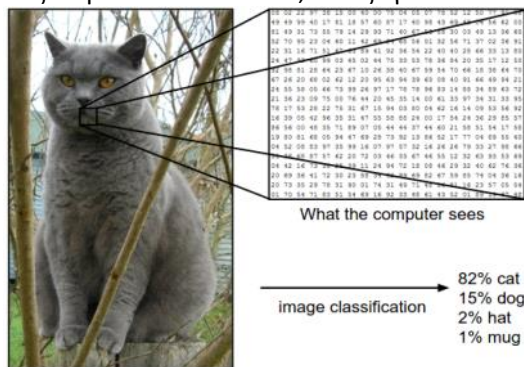
Στην πράξη, ένα νευρωνικό δίκτυο αποτελεί μια πολύ απλή μηχανή αναγνώρισης προτύπων (με εξαιρετικά περιορισμένη χωρητικότητα), αλλά μπορεί να είναι αρκετά παράξενο αυτό που καταλήγει να αναγνωρίσει. Για παράδειγμα, κάποιος μπορεί να εκπαιδεύσει ένα νευρωνικό δίκτυο να αναγνωρίζει τη διαφορά μεταξύ “σκύλων” και “λύκων”, και να δουλέψει καλά κοιτώντας το χιόνι και το δάσος στο φόντο των φωτογραφιών με τους λύκους.

Ταξινόμηση Εικόνων

[<https://cs231n.github.io/classification/>]

Το πρόβλημα της ταξινόμησης εικόνων, αποτελεί ένα από τα βασικά προβλήματα στην Υπολογιστική Όραση και περιλαμβάνει την ανάθεση ετικέτας σε μια εικόνα εισόδου από ένα σταθερό σύνολο κατηγοριών. Η ταξινόμηση εικόνων έχει μεγάλη ποικιλία πρακτικών εφαρμογών.

Για παράδειγμα, στην παρακάτω εικόνα, ένα μοντέλο ταξινόμησης εικόνων λαμβάνει μια εικόνα και αποδίδει πιθανότητες σε 4 ετικέτες, {γάτα, σκύλος, καπέλο, κούπα}. Όπως φαίνεται στην εικόνα, η αναπαράσταση μιας εικόνας σε έναν υπολογιστή γίνεται ως ένας μεγάλος τρισδιάστατος πίνακας αριθμών. Σε αυτό το παράδειγμα, η εικόνα της γάτας έχει πλάτος 248 εικονοστοιχεία, ύψος 400 εικονοστοιχεία και τρία χρωματικά κανάλια Red, Green, Blue (ή RGB για συντομία). Επομένως, η εικόνα αποτελείται από $248 \times 400 \times 3$ αριθμούς ή συνολικά 297.600 αριθμούς. Κάθε αριθμός είναι ένας ακέραιος που κυμαίνεται από 0 (μαύρο) έως 255 (λευκό). Ο στόχος μας είναι να μετατρέψουμε αυτούς τους αριθμούς σε μια ενιαία ετικέτα, όπως "γάτα".



Προκλήσεις

Ενώ η αναγνώριση εικόνας για τον άνθρωπο είναι μια τετριμμένη διαδικασία, από την πλευρά του υπολογιστή προκύπτουν προκλήσεις τις οποίες πρέπει να λάβουμε υπόψιν.

- **Διαφοροποίηση της οπτικής γωνίας.** Η οπτική γωνία του αντικειμένου μπορεί να αλλάζει ανάλογα με τη θέση της κάμερας.
- **Μεταβολή της κλίμακας.** Οι κατηγορίες μπορεί να παρουσιάζουν διαφορές σε σχέση με το μέγεθός του (και στον πραγματικό κόσμο όχι μόνο μέσα στην εικόνα)
- **Παραμόρφωση.** Κάποια αντικείμενα που δεν είναι άκαμπτα μπορεί να εμφανιστούν πολύ παραμορφωμένα
- **Απόκρυψη.** Μερικές φορές τα αντικείμενα ενδιαφέροντος μπορεί να αποκρύπτονται από άλλα και να καταλαμβάνουν μικρό ποσοστό της εικόνας.
- **Συνθήκες φωτισμού.** Ο διαφορετικός φωτισμός μπορεί να επηρεάσει σε μεγάλο βαθμό την εικόνα σε επίπεδο pixel.
- **Ακαταστασία φόντου.** Το αντικείμενο μπορεί να μην είναι ευδιάκριτο σε σχέση με το περιβάλλον στο οποίο βρίσκεται.
- **Διαφοροποίηση εντός της κατηγορίας.** Σε μία κατηγορία μπορεί να υπάρχουν πολλές παραλλαγές του ίδιου αντικειμένου, π.χ. μια καρέκλα.



Ένα καλό μοντέλο ταξινόμησης εικόνας πρέπει να είναι αναλλοίωτο σε όλες τις παραπάνω παραλλαγές, διατηρώντας ταυτόχρονα την ευαισθησία στις παραλλαγές εντός της ίδιας κατηγορίας.

Προσέγγιση με Βάση τα Δεδομένα (Data-driven approach)

Σε αντίθεση με έναν αλγόριθμο που ταξινομεί μια λίστα αριθμών , δεν είναι προφανής ο τρόπος υλοποίησης ενός αλγορίθμου που θα αναγνωρίζει γάτες σε εικόνες. Αντί να προσπαθούμε να προσδιορίσουμε τα χαρακτηριστικά της κάθε κατηγορίας με κώδικα, η προσέγγιση που ακολουθούμε μοιάζει με αυτή που θα ακολουθούσαμε με ένα παιδί. Δίνουμε στον υπολογιστή πολλά παραδείγματα κάθε κατηγορίας και στη συνέχεια αναπτύσσουμε αλγορίθμους μάθησης που εξετάζουν αυτά τα παραδείγματα και μαθαίνουν για την οπτική εμφάνιση κάθε κατηγορίας. Αυτή η προσέγγιση αναφέρεται ως προσέγγιση με βάση τα δεδομένα, καθώς πρώτα συλλέγει δεδομένα από ένα σύνολο κατηγοριοποιημένων δεδομένων.

Νευρωνικά Δίκτυα

...

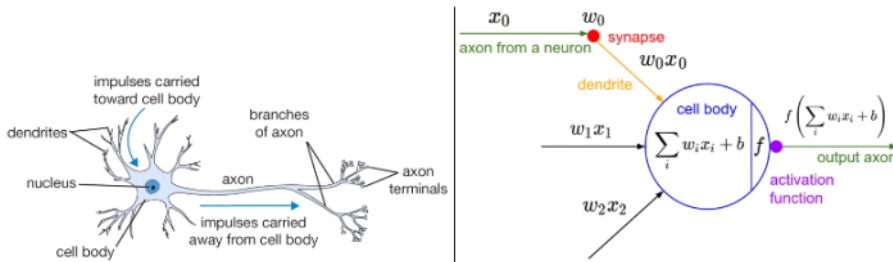
<https://homes.cs.washington.edu/~pedrod/papers/cacm12.pdf>

Νευρωνικά Δίκτυα

<https://cs231n.github.io/neural-networks-1/>

Μοντελοποίηση ενός νευρώνα

Οι νευρώνες είναι εμπνευσμένοι από τους βιολογικούς νευρώνες του ανθρώπινου νευρικού συστήματος. Παρακάτω φαίνεται μια απεικόνιση ενός βιολογικού νευρώνα και η μαθηματική μοντελοποίησή του.



A cartoon drawing of a biological neuron (left) and its mathematical model (right).

Κάθε νευρώνας λαμβάνει σήματα από τους δενδρίτες και παράγει ένα σήμα εξόδου στον άξονα. Στη συνέχεια ο άξονας διακλαδίζεται μέσω συνάψεων σε δενδρίτες άλλων νευρώνων. Στο υπολογιστικό μοντέλο τα σήματα (x_0) στον άξονα, αλληλοεπιδρούν πολλαπλασιαστικά μέσω των συνάψεων με τους δενδρίτες ($w_0 x_0$). Οι συνάψεις θεωρούμε πως είναι τα εκπαιδευσιμα στοιχεία (βάρη w) τα οποία ελέγχουν την επιρροή του ενός νευρώνα σε κάποιον άλλο. Στο κυρίως μέρος του νευρώνα, αθροίζονται τα σήματα από τους δενδρίτες. Εάν αυτό το άθροισμα ξεπερνά ένα συγκεκριμένο κατώφλι, ο νευρώνας ενεργοποιείται και στέλνει σήμα στον άξονα εξόδου. Στο μαθηματικό μοντέλο ο ακριβής χρόνος που παράγονται τα σήματα δεν έχει σημασία, η συχνότητα ενεργοποίησης του νευρώνα μας ενδιαφέρει. Ο τρόπος με τον οποίο το αναπαριστούμε αυτό είναι με μια συνάρτηση ενεργοποίησης. Η πιο συνηθισμένη συνάρτηση είναι η σιγμοειδής, η οποία έχει σαν είσοδο μια πραγματική τιμή (την ισχύ του σήματος μετά το άθροισμα) και το περιορίζει στο διάστημα μεταξύ 0 και 1. Με άλλα λόγια, κάθε νευρώνας εκτελεί το εσωτερικό γινόμενο της εισόδου με τα βάρη προσθέτοντας και μία σταθερά (bias) και εφαρμόζει μία μη γραμμική συνάρτηση, σε αυτή την περίπτωση η σιγμοειδής $\sigma(x) = 1/(1 + e^{-x})$.

Νευρώνες

[πηγή: https://www.eecg.utoronto.ca/~moshovos/CUDA08/arx/convnet_report.pdf]

Ο νευρώνας είναι μια απλή συνάρτηση. Παίρνει μία είσοδο x , υπολογίζει μια συνάρτηση $f(x)$ και τις εξόδους της. Επιλέγουμε τη λογιστική συνάρτηση

$$f(x) = \frac{1}{1 + e^{-x}},$$

Η γραφική παράστασή της φαίνεται στο Σχήμα 1. Είναι η πιο συχνά χρησιμοποιούμενη συνάρτηση. Έχει την ιδιότητα ότι η έξοδός της είναι ουσιαστικά γραμμική ως προς την είσοδο, εάν το μέγεθος της εισόδου είναι μικρό. Αυτό σημαίνει ότι τα νευρωνικά δίκτυα με μικρά βάρη υπολογίζουν μια γραμμική συνάρτηση και η σταδιακή αύξηση των βαρών επιτρέπει τον έλεγχο της μη γραμμικότητας. Ο βαθμός της μη γραμμικότητας ελέγχει την "χωρητικότητα" του νευρωνικού δικτύου. Για παράδειγμα, ένα δίκτυο μόνο με γραμμικούς νευρώνες μπορεί να υπολογίσει μόνο γραμμικές συναρτήσεις

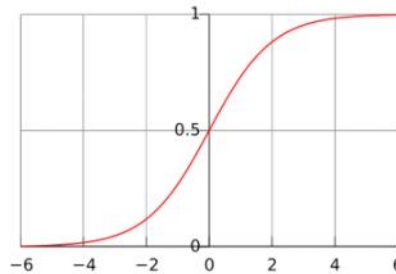


Figure 1: The logistic function $f(x) = \frac{1}{1+e^{-x}}$. [1]

Νευρωνικό Δίκτυο Πρόσθιας Τροφοδότησης (feed forward)

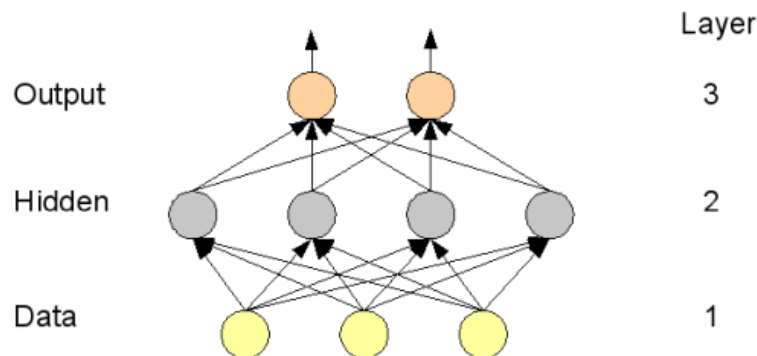


Figure 2: A feed-forward neural network with one hidden layer.

Τα feed-forward νευρωνικά δίκτυα συνδέουν μεταξύ τους τους επίπεδα νευρώνων. Κάθε νευρώνας στο επίπεδο l συνδέεται με κάθε νευρώνα στο επίπεδο $l + 1$, αλλά δεν υπάρχουν συνδέσεις εντός του επιπέδου. Αυτή η αρχιτεκτονική απεικονίζεται στο Σχήμα 2. Σε κάθε σύνδεση υπάρχει ένα βάρος με τιμές πραγματικούς αριθμούς. Ο νευρώνας k στο επίπεδο l δέχεται σαν είσοδο την τιμή:

$$x_k^l = b_k^l + \sum_{i=1}^{N_{l-1}} w_{ik}^{l-1} y_i^{l-1}$$

Όπου

- b_k^l είναι το bias στον νευρώνα k στο επίπεδο l ,
- N_{l-1} είναι ο αριθμός των νευρώνων στο επίπεδο $l - 1$,
- w_{ik}^{l-1} είναι το βάρος μεταξύ του νευρώνα i στο επίπεδο $l - 1$ και του νευρώνα k στο επίπεδο l , και
- y_i^{l-1} είναι η έξοδος του νευρώνα i στο επίπεδο $l - 1$

Στη συνέχεια ο νευρώνα υπολογίζει την έξοδο

$$y_k^l = f(x_k^l)$$

Όπου f είναι οποιαδήποτε διαφορίσιμη συνάρτηση του συνόλου των εισόδων του νευρώνα (εδώ η λογιστική συνάρτηση παραπάνω). Οι νευρώνες στο επίπεδο *data* έχουν σαν έξοδο απλά τα δεδομένα.

Τέλος, η συνάρτηση

$$E(y_1^L, \dots, y_{N_L}^L)$$

της εξόδου του νευρωνικού δικτύου που θα θέλαμε να μεγιστοποιήσουμε (μπορούμε να το θεωρήσουμε σαν ένα επιπλέον επίπεδο στην έξοδο), όπου L είναι ο αριθμός των επιπέδων στο νευρωνικό δίκτυο. Η E πρέπει να είναι διαφορίσιμη ώστε το $\frac{\partial E}{\partial y_k^L}$ να είναι εύκολα υπολογίσιμο.

Training the network consists of clamping the data neurons at the data and updating the parameters

Η εκπαίδευση του νευρωνικού δικτύου περιλαμβάνει clamping τους νευρώνες δεδομένων με τα δεδομένα και ενημερώνοντας τις παραμέτρους (τα βάρη και των biases). Οι παράγωγοι υπολογίζονται με τον παρακάτω :

$$\begin{aligned} \frac{\partial E}{\partial w_{ik}^{l-1}} &= \frac{\partial E}{\partial x_k^l} y_i^{l-1} \\ \frac{\partial E}{\partial b_k^l} &= \frac{\partial E}{\partial x_k^l} \end{aligned}$$

Όπου

$$\begin{aligned} \frac{\partial E}{\partial x_k^l} &= \frac{\partial E}{\partial y_k^l} \frac{\partial y_k^l}{\partial x_k^l} \\ \frac{\partial E}{\partial y_k^l} &= \begin{cases} \frac{\partial E}{\partial y_k^L} & \text{if } l = L \\ \sum_{i=1}^{N_{l+1}} \frac{\partial E}{\partial x_i^{l+1}} w_{ki}^{l+1} & \text{otherwise} \end{cases} \end{aligned}$$

και το $\frac{\partial E}{\partial y_k^L}$ να είναι εύκολα υπολογίσιμο. Από αυτά, οι παράγωγοι ως προς τα βάρη και τα bias μπορούν να υπολογιστούν ξεκινώντας από το τελευταίο επίπεδο. Αυτό είναι γνωστό ως αλγόριθμος οπισθοδρομικής διάδοσης.

Αν αυτό το δίκτυο χρησιμοποιηθεί για κατηγοριοποίηση, ο αριθμός των εξόδων ισούται με τον αριθμό των πιθανών κατηγοριών. Η έξοδος κάθε νευρώνα αντιστοιχεί στην εκτίμηση του δικτύου για κάποια κλάση.

Συνελικτικά Νευρωνικά Δίκτυα

Στα συνηθισμένα νευρωνικά δίκτυα πρόσθιας διάδοσης, κάθε νευρώνας στο επίπεδο l συνδέεται με κάθε νευρώνα του επιπέδου $l - 1$. Αυτό αρκεί αν δεν υπάρχουν τοπικές δομές στην ενεργοποίηση των νευρώνων του επιπέδου $l - 1$. Όμως στην περίπτωση των εικόνων υπάρχει κάποια δομή. Τα διπλανά pixel συσχετίζονται σε μεγαλύτερο βαθμό από αυτά που είναι μακριά. Για αυτό το λόγο κατευθύνουμε το νευρωνικό δίκτυο στην εξαγωγή τοπικών χαρακτηριστικών της εικόνας. Τα συνελικτικά δίκτυα το πετυχαίνουν αυτό συνδέοντας κάθε νευρώνα του κρυφού επιπέδου σε ένα μικρό σε ένα μικρό κομμάτι της εικόνας. Σε αυτή την περίπτωση επιλέγεται ένα κομμάτι 8×8 . Ένα συνελικτικό δίκτυο με έναν νευρώνα θα εφαρμόσει αυτό το φίλτρο 8×8 σε όλες τις 33×33 πιθανές θέσεις μίας 32×32 εικόνας (συμπληρώνοντας μηδενικά πλάτους 4pixel σε κάθε άκρη). Οι έξοδοι των κρυμμένων επιπέδων αποτελούνται από εξόδους 33×33 αυτού του φίλτρου που εφαρμόζονται σε όλη την εικόνα. Σε αυτό το σημείο εμφανίζεται η συνέλιξη.

Το να έχουμε μόνο ένα νευρώνα είναι περιοριστικό, παρόλο που το φίλτρο εφαρμόζεται σε όλη την εικόνα και ο νευρώνας έχει πάνω από χίλιες εξόδους. Θα θέλαμε να έχουμε πολλούς νευρώνες που μαθαίνουν διαφορετικά φίλτρα, καθένα εφαρμόζεται σε μια εικόνα. Αυτό το σενάριο απεικονίζεται στην εικόνα 3. Στην εικόνα υπάρχουν F κρυφοί νευρώνες συνολικά, και κάθε 32×32 πλάκα αναπαριστά τις εξόδους 32×32 που παράγονται από ένα κρυφό νευρώνα όπου κάνει συνέλιξη το φίλτρο 8×8 με την εικόνα και εφαρμόζει την λογιστική συνάρτηση. Υπάρχουν 32×32 έξοδοι

αντί για 33×33 επειδή στις GPU προτιμάμε αριθμούς όπως το 32 παρά το 33, και το νευρωνικό δίκτυο δεν θα χάσει αυτές τις λίγες επιπλέον εξόδους που λείπουν στις άκρες της εικόνας. Σε ένα τυπικό νευρωνικό δίκτυο για να πάρουμε 32×32 εξόδους από το κρυφό επίπεδο, θα πρέπει να έχουμε 32×32 νευρώνες σε αυτό, και αυτό επιτυγχάνεται εδώ με ένα μόνο νευρώνα. Έτσι μπορούμε να θεωρήσουμε τα συνελκτικά νευρωνικά δίκτυα σαν κανονικά νευρωνικά δίκτυα, αλλά με τον περιορισμό ότι ορισμένες ομάδες νευρώνων μοιράζονται τα βάρη (και επίσης οι αυτοί νευρώνες είναι μόνο τοπικά συνδεδεμένες με την εικόνα).

Οι έξοδοι ενός κρυφού νευρώνα όταν εφαρμόζεται συνέλιξη σε κοντινές περιοχές της εικόνας θα είναι παρόμοιες. Επομένως, η τιμές σε κοντινές περιοχές της ίδιας πλάκας 32×32 των εξόδων της κρυφής μονάδας (Σχήμα 3) στο κρυφό στρώμα είναι παρόμοιες. Έτσι, τα συνεπτυγμένα δίκτυα συχνά περιλαμβάνουν ένα στρώμα υποδειγματοληψίας ακριβώς πάνω από το συνεπτυγμένο στρώμα, για να υπολογίζεται τοπικά ο μέσος όρος των αποκρίσεων των κοντινών κρυφών μονάδων. Στο δίκτυό μου, οι μονάδες υπολογισμού του μέσου όρου υπολογίζουν τον μέσο όρο 4×4 μη επικαλυπτόμενων επιφανειών των εξόδων των κρυφών μονάδων. Το στρώμα υπολογισμού του μέσου όρου μειώνει το μέγεθος των πλακών των αποκρίσεων των κρυφών μονάδων από 32×32 σε 8×8 . Μπορεί επίσης να υποστηριχθεί ότι η απώλεια κάποιας ακρίβειας όσον αφορά την ακριβή θέση των χαρακτηριστικών στην εικόνα είναι στην πραγματικότητα πλεονεκτική, επειδή επιτυγχάνεται μεγαλύτερος βαθμός αναλλοίωτης λειτουργίας. Στην όραση αποδεικνύεται συχνά ότι η ακριβής θέση ενός χαρακτηριστικού δεν είναι τόσο σημαντική όσο η κατά προσέγγιση θέση του σε σχέση με άλλα χαρακτηριστικά.

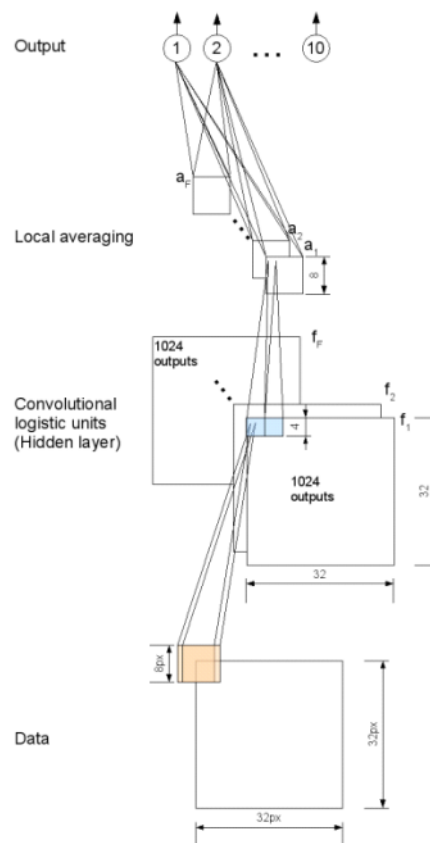


Figure 3: The architecture of the convolutional neural network that I implemented. The convolutional units compute the logistic function: $f(x) = \frac{1}{1+e^{-x}}$. The output layer is a standard logistic regression classifier: $f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$.

Συνελικτικά νευρωνικά δίκτυα (CNNs / ConvNets)

[πηγη: <https://cs231n.github.io/convolutional-networks/>]

Τα Συνελικτικά νευρωνικά δίκτυα μοιάζουν πολύ με τα συνηθισμένα Νευρωνικά Δίκτυα: αποτελούνται από νευρώνες που έχουν εκπαιδευσιμα βάρη και σταθερές (**biases**). Κάθε νευρώνας λαμβάνει κάποιες εισόδους, εκτελεί ένα εσωτερικό γινόμενο και προαιρετικά ακολουθεί μια μη γραμμικότητα. Ολόκληρο το δίκτυο εκφράζει μια ενιαία διαφορίσιμη συνάρτηση βαθμολογίας(score): από τα pixel μιας εικόνας στο ένα άκρο έως τις βαθμολογίες της κλάσης στο άλλο. Ακολουθεί μια συνάρτηση απωλειών (π.χ. SVM/Softmax) στο τελευταίο (πλήρως συνδεδεμένο) επίπεδο.

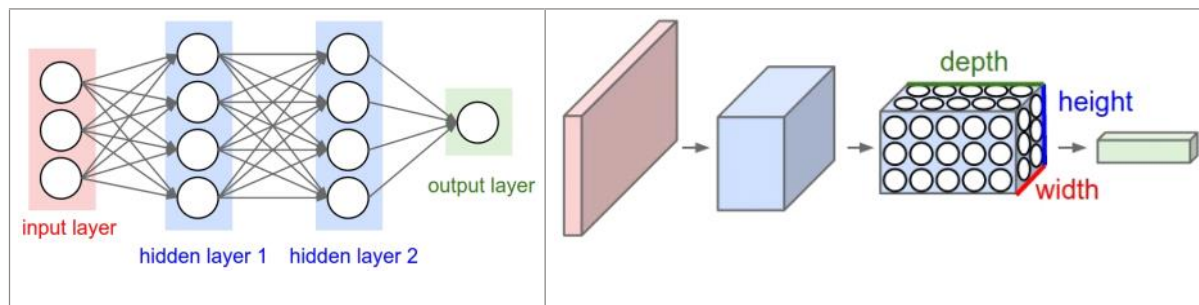
Οι αρχιτεκτονικές ConvNet κάνουν τη ρητή παραδοχή ότι οι είσοδοι είναι εικόνες, γεγονός που μας επιτρέπει να κωδικοποιήσουμε ορισμένες ιδιότητες στην αρχιτεκτονική. Αυτές στη συνέχεια καθιστούν πιο αποτελεσματική την υλοποίηση της συνάρτησης forward και μειώνουν κατά πολύ τον αριθμό των παραμέτρων στο δίκτυο.

Επισκόπηση Αρχιτεκτονικής

Τα απλά Νευρωνικά Δίκτυα λαμβάνουν μια είσοδο (ένα απλό διάνυσμα) και τη μετασχηματίζουν μέσω μιας σειράς κρυφών επιπέδων. Κάθε κρυφό επίπεδο αποτελείται από ένα σύνολο νευρώνων, όπου κάθε νευρώνας συνδέεται πλήρως με όλους τους νευρώνες του προηγούμενου επιπέδου και όπου οι νευρώνες ενός επιπέδου λειτουργούν εντελώς ανεξάρτητα και δεν μοιράζονται καμία σύνδεση. Το τελευταίο πλήρως συνδεδεμένο επίπεδο ονομάζεται "επίπεδο εξόδου" και αντιπροσωπεύει τις βαθμολογίες των κλάσεων.

Τα απλά Νευρωνικά Δίκτυα δεν προσαρμόζονται καλά (don't scale) σε πλήρεις εικόνες(full images). Στο CIFAR-10, οι εικόνες έχουν μέγεθος μόνο $32 \times 32 \times 3$ (32 πλάτος, 32 ύψος, 3 χρωματικά κανάλια), οπότε ένας μόνο πλήρως συνδεδεμένος νευρώνας στο πρώτο κρυφό επίπεδο ενός κανονικού Νευρωνικού Δικτύου θα είχε $32 \times 32 \times 3 = 3072$ βάρη. Αυτό το ποσό φαίνεται ακόμα διαχειρίσιμο, αλλά αυτή η πλήρως διασυνδεδεμένη δομή δεν κλιμακώνεται (does not scale) σε μεγαλύτερες εικόνες. Για παράδειγμα, μια εικόνα πιο μεγάλου μεγέθους, π.χ. $200 \times 200 \times 3$, θα οδηγούσε σε νευρώνες που έχουν $200 \times 200 \times 3 = 120.000$ βάρη. Επιπλέον, θα χρειαζόμασταν πολλούς τέτοιους νευρώνες, οπότε οι παράμετροι θα αυξάνονταν κατά πολύ. Η πλήρης συνδεσιμότητα σε αυτή την περίπτωση κοστίζει σε όγκο και ο μεγάλος αριθμός παραμέτρων μπορεί να οδηγήσει σε overfitting.

Τα Συνελικτικά Νευρωνικά Δίκτυα εκμεταλλεύονται το γεγονός ότι η είσοδος αποτελείται από εικόνες και χρησιμοποιούν την αρχιτεκτονική με τρόπο που έχει νόημα. Πιο συγκεκριμένα, σε αντίθεση με τα απλά Νευρωνικά Δίκτυα, τα επίπεδα του ConvNet έχουν νευρώνες οργανωμένους σε 3 διαστάσεις: πλάτος, ύψος, βάθος. (Με το βάθος αναφερόμαστε στην τρίτη διάσταση του όγκου ενεργοποίησης, όχι το βάθος ολόκληρου του Νευρωνικού Δικτύου, το οποίο αναφέρεται στον συνολικό αριθμό των επιπέδων στο δίκτυο.) Για παράδειγμα, οι εικόνες εισόδου στο CIFAR-10 αποτελεί τον όγκο εισόδου των ενεργοποιήσεων, και ο όγκος έχει διαστάσεις $32 \times 32 \times 3$ (πλάτος, ύψος, βάθος αντίστοιχα).



Επιπλέον: <https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html>

Layers used to build ConvNets

[<https://cs231n.github.io/convolutional-networks/#layers>]

Layers used to build ConvNets

Ένα απλό ConvNet είναι μια ακολουθία επιπέδων και κάθε επίπεδο ενός ConvNet μετατρέπει έναν όγκο ενεργοποιήσεων (activations) σε έναν άλλο μέσω μιας διαφορίσιμης συνάρτησης. Χρησιμοποιούμε τρεις βασικούς τύπους επιπέδων για την κατασκευή αρχιτεκτονικών ConvNet : **Convolutional Layer**, **Pooling Layer**, και **Fully-Connected Layer** (όπως και στα κανονικά νευρωνικά δίκτυα).

Για παράδειγμα. Ένα απλό ConvNet για ταξινόμηση του CIFAR-10, θα μπορούσε να έχει την παρακάτω αρχιτεκτονική [INPUT - CONV - RELU - POOL - FC], όπου:

- INPUT (Είσοδος) [32x32x3] περιέχει τις τιμές των pixel της εικόνας, στην περίπτωση μας μία εικόνα πλάτους 32, ύψους 32 και τρία κανάλια χρωμάτων R,G,B.
- CONV layer (Συνελικτικό επίπεδο) υπολογίζει την έξοδο των νευρώνων που συνδέονται τοπικά σε περιοχές της εικόνας εισόδου, καθένας υπολογίζει το εσωτερικό γινόμενο από τα βάρη και μια μικρή περιοχή που συνδέεται στην είσοδο του όγκου. Αυτό θα έχει σαν αποτέλεσμα έναν όγκο [32x32x12] εάν αποφασίσουμε να χρησιμοποιήσουμε 12 φίλτρα.
- RELU layer (επίπεδο συνάρτησης ReLu) σε κάθε στοιχείο εφαρμόζεται μια συνάρτηση ενεργοποίησης, όπως η $\max(0, x)$ με κατώφλι στο 0. Αυτό το επίπεδο αφήνει το μέγεθος του όγκου το ίδιο ([32x32x12]).
- POOL layer αυτό το επίπεδο εκτελεί μια πράξη υποδειγματοληψίας κατά μήκος των χωρικών διαστάσεων (πλάτος, ύψος), με αποτέλεσμα έναν όγκο [16x16x12].
- FC (i.e. fully-connected) layer (Πλήρως διασυνδεδεμένο επίπεδο) θα υπολογίσει τις βαθμολογίες των κλάσεων, με αποτέλεσμα έναν όγκο μεγέθους [1x1x10], όπου κάθε ένας από τους 10 αριθμούς αντιστοιχεί στη βαθμολογία μιας κλάσης αναμεσα στις 10 κατηγορίες του CIFAR-10. Όπως και στα συνηθισμένα Νευρωνικά Δίκτυα και όπως υποδηλώνει το όνομα, κάθε νευρώνας σε αυτό το επίπεδο συνδέεται με όλους τους αριθμούς του προηγούμενου όγκου.

Με αυτό τον τρόπο, τα ConvNets μετατρέπουν μια εικόνα, επ layer by layer from the original pixel values to the final class scores. Note that some layers contain parameters and other don't. In particular, the CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons). On the other hand, the RELU/POOL layers will implement a fixed function. The parameters in the CONV/FC layers will be trained with gradient descent so that the class scores that the ConvNet computes are consistent with the labels in the training set for each image.

In summary:

- A ConvNet architecture is in the simplest case a list of Layers that transform the image volume into an output volume (e.g. holding the class scores)
- There are a few distinct types of Layers (e.g. CONV/FC/RELU/POOL are by far the most popular)
- Each Layer accepts an input 3D volume and transforms it to an output 3D volume through a differentiable function
- Each Layer may or may not have parameters (e.g. CONV/FC do, RELU/POOL don't)
- Each Layer may or may not have additional hyperparameters (e.g. CONV/FC/POOL do, RELU doesn't)
-

Τί είναι το Accelerated Computing?

? Η επιταχυνόμενη υπολογιστική είναι ένα υπολογιστικό μοντέλο που χρησιμοποιείται σε επιστημονικές εφαρμογές και εφαρμογές μηχανικής, σύμφωνα με το οποίο οι υπολογισμοί εκτελούνται σε ειδικούς επεξεργαστές (γνωστούς ως επιταχυντές) σε συνδυασμό με τις παραδοσιακές CPU για την επίτευξη ταχύτερης εκτέλεσης σε πραγματικό χρόνο. Οι επιταχυντές είναι εξειδικευμένοι μικροεπεξεργαστές που έχουν σχεδιαστεί με γνώμονα τον παραλληλισμό δεδομένων. Πρακτικά, οι χρόνοι εκτέλεσης μειώνονται με την φόρτωση παραλληλοποιήσιμων υπολογιστικά εντατικών τμημάτων μιας εφαρμογής στους επιταχυντές, ενώ ο υπόλοιπος κώδικας συνεχίζει να εκτελείται στην CPU.

ACCELERATED COMPUTING GUIDE	https://www.olcf.ornl.gov/tutorials/accelerated-computing-guide
What Is Accelerated Computing?	https://blogs.nvidia.com/blog/2021/09/01/what-is-accelerated-computing/

→ Η επιταχυνόμενη υπολογιστική χρησιμοποιεί την παράλληλη επεξεργασία για την μειώσει τον χρόνο εκτέλεσης σε απαιτητικές εφαρμογές, από προβλήματα Τεχνητής Νοημοσύνης και ανάλυση δεδομένων μέχρι προσομοιώσεις και απεικόνιση δεδομένων.

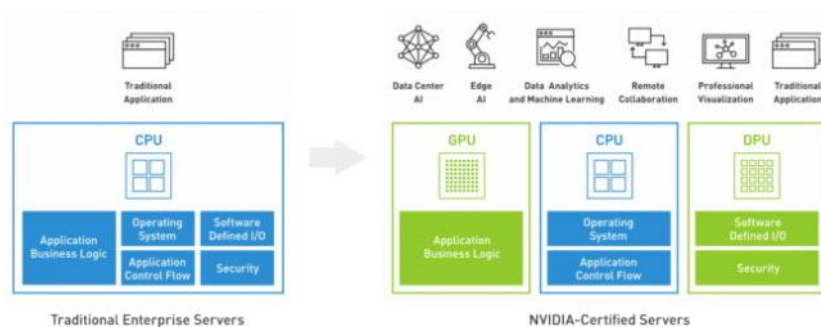
Η επιταχυνόμενη υπολογιστική περιλαμβάνει την χρήση εξειδικευμένου υλικού για να επιταχύνει σημαντικά την εκτέλεση μιας διεργασίας, με παράλληλη επεξεργασία που ομαδοποιεί συχνά εμφανιζόμενες διαδικασίες. Απελευθερώνει από τις απαιτητικές εργασίες που μπορεί να επιβαρύνουν τις CPU, επεξεργαστές που συνήθως εκτελούν τις διαδικασίες σειριακά.

Ξεκινώντας από τους προσωπικούς υπολογιστές, η επιταχυνόμενη υπολογιστική έχει εξελιχθεί στους υπερυπολογιστές. Σήμερα χρησιμοποιείτε στα smartphone και σε κάθε υπηρεσία cloud.

Οι επιταχυνόμενοι υπολογιστές συνδυάζουν CPU και άλλα είδη επεξεργαστών μαζί εξίσου σε μια αρχιτεκτονική που αποκαλείται ετερογενής υπολογιστική.

Accelerated Computers: A Look Under the Hood

Οι GPU είναι οι πιο ευρέως χρησιμοποιούμενοι επιταχυντές. Οι μονάδες επεξεργασίας δεδομένων ([Data processing units - DPUs](#)) είναι μια ταχέως αναδυόμενη κατηγορία που επιτρέπει την ενισχυμένη, επιταχυνόμενη δικτύωση. Κάθε μία έχει να παίξει έναν ρόλο μαζί με την κεντρική CPU για τη δημιουργία ενός ενοποιημένου, ισορροπημένου συστήματος.



Ένας επιταχυνόμενος υπολογιστής προσφέρει χαμηλότερο συνολικό κόστος και υψηλότερη απόδοση και ενεργειακή αποδοτικότητα από ένα σύστημα μόνο με CPU.

Τόσο τα εμπορικά όσο και τα τεχνικά συστήματα εκμεταλλεύονται σήμερα την επιταχυνόμενη υπολογιστική για να διεκπεραιώσουν εργασίες όπως η μηχανική μάθηση, η ανάλυση δεδομένων, προσομοιώσεις και απεικονίσεις. Πρόκειται για ένα σύγχρονο στυλ υπολογιστικής που προσφέρει υψηλές επιδόσεις και ενεργειακή απόδοση.

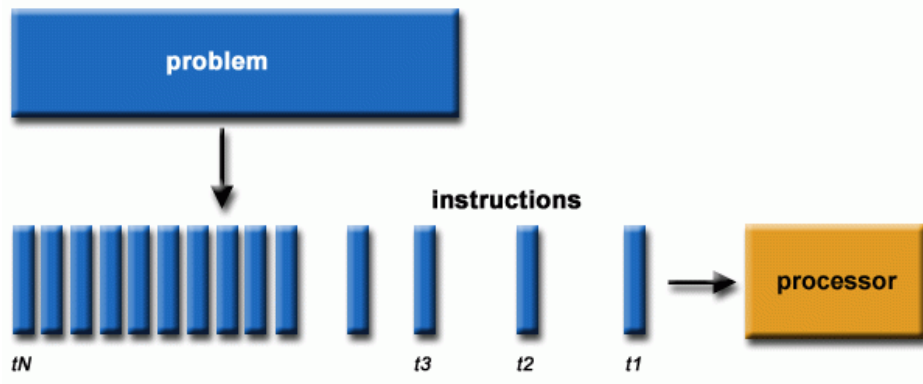
From <https://blogs.nvidia.com/blog/2021/09/01/what-is-accelerated-computing/>

Παράλληλη Υπολογιστική

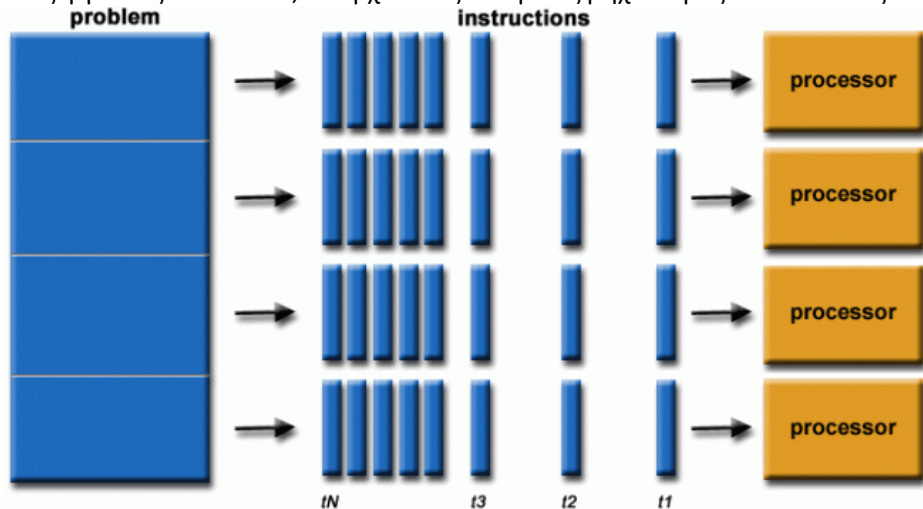
Τι είναι παράλληλος προγραμματισμός

<https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>

Παραδοσιακά, τα προγράμματα γράφονται για σειριακή εκτέλεση και υπολογισμούς. Ένα πρόβλημα αναλύεται σε μία σειρά διακριτών εντολών οι οποίες εκτελούνται διαδοχικά ή μία μετά την άλλη από έναν μόνο επεξεργαστή. Μόνο μία εντολή μπορεί να εκτελεστεί ανά πάσα στιγμή.



Ο παράλληλος προγραμματισμός αναφέρεται στην ταυτόχρονη χρήση πολλαπλών υπολογιστικών πόρων για την επίλυση ενός υπολογιστικού προβλήματος. Το πρόβλημα χωρίζεται σε διακριτά μέρη που μπορούν να επιλυθούν ταυτόχρονα. Κάθε μέρος αναλύεται περαιτέρω σε μια σειρά εντολών, οι οποίες εκτελούνται ταυτόχρονα σε διαφορετικούς επεξεργαστές. Επιπλέον, υπάρχει ένας κεντρικός μηχανισμός που συντονίζει και ελέγχει την εκτέλεση όλων αυτών.



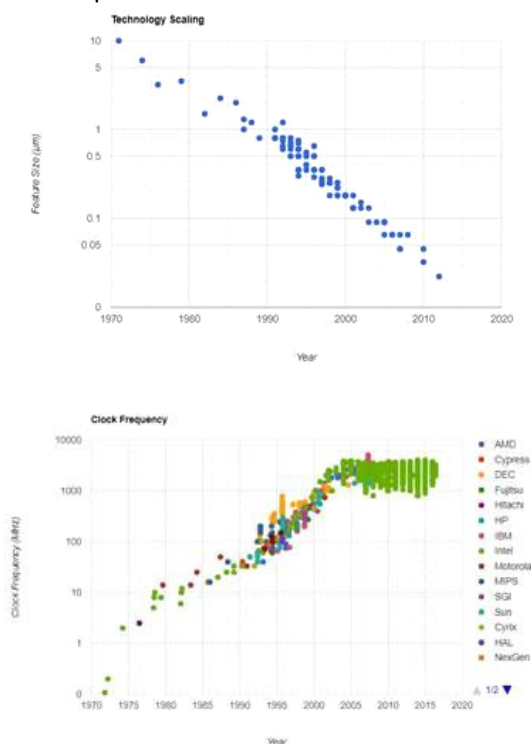
Γιατί Παράλληλος Προγραμματισμός;

Στο σημερινό κόσμο, στον τομέα της επιστήμης και της τεχνολογίας, ο όγκος των δεδομένων που χρειάζεται να επεξεργαστούμε αυξάνεται εκθετικά. Οι απαιτήσεις των εφαρμογών που δίνουν λύση σε πολλά σύγχρονα προβλήματα επιβάλλουν όλο και καλύτερες επιδόσεις στους επεξεργαστές. Ειδικότερα στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης.

Η αύξηση της απόδοσης των σειριακών επεξεργαστών έχει σταματήσει, καθώς οι σχεδιαστές επεξεργαστών έχουν φθάσει στα όρια της σμίκρυνσης, της συχνότητας ρολογιού, της ισχύος, ακόμη και της θερμότητας.

Με την πάροδο του χρόνου, το μέγεθος των τρανζίστορ έχει μειωθεί. Λειτουργούν ταχύτερα, καταναλώνουν λιγότερη ενέργεια και μπορούμε να τοποθετήσουμε περισσότερα σε ένα ολοκληρωμένο κύκλωμα.

Η αύξηση της συχνότητας του ρολογιού σε έναν επεξεργαστή, έχει σταματήσει λόγω της μεγάλης κατανάλωσης ενέργειας και την έκλυση θερμότητας που αυτή συνεπάγεται.



Καθώς η κατανάλωση ισχύος είναι από τους πιο σημαντικούς παράγοντες στον σχεδιασμό σύγχρονων επεξεργαστών, η αύξηση της υπολογιστικής ισχύος, σήμερα, επιτυγχάνεται κυρίως μέσω της προσθήκης όλο και περισσότερων υπολογιστικών πυρήνων στους επεξεργαστές. Αυτή η τάση της αύξησης του αριθμού των πυρήνων αντί της ταχύτητας του ρολογιού φαίνεται να επιτυγχάνει την ιδανικότερη απόδοση από άποψη κατανάλωσης αλλά και επεξεργαστικής ισχύος.

* ☐ Για να μπορέσουμε να αξιοποιήσουμε αυτού του είδους του hardware πρέπει να ξεφύγουμε από τον σειριακό τρόπο σκέψης στον προγραμματισμό και να αρχίσουμε να σκεφτόμαστε παράλληλα.

* ☐

Προγραμματισμός σε GPU

<https://developer.nvidia.com/blog/cuda-refresher-reviewing-the-origins-of-gpu-computing/>

<https://www.heavy.ai/technical-glossary/cpu-vs-gpu>

Programming Massively Parallel Processors A Hands-On Approach by David B. Kirk, Wen-Mei W Hwu

CUDA

CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on graphical processing units (GPUs). With CUDA, developers are able to dramatically speed up computing applications by harnessing the power of GPUs.

In GPU-accelerated applications, the sequential part of the workload runs on the CPU – which is optimized for single-threaded performance – while the compute intensive portion of the application runs on thousands of GPU cores in parallel. When using CUDA, developers program in popular languages such as C, C++, Fortran, Python and MATLAB and express parallelism through extensions in the form of a few basic keywords.

The [CUDA Toolkit](https://developer.nvidia.com/cuda-toolkit) from NVIDIA provides everything you need to develop GPU-accelerated applications. The CUDA Toolkit includes GPU-accelerated libraries, a compiler, development tools and the CUDA runtime.

From <<https://developer.nvidia.com/cuda-zone>>

OpenACC

Overview

The OpenACC Application Programming Interface (API) is a collection of compiler directives and runtime routines that allow software developers to specify loops and regions of code in standard Fortran, C++ and C programs that should be executed in parallel either by offloading to an accelerator such as a GPU or by executing on all the cores of a host CPU. The OpenACC API was designed and is maintained by an industry consortium. See [the OpenACC website](https://docs.nvidia.com/hpc-sdk/compiler/openacc-gs/) for more information about the OpenACC API.

Η OpenACC είναι διεπαφή προγραμματισμού εφαρμογών (API) που περιλαμβάνει ένα σύνολο οδηγιών μεταγλωττιστή και ρουτινών που επιτρέπουν στους προγραμματιστές να καθορίζουν βρόχους και περιοχές κώδικα σε προγράμματα Fortran, C++ και C που θα πρέπει να εκτελούνται παράλληλα είτε με μεταφόρτωση σε έναν επιταχυντή, όπως μια GPU, είτε με εκτέλεση σε όλους τους πυρήνες μιας κεντρικής CPU.

From <<https://docs.nvidia.com/hpc-sdk/compiler/openacc-gs/>>

Περιβάλλον Μεταβλητών - Data Enviroment

[πηγή: OpenACC for Programmers, ch1.3]

Η OpenACC είναι σχεδιασμένη για να διαχειρίζεται περιβάλλοντα όπου οι υπολογιστικές δομές εκτελούνται σε μια συσκευή που έχει διαφορετικό χώρο μνήμης από το κυρίως πρόγραμμα και αυτό απαιτεί μετακίνηση δεδομένων μεταξύ του host και της συσκευής.

[notes]Γιατί παράλληλος προγραμματισμός;

[Source: CHAPTER 1 Why parallel computing?, Robey, R., & Zamora, Y. (2021). Parallel and High Performance Computing. Manning Publications. <https://www.manning.com/books/parallel-and-high-performance-computing>]

Στο σημερινό κόσμο, στον τομέα της επιστήμης και της τεχνολογίας, ο όγκος των δεδομένων που χρειάζεται να επεξεργαστούμε αυξάνεται εκθετικά. Οι απαιτήσεις των εφαρμογών που δίνουν λύση σε πολλά σύγχρονα προβλήματα επιβάλουν όλο και καλύτερες επιδόσεις στους επεξεργαστές. Ειδικότερα στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης.

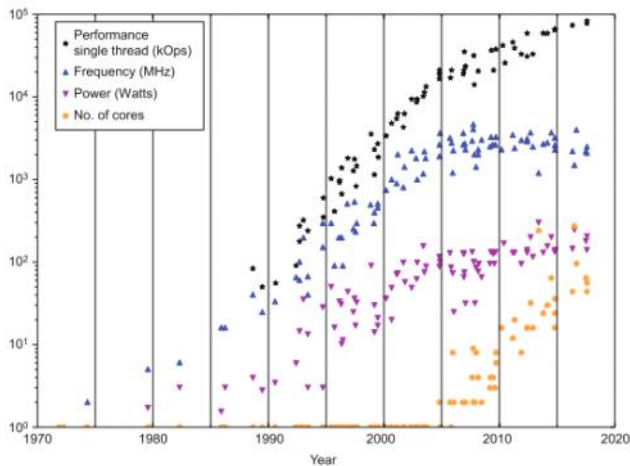
Στο σημερινό κόσμο, στον τομέα της επιστήμης και της τεχνολογίας, ο όγκος των δεδομένων που χρειάζεται να επεξεργαστούμε αυξάνεται εκθετικά. Οι μικροεπεξεργαστές που βασίζονται σε μία μοναδική κεντρική μονάδα επεξεργασίας (CPU)

Στο σημερινό κόσμο, στον τομέα της επιστήμης και της τεχνολογίας, τα νέα προβλήματα που γεννιούνται επιβάλουν όλο και καλύτερες επιδόσεις στους επεξεργαστές. Ειδικότερα στον τομέα της τεχνητής νοημοσύνης και της μηχανικής μάθησης, οι απαιτήσεις σε υπολογιστικούς πόρους είναι όλο και μεγαλύτερες. **Μερικά παραδείγματα εφαρμογών είναι:**

- Μοντελοποίηση μεγάλων πυρκαγιών για να βοηθήσουν τα πυροσβεστικά και τους πολίτες
- Μοντελοποίηση τσουνάμι και ορμητικών κυμάτων από τυφώνες
- Αναγνώριση φωνής σε υπολογιστές
- Μοντελοποίηση εξάπλωσης ιών και ανάπτυξη εμβολίων
- Μοντελοποίηση κλιματικών συνθηκών για δεκαετίες και αιώνες
- Αναγνώριση εικόνας για αυτοκίνητα χωρίς οδηγό
- Εξοπλισμός των σωμάτων έκτακτης ανάγκης με προσομοιώσεις κινδύνων όπως οι πλημμύρες
- Μείωση της κατανάλωσης ενέργειας για κινητές συσκευές

Ένας από τρόπους που οι σχεδιαστές υλικού (*hardware*) κάνουν τους υπολογιστές πιο γρήγορους είναι η αύξηση της συχνότητας του ρολογιού των επεξεργαστών για να μειώσουν το χρόνο σε κάθε βήμα ενός υπολογισμού. Ωστόσο, η αύξηση της ταχύτητας του ρολογιού αυξάνει επίσης την κατανάλωση ενέργειας ενός επεξεργαστή.

Η αύξηση της απόδοσης των σειριακών επεξεργαστών έχει σταματήσει, καθώς οι σχεδιαστές επεξεργαστών έχουν φθάσει στα όρια της **μίκρυνσης**, της συχνότητας ρολογιού, της ισχύος, ακόμη και της θερμότητας. Στο Σχήμα παρουσιάζονται οι τάσεις στη συχνότητα ρολογιού (ο ρυθμός με τον οποίο μπορεί να εκτελεστεί μια εντολή), στην κατανάλωση ενέργειας, στον αριθμό των υπολογιστικών πυρήνων (ή πυρήνων για συντομία) και στις επιδόσεις υλικού με την πάροδο του χρόνου για επεξεργαστές κοινής χρήσης.



Εικόνα. Επιδόσεις ενός thread, συχνότητα ρολογιού CPU (MHz), κατανάλωση ισχύος CPU (watts) και αριθμός πυρήνων CPU από το 1970 έως το 2018. (Horowitz et al. και Rupp, <https://github.com/karlrupp/microprocessor-trend-data>).

Η εποχή του παράλληλου υπολογισμού αρχίζει περίπου το 2005, όταν ο αριθμός των πυρήνων στις CPU αρχίζει να αυξάνεται, ενώ η συχνότητα ρολογιού και η κατανάλωση ισχύος παραμένουν στάσιμα, αλλά η απόδοση αυξάνεται σταθερά

Udacity cs344 : <https://www.udacity.com/course/cs344>

[Όταν μιλάω για ταχύτερο σκάψιμο, στην πραγματικότητα μιλάω για το ότι] ζητάω από τον επεξεργαστή μας να τρέξει με το ταχύτερο ρολόι για να ξοδέψει λιγότερο χρόνο σε κάθε βήμα ενός υπολογισμού. Ωστόσο, σε έναν σύγχρονο επεξεργαστή η αύξηση της ταχύτητας του ρολογιού αυξάνει επίσης την κατανάλωση ενέργειας και είμαστε στο όριο όσον αφορά την κατανάλωση ενέργειας σε ένα τσιπ.

[Όταν μιλάω για την αγορά ενός πιο παραγωγικού φτυαριού, στην πραγματικότητα μιλάω για το]να ζητήσουμε από τον επεξεργαστή μας να κάνει περισσότερη δουλειά σε κάθε βήμα, σε κάθε κύκλο ρολογιού. [Αλλά όπως το σούπερ φτυάρι,] έτσι και ένας μεμονωμένος επεξεργαστής έχει φτάσει σε φθίνουσα απόδοση ως προς το πόση δουλειά μπορεί να κάνει ανά κύκλο. Για να γίνω λίγο πιο τεχνικός, οι πιο εξελιγμένοι επεξεργαστές βρίσκονται σε ένα όριο, όσον αφορά το πόσο παραλληλισμό σε επίπεδο εντολών μπορούν να εξάγουν ανά κύκλο ρολογιού.

[Και όταν μιλάω για την πρόσληψη περισσότερων σκαπανέων, αναφέρομαι]στον παράλληλο υπολογισμό με έναν τρόπο που ελπίζουμε να σας διδάσουμε σε αυτό το μάθημα. Αντί να έχουμε έναν γρήγορο εκσκαφέα με ένα φοβερό φτυάρι, θα έχουμε πολλούς εκσκαφείς με πολλά φτυάρια. Αντί να έχουμε έναν ή λίγους πολύ ισχυρούς επεξεργαστές, θα έχουμε πολλούς, πολλούς ασθενέστερους, λιγότερο ισχυρούς επεξεργαστές.

From <https://www.youtube.com/watch?v=bun_W5B9lRw>

Τρεις τρόποι να αυξήσουμε την ταχύτητα των υπολογιστών:

1. Πιο γρήγορα ρολόγια: Όταν ο επεξεργαστής τρέχει με γρηγορότερο ρολόι ξοδεύει λιγότερο χρόνο σε κάθε βήμα υπολογισμού. Ωστόσο, σε έναν σύγχρονο επεξεργαστή το να αυξήσουμε την ταχύτητα του ρολογιού αυξάνει και την κατανάλωση ενέργειας και είμαστε στο όριο όσον αφορά την κατανάλωση ενέργειας σε ένα τσιπ.

To understand why the GPU is such an interesting processor today, we'll start with technology trends. Why the world has gone parallel. And first, we'll start with some good news. Modern processors are made from transistors. Each year those transistors get smaller and smaller.

[https://www.youtube.com/watch?v=bun_WSB9iRw>](#)

1. Πιο γρήγορα ρολόγια: Όταν ο επεξεργαστής τρέχει με γρηγορότερο ρολόι ξοδεύει λιγότερο χρόνο σε κάθε βήμα υπολογισμού. Ωστόσο, σε έναν σύγχρονο επεξεργαστή το να αυξήσουμε την ταχύτητα του ρολογιού αυξάνει και την κατανάλωση ενέργειας και είμαστε στο όριο όσον αφορά την κατανάλωση ενέργειας σε ένα τσιπ.
2. Περισσότερη δουλειά αν κύκλο ρολογιού:

When I talk about digging faster I'm actually talking about asking our processor should run with the **faster clock** to spend a shorter amount of time on each step of a computation. However, in a modern processor turning up clock speed also increases power consumption and we're the limit as far as power consumption on a chip.

When I talk about buying a more productive shovel, I'm actually talking about asking our processor to **do more work on each step, on each clock cycle**. But like the super shovel, a single processor has also reached diminishing returns on how much work it can do per cycle. To be a little more technical, the most advanced CPUs are at a limit, as far as how much instruction-level parallelism they can extract per clock cycle.

And when I talk about hire more diggers, I'm referring to **parallel computing** in a way that we hope to teach you in this class. Instead of having one fast digger with an awesome shovel, we're going to have many diggers with many shovels. Instead of having one or just a few very powerful processors, we're going to have many, many weaker, less powerful processors.

From https://www.youtube.com/watch?v=bun_WSB9iRw

Ο λόγος που δεν αυξάνουμε την ταχύτητα ρολογιού δεν είναι ότι τα τρανζίστορ έχουν σταματήσει να γίνονται μικρότερα και ταχύτερα. Παρόλο που τα τρανζίστορ συνεχίζουν να γίνονται μικρότερα και ταχύτερα και να καταναλώνουν λιγότερη ενέργεια ανά τρανζίστορ, το πρόβλημα είναι ότι η λειτουργία ενός δισεκατομμυρίου τρανζίστορ παράγει πάρα πολλή θερμότητα και δεν μπορούμε να διατηρήσουμε όλους αυτούς τους επεξεργαστές δροσερούς. **Η ισχύς έχει αναδειχθεί σε πρωταρχικό κινητήριο παράγοντα, ενδεχομένως τον πιο σημαντικό παράγοντα στο σχεδιασμό σύγχρονων επεξεργαστών σε όλες τις κλίμακες, από ένα κινητό τηλέφωνο που έχετε στην τσέπη σας μέχρι τους πολύ μεγάλους υπερυπολογιστές.** Η συνέπεια είναι ότι δεν μπορούμε να συνεχίσουμε να κατασκευάζουμε επεξεργαστές με τον τρόπο που κάναμε πάντα, κάνοντας έναν μεμονωμένο επεξεργαστή όλο και πιο γρήγορο. Όταν το κάνουμε αυτό, καταλήγουμε σε επεξεργαστές που απλά δεν μπορούμε να διατηρήσουμε δροσερούς. Αντ' αυτού, **οι σχεδιαστές επεξεργαστών έχουν στραφεί στην κατασκευή μικρότερων, πιο αποδοτικών επεξεργαστών από άποψη ισχύος και στη συνέχεια ξοδεύουν τους επιπλέον πόρους τους για να κατασκευάσουν μεγαλύτερο αριθμό αποδοτικών επεξεργαστών αντί για ταχύτερους λιγότερο αποδοτικούς επεξεργαστές.**

to understand why the world is such an interesting processor today, we'll start with technology trends. **Why the world has gone parallel. And first, we'll start with some good news.** Modern processors are made from transistors. Each year those transistors get smaller and smaller.

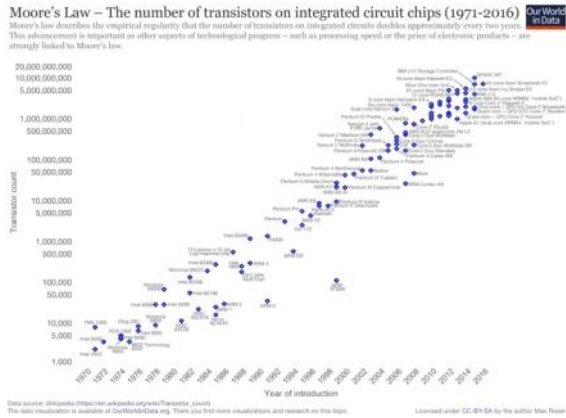
This graph is from Stanford's CPUDB project. **Thanks, guys.** What it shows is the feature size of processors over time, where the feature size is the minimum size of a transistor or wire on a chip. **So what we're seeing is this is time going in this direction. That's a long time ago and that's today. This is the feature size, so how big transistors are.** Notice that it's getting smaller and smaller every generation. **When you hear talk about 45 nanometer process or a 22 nanometer that's what we're referring to, the feature size.** We see that it's consistently going down over time. **As the feature size decreases, transistors get smaller, run faster, use less power, and we can put more of them on a chip. The consequence is that we have more and more resources for computation every single year.**

However, we've also got some bad news. Historically, **as transistors improved, processor designers would then increase the clock rates of processors, running them faster and faster every year.** Let's take a look at this diagram of clock speeds over the years.

So again, we have time going on this axis, okay? So, a long time ago, today. Here we have clock frequency—how fast we're clocking these transistors. Historically, one of the primary drivers of clock performance has been clock speed increases. We see over many years, we see clock speeds continue to go up. However, over the last decade we see that clock speeds have essentially remained constant.

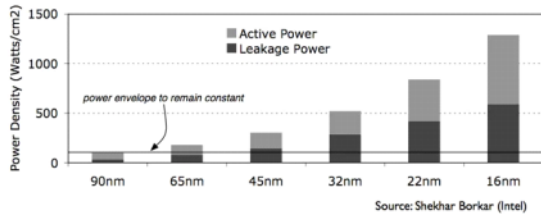
The reason that we're not increasing clock speed is not that the transistors have stopped getting smaller and faster. Even though transistors are continuing to get smaller and faster and consume less energy per transistor, the problem is running a billion transistors generates an awful lot of heat, and we can't keep all these processors cool. Power has emerged as a primary driving factor, possibly the most important factor in modern processor design at all scales, from a mobile phone that you keep in your pocket all the way to the very largest supercomputers. The consequence is we can't keep building processors in the way we always have by making a single processor faster and faster. When we do that, we end up with processors that we simply can't keep cool. Instead, processor designers have turned to building smaller, more efficient processors in terms of power, and they then spend their extra resources to build a larger number of efficient processors rather than faster less-efficient processors.

{other source: <https://www.maketecheasier.com/why-cpu-clock-speed-isnt-increasing/> }



Όπως γνωρίζουμε από το νόμο του Moore, το μέγεθος των τρανζίστορ συρρικνώνεται σε σταθερή βάση. Αυτό σημαίνει ότι περισσότερα τρανζίστορ μπορούν να χωρέσουν σε έναν επεξεργαστή. Συνήθως αυτό σημαίνει μεγαλύτερη επεξεργαστική ισχύ. Υπάρχει επίσης και ένας άλλος παράγοντας, που ονομάζεται κλιμάκωση Dennard. Αυτή η αρχή δηλώνει ότι η ισχύς που απαιτείται για τη λειτουργία των τρανζίστορ σε μια συγκεκριμένη μονάδα όγκου παραμένει σταθερή ακόμη και όταν αυξάνεται ο αριθμός των τρανζίστορ.

Ωστόσο, έχουμε αρχίσει να συναντάμε τα όρια της κλιμάκωσης Dennard και ορισμένοι ανησυχούν ότι ο νόμος του Moore επιβραδύνεται. Τα τρανζίστορ έχουν γίνει τόσο μικρά που η κλιμάκωση Dennard δεν ισχύει πλέον. Τα τρανζίστορ συρρικνώνονται, αλλά η ισχύς που απαιτείται για τη λειτουργία τους αυξάνεται.

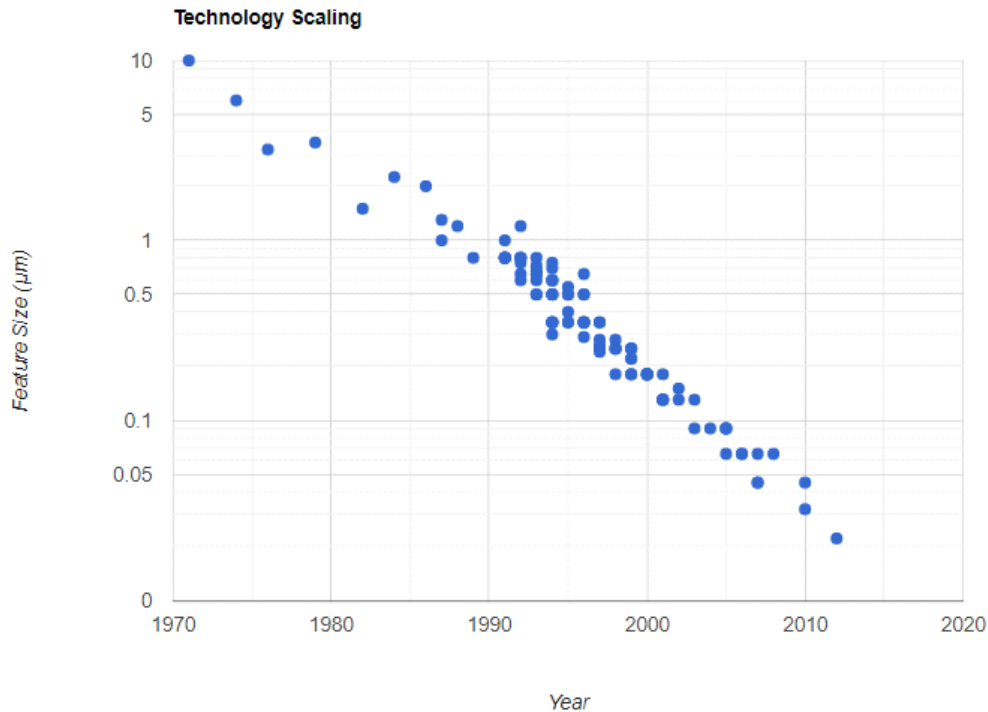


Οι θερμικές απώλειες αποτελούν επίσης σημαντικό παράγοντα στο σχεδιασμό των τσιπ. Η συμπίεση δισεκατομμυρίων τρανζίστορ σε ένα τσιπ και η ενεργοποίηση και απενεργοποίησή τους χιλιάδες φορές ανά δευτερόλεπτο δημιουργεί έναν τόνο θερμότητας. Αυτή η θερμότητα είναι θανατηφόρα για το πυρίτιο υψηλής ακρίβειας και υψηλής ταχύτητας. Αυτή η θερμότητα πρέπει να πάει κάπου, και απαιτούνται κατάλληλες λύσεις ψύξης και σχεδιασμός τσιπ για τη διατήρηση λογικών ταχυτήτων ρολογιού. Όσο περισσότερα τρανζίστορ προστίθενται, τόσο πιο ισχυρό πρέπει να είναι το σύστημα ψύξης για να μπορεί να απορροφήσει την αυξημένη θερμότητα.

Η αύξηση των ταχυτήτων ρολογιού συνεπάγεται επίσης αύξηση της τάσης, η οποία οδηγεί σε κυβική αύξηση της κατανάλωσης ενέργειας για το τσιπ. Έτσι, καθώς αυξάνονται οι ταχύτητες ρολογιού, παράγεται περισσότερη θερμότητα, γεγονός που απαιτεί ισχυρότερες λύσεις ψύξης. Η λειτουργία αυτών των τρανζίστορ και η αύξηση των ταχυτήτων ρολογιού απαιτεί μεγαλύτερη τάση, γεγονός που οδηγεί σε δραματικά μεγαλύτερη κατανάλωση ισχύος. Έτσι, καθώς προσπαθούμε να αυξήσουμε την ταχύτητα ρολογιού, διαπιστώνουμε ότι η θερμότητα και η κατανάλωση ισχύος αυξάνονται δραματικά. Στο τέλος, οι απαιτήσεις ισχύος και η παραγωγή θερμότητας ξεπερνούν την αύξηση της ταχύτητας ρολογιού.

[notes]Udacity Videos

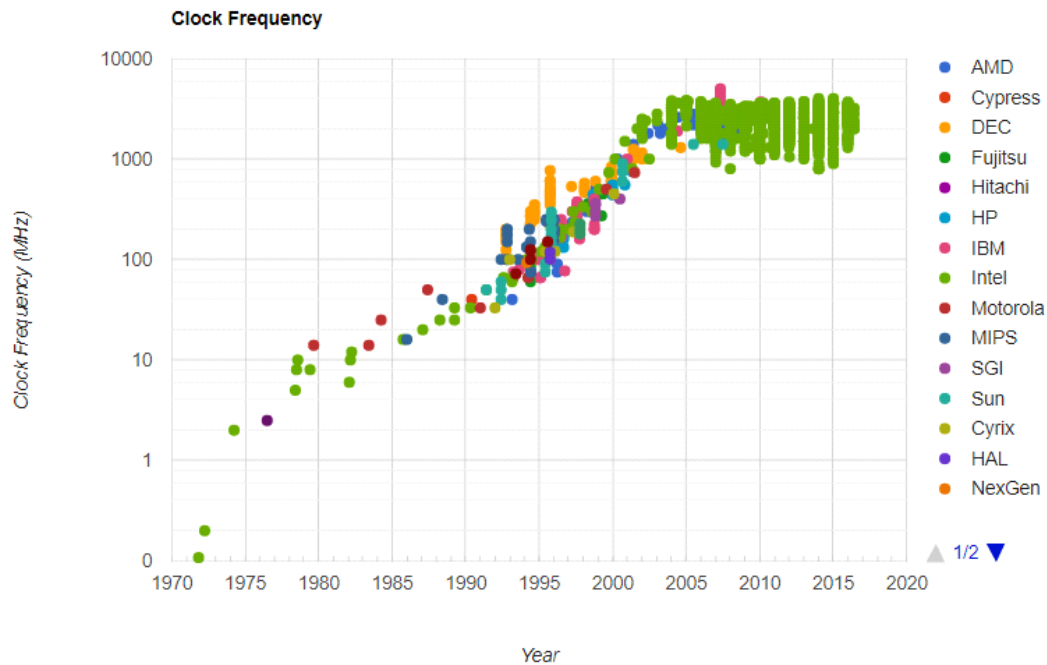
Οι σύγχρονοι επεξεργαστές κατασκευάζονται από τρανζίστορ. Το σχήμα δείχνει το μέγεθος των χαρακτηριστικών των επεξεργαστών σε σχέση με τον χρόνο, όπου το μέγεθος χαρακτηριστικών είναι το ελάχιστο μέγεθος ενός τρανζίστορ ή σύρματος σε ένα τσιπ.



Βλέπουμε ότι το μέγεθος μειώνεται σταθερά με την πάροδο του χρόνου. Καθώς το μέγεθος των χαρακτηριστικών μειώνεται, τα τρανζίστορ γίνονται μικρότερα, λειτουργούν ταχύτερα, καταναλώνουν λιγότερη ενέργεια και μπορούμε να τοποθετήσουμε περισσότερα σε ένα ολοκληρωμένο κύκλωμα. Συνεπώς έχουμε όλο και περισσότερους υπολογιστικούς πόρους.

Επιπλέον, οι σχεδιαστές επεξεργαστών κάνουν τους υπολογιστές πιο γρήγορους αυξάνοντας την συχνότητα του ρολογιού των επεξεργαστών, για να μειώσουν το χρόνο σε κάθε βήμα ενός υπολογισμού.

Clock Frequency



Ωστόσο, κατά την τελευταία δεκαετία βλέπουμε ότι οι ταχύτητες ρολογιού παραμένουν σταθερές.

Οι παραδοσιακές κεντρικές μονάδες επεξεργασίας (CPU) είναι εξαιρετικές στην εκτέλεση πολύπλοκων λειτουργιών ελέγχου. Ωστόσο, δεν είναι απαραίτητα βέλτιστες για τις απαιτήσεις πολλών εφαρμογών που πρέπει να επεξεργάζονται μεγάλες ποσότητες δεδομένων. Με τον ολοένα και πιο έξυπνο κόσμο στον οποίο ζούμε, ο όγκος των δεδομένων που απαιτείται για την επεξεργασία αυξάνεται εκθετικά. Χρειάζεται επιτάχυνση για να γεφυρωθεί το αυξανόμενο χάσμα μεταξύ της ανάγκης επεξεργασίας δεδομένων και των παραδοσιακών δυνατοτήτων της CPU.

Traditional Central Processing Units (CPUs) are excellent at performing complex control functions. However, they are not necessarily optimal for the demands of many applications that need to process large quantities of data. With the increasingly smart world we live in, the amount of data processing needed is increasing exponentially. Acceleration is needed to bridge the growing gap between data-processing need and traditional CPU capability.

Χαρακτηριστικά Υπολογιστή

Χαρακτηριστικά Υπολογιστή:

Λειτουργικό σύστημα: Ubuntu 20.04.5 LTS

Επεξεργαστής: Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz

Πληροφορίες:

```
$ nvcpuuid
vendor id      : GenuineIntel
model name     : Intel(R) Core(TM) i5-5200U CPU @ 2.20GHz
cpu family     : 6
model          : 61
name           : Broadwell Core gen 5 M-5xxx
stepping       : 4
processors     : 4
threads        : 2
clflush size   : 8
L2 cache size  : 256KB
L3 cache size  : 3072KB
flags           : acpi aes apic avx erms avx2 md_clear ibrs stibp l1d_flush
flags           : ssdb cflush cmov cplds cx8 cx16 de dtes f16c ferr fma fpu
flags           : fxsr ht lm mca mce mmx monitor movbe msr mtrr nx osxsave
flags           : pae pat pdcm pge popcnt pse pseg36 selfsnoop speedstep sep
flags           : sse sse2 sse3 ssse3 sse4.1 sse4.2 syscall tm tm2 tsc vme
flags           : xsave xtptr
default target : -tp haswell
```

Κάρτα Γραφικών: NVIDIA GeForce 920M

Πληροφορίες:

```
$ nvidia-smi
```

```
CUDA Driver Version:      11040
NVRM version:             NVIDIA UNIX x86_64 Kernel Module  470.141.03  Thu Jun 30
18:45:31 UTC 2022
```

```
Device Number:            0
Device Name:               NVIDIA GeForce 920M
Device Revision Number:    3.5
Global Memory Size:        2101739520
Number of Multiprocessors: 2
Concurrent Copy and Execution: Yes
Total Constant Memory:     65536
Total Shared Memory per Block: 49152
Registers per Block:       65536
Warp Size:                 32
Maximum Threads per Block: 1024
Maximum Block Dimensions:  1024, 1024, 64
Maximum Grid Dimensions:   2147483647 x 65535 x 65535
Maximum Memory Pitch:      2147483647B
Texture Alignment:         512B
Clock Rate:                 954 MHz
Execution Timeout:         Yes
Integrated Device:         No
Can Map Host Memory:       Yes
Compute Mode:               default
Concurrent Kernels:        Yes
ECC Enabled:                No
Memory Clock Rate:         900 MHz
Memory Bus Width:          64 bits
```

L2 Cache Size:	524288 bytes
Max Threads Per SMP:	2048
Async Engines:	1
Unified Addressing:	Yes
Managed Memory:	Yes
Concurrent Managed Memory:	No
Default Target:	cc35

Εκδόσεις διαθέσιμων Compiler:

NVIDIA	nvc	22.3-0 64-bit target on x86-64 Linux -tp haswell
GNU	gcc	(Ubuntu 9.4.0-1ubuntu1~20.04.1) 9.4.0
	gcc-10	(Ubuntu 10.3.0-1ubuntu1~20.04) 10.3.0
LLVM	clang	clang version 10.0.0-4ubuntu1 Target: x86_64-pc-linux-gnu Thread model: posix

Απλή εκτέλεση κωδικα και προβλήματα

Κώδικας

Πήραμε τον κώδικα από το Github: <https://github.com/61c-teach/sp19-proj4-starter>
Site από αντίστοιχο μάθημα στο Berkeley: [Project 4: Performance Programming](#)

Εκτέλεση κώδικα με GCC

Απλή εκτέλεση του κώδικα χωρίς καμία αλλαγή.

Στο **Makefile** χρησιμοποιώ τα εξής Flags:

```
CFLAGS?=-Wall -Wno-unused-result -march=native -fopenmp -O3
```

Εκτέλεση:

```
$ make clean && make benchmark && ./benchmark benchmark
rm -f *.o
rm -f benchmark
rm -f benchmark_baseline
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c benchmark.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c network.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c layers.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -c volume.c
gcc -Wall -Wno-unused-result -march=native -fopenmp -O3 -o benchmark benchmark.o network.o
layers.o volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Running classification...
78.250000% accuracy
19103874 microseconds
```

Χρόνος Εκτέλεσης: 19,10 sec

Εκτέλεση Κώδικα με NVC

Εκτέλεση του κώδικα χωρίς καμία αλλαγή, με τον compiler της NVIDIA:

Makefile

```
CFLAGS?=-Wall -Minfo -mp -march=native -O3
CC=nvc
```

Στο **Makefile** αφαιρέθηκε το **-Wno-unused-result** καθώς δεν αναγνωρίζεται από τον nvc compiler. Προσθέσαμε όμως το flag **-Minfo** ώστε να παίρνουμε πληροφορίες σχετικά με βελτιστοποιήσεις που γίνονται.

```
$ make clean && make benchmark && ./benchmark benchmark
rm -f *.o
```

```

rm -f benchmark
rm -f benchmark_baseline
nvc -Wall -Minfo=opt -mp -march=native -O3 -c benchmark.c
run_classification:
    141, Memory zero idiom, loop replaced by call to __c_mzero8
nvc -Wall -Minfo=opt -mp -march=native -O3 -c network.c
net_classify:
    147, Loop unrolled 10 times (completely unrolled)
nvc -Wall -Minfo=opt -mp -march=native -O3 -c layers.c
conv_forward:
    109, FMA (fused multiply-add) instruction(s) generated
fc_forward:
    283, FMA (fused multiply-add) instruction(s) generated
softmax_forward:
    361, Loop unrolled 8 times
nvc -Wall -Minfo=opt -mp -march=native -O3 -c volume.c
nvc -Wall -Minfo=opt -mp -march=native -O3 -o benchmark benchmark.o network.o layers.o
volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Segmentation fault (core dumped)

```

το σφάλμα Segmentation fault (core dumped) δεν συμβαίνει εάν αφαιρέσουμε το optimization -O3

```

nvc -Wall -Minfo=opt -mp -march=native -c benchmark.c
nvc -Wall -Minfo=opt -mp -march=native -c network.c
nvc -Wall -Minfo=opt -mp -march=native -c layers.c
conv_forward:
    109, FMA (fused multiply-add) instruction(s) generated
fc_forward:
    283, FMA (fused multiply-add) instruction(s) generated
nvc -Wall -Minfo=opt -mp -march=native -c volume.c
nvc -Wall -Minfo=opt -mp -march=native -o benchmark benchmark.o network.o layers.o
volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Loading batches...
Loading input batch 0...
Running classification...
78.250000% accuracy
30988236 microseconds

```

Το optimization που δημιουργεί το πρόβλημα είναι το `-Mvect`. Ο κώδικας τρέχει κανονικά με `-O3 -Mnovect`.

Χρήση του Valgrind

Ελέγχουμε την εκτέλεση του κώδικα με το Valgrind για σφάλματα Memory Access και Leaks.

```

make CFLAGS="-Wall -Wno-unused-result -march=native -fopenmp -g -O0"
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark

```

✓ Με -O0 και -g ο gcc και ο nvc δεν βγάζουν κάποιο πρόβλημα.

GCC Compiler

```
make CFLAGS="-Wall -Wno-unused-result -march=native -fopenmp -g -O3"  
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark
```

```
==11846== Memcheck, a memory error detector  
==11846== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==11846== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info  
==11846== Command: ./benchmark benchmark  
==11846==  
RUNNING BENCHMARK ON 1200 PICTURES...  
Making network...  
Loading batches...  
Loading input batch 0...  
Running classification...  
78.250000% accuracy  
1335761832 microseconds  
==11846==  
==11846== HEAP SUMMARY:  
==11846==      in use at exit: 0 bytes in 0 blocks  
==11846==    total heap usage: 21,439 allocs, 21,439 frees, 247,283,464 bytes allocated  
==11846==  
==11846== All heap blocks were freed -- no leaks are possible  
==11846==  
==11846== For lists of detected and suppressed errors, rerun with: -s  
==11846== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

NVC Compiler

```
make CFLAGS?="-Wall -Minfo=opt -mp -march=native -g -O3"  
valgrind --leak-check=full --show-leak-kinds=all ./benchmark benchmark
```

```
==14505== Memcheck, a memory error detector  
==14505== Copyright (C) 2002-2017, and GNU GPL'd, by Julian Seward et al.  
==14505== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info  
==14505== Command: ./benchmark benchmark  
==14505==  
RUNNING BENCHMARK ON 1200 PICTURES...  
Making network...  
Loading batches...  
Loading input batch 0...  
==14505== Invalid write of size 8  
==14505==    at 0x404C56: volume_set (volume.c:23)  
==14505==    by 0x401ACC: run_classification (benchmark.c:91)  
==14505==    by 0x402CCA: main (benchmark.c:223)  
==14505== Address 0x62bda30 is 0 bytes after a block of size 24,576 alloc'd  
==14505==    at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-  
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)  
==14505==    by 0x404CD3: make_volume (volume.c:28)  
==14505==    by 0x401A67: run_classification (benchmark.c:82)  
==14505==    by 0x402CCA: main (benchmark.c:223)  
==14505==  
--14505-- VALGRIND INTERNAL ERROR: Valgrind received a signal 11 (SIGSEGV) - exiting
```

```
--14505-- si_code=128; Faulting address: 0x0; sp: 0x1002ca9e10
valgrind: the 'impossible' happened:
  Killed by fatal signal
host stacktrace:
==14505==      at 0x5804F830: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-
linux)
==14505==      by 0x58051B46: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-
linux)
==14505==      by 0x58005BF7: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-
linux)
==14505==      by 0x580A7204: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-
linux)
==14505==      by 0x580F5FD4: ??? (in /usr/lib/x86_64-linux-gnu/valgrind/memcheck-amd64-
linux)
sched status:
  running_tid=1
Thread 1: status = VgTs_Runnable (lwpid 14505)
==14505==      at 0x483B7F3: malloc (in /usr/lib/x86_64-linux-
gnu/valgrind/vgpreload_memcheck-amd64-linux.so)
==14505==      by 0x404CAD: make_volume (volume.c:27)
==14505==      by 0x401A67: run_classification (benchmark.c:82)
==14505==      by 0x402CCA: main (benchmark.c:223)
client stack range: [0x1FFEFFFC000 0x1FFEFFFD000] client SP: 0x1FFEFFD650
valgrind stack range: [0x1002BAA000 0x1002CA9FFF] top usage: 18696 of 1048576
```

Το πρόβλημα εντοπίζεται στο κομμάτι κώδικα `benchmark.c:91` στην συνάρτηση `volume_set()`

benchmark.c:87-94

```
int outp = 1;
for(int d = 0; d < 3; d++) {
    for(int y = 0; y < 32; y++) {
        for(int x = 0; x < 32; x++) {
            volume_set(batchdata[i], x, y, d, ((double)data[outp++])/255.0-0.5);
        }
    }
}
```

🔗 Παρατήρηση 1:

Στο παραπάνω κομμάτι κώδικα, αν προσθέσουμε μια εντολή `if()` ανάμεσα στα `loop`, η οποία δεν επηρεάζει την αρχική λειτουργία του κώδικα, το πρόγραμμα τρέχει κανονικά.

```
int outp = 1;
for(int d = 0; d < 3; d++) {
    for(int y = 0; y < 32; y++) {
        for(int x = 0; x < 32; x++) {
            volume_set(batchdata[i], x, y, d, ((double)data[outp++])/255.0-0.5);
        }
    }
    if(outp==4) printf(".");
}
```

🔗 Παρατήρηση 2:

το πρόγραμμα λειτουργεί κανονικά με παλιότερη έκδοση του NVC compiler `nvc 21.3` οι τιμές των `enviroment`

variables για κάθε έκδοση βρίσκονται στα παρακάτω αρχεία

[nvc-show-v21.3.txt](#)

[nvc-show-v22.3.txt](#)

Κωδικας - Μεταφορά σε Παραμέτρων

Κατανόηση του Κώδικα

Δομή φακέλου

Ο φάκελος περιλαμβάνει τα παρακάτω αρχεία:

```
Makefile
layers.h
snapshot/
layers_baseline.c
test/
benchmark.c
network.c
volume.c
network.h
volume.h
huge_test.sh
network_baseline.c
layers.c
run_test.sh
```

Τα μόνα αρχεία που επεμβαίνουμε είναι:

- layers.c
- network.c
- volume.c

Τα αρχεία layers_baseline.c, network_baseline.c, και volume_baseline.c περιλαμβάνουν την σειριακή έκδοση του κώδικα σαν σημείο αναφοράς για μέτρηση της επιτάχυνσης.

Επισκόπηση Κώδικα

Στον κώδικα χρησιμοποιούνται διάφοροι τύποι δεδομένων. Με τον όρο τύποι δεδομένων, εννοούμε τις δομές που με την typedef έχουν μετονομαστεί ώστε να έχουν νόημα. Επίσης στα header files υπάρχουν περιγραφές για το τι κάνει κάθε συνάρτηση.

Πρώτος τύπος δεδομένων volume_t, περιλαμβάνει έναν τρισδιάστατο πίνακα (ή όγκο) μεταβλητών double. Χρησιμοποιείται για την αναπαράσταση των αποτελεσμάτων κάθε επιπέδου, καθώς και για τα βάρη κάποιων επιπέδων.

Στη συνέχεια, έχουμε τα διαφορετικά είδη επιπέδων: conv, relu, pool, fc, και softmax. Για το καθένα από αυτά υπάρχει:

- Μία δομή δεδομένων που περιέχει την περιγραφή των παραμέτρων του επιπέδου. Κάθε επίπεδο έχει συγκεκριμένο αριθμό παραμέτρων που δεν αλλάζει κατά την διάρκεια της εκτέλεσης του προγράμματος. Για παράδειγμα, το μέγεθος των όγκων της εισόδου και εξόδου ενός επιπέδου δηλώνεται στον ορισμό του CNN.
- Μία συνάρτηση *_forward οποία εκτελεί την κύρια λειτουργία του επιπέδου. Αυτές οι συναρτήσεις αυτές παίρνουν τη δομή δεδομένων του επιπέδου, και έναν πίνακα pointers των όγκων των εισόδων και των εξόδων. Επιπλέον οι μεταβλητές start και end δείκτες σε αυτό τον πίνακα. Αυτό επιτρέπει σε κάθε επίπεδο να επεξεργάζεται μια ομάδα εισόδων. Για παράδειγμα, μπορεί να ορίσουμε start = 5 και end = 9 για την επεξεργασία των εισόδων 5,6,7,8,9 ταυτόχρονα. Τα

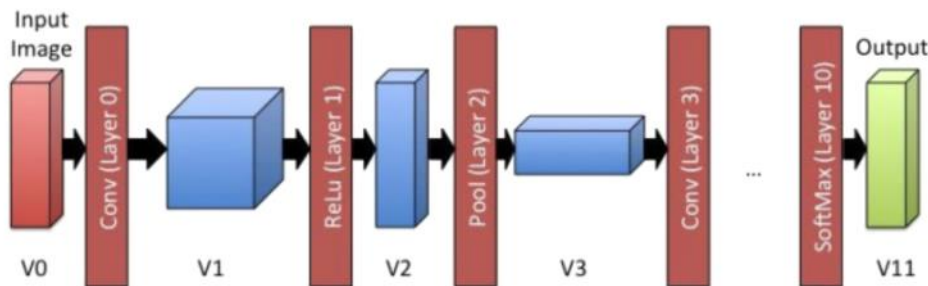
αποτελέσματα επιστρέφονται στους αντίστοιχους πίνακες εξόδων.

- Δύο συναρτήσεις, την `make_*` και `*_load`. Η πρώτη παράγει ένα επίπεδο και με συγκεκριμένο σύνολο παραμέτρων, και η δεύτερη φορτώνει τα βάρη του επιπέδου από αρχείο `*.txt`.

Τελευταία σημαντική δομή δεδομένων είναι η `network_t`, η οποία περιέχει όλες τις πληροφορίες που περιγράφουν το CNN. Αυτό περιλαμβάνει όλα τα επίπεδα, και μία παράμετρο που αφορά στους διάφορους ενδιάμεσους όγκους (volumes). Οι όγκοι (volumes) δεν χρησιμοποιούνται για την αποθήκευση δεδομένων, αυτό γίνεται με τα batches. Χρησιμοποιούνται για να υπάρχουν διαθέσιμες οι διαστάσεις του κάθε όγκου.

Όλα τα ενδιάμεσα δεδομένα αναπαρίστανται ως batches. Περιλαμβάνουν τα ενδιάμεσα δεδομένα που σχετίζονται με ένα σύνολο εικόνων εισόδου. Το `batch_t` είναι συντομογραφία του `volume_t***`, πρόκειται δηλαδή για δισδιάστατους πίνακες δεικτών σε volumes. Η πρώτη διάσταση δηλώνει το επίπεδο που ανήκει το volume (π.χ., V0, V1, κλπ. στο σχήμα) και η δεύτερη διάσταση δηλώνει την είσοδο. Στο αρχικό πρόγραμμα τα batches έχουν μέγεθος 1 (π.χ., επεξεργαζόμαστε ένα παράδειγμα κάθε φορά), για παραλληλοποίηση θα χρειαστούν μεγαλύτερα batches, τα οποία κάνουμε *allocate* στη μνήμη με την συνάρτηση `make_batch` και απελευθερώνουμε από τη μνήμη με την `free_batch`.

Τέλος, η συνάρτηση `net_forward` δέχεται ένα batch (καθώς και δείκτες `start/end`) και εφαρμόζει το CNN για κάθε είσοδο από την αρχή μέχρι το τέλος καλώντας τις συναρτήσεις `forward` για κάθε επίπεδο. Αυτή η συνάρτηση χρησιμοποιείται από την `net_classify`. Παίρνει ένα σύνολο εικόνων, βάζει κάθε μία από αυτές στο volume V0 ενός στοιχείου του batch, και στη συνέχεια τρέχει το δίκτυο σε αυτά. Αποθηκεύουμε τα *likelihoods* (δηλαδή τις τιμές στο τελευταίο επίπεδο του δικτύου) σε έναν 2D πίνακα τύπου `double` ώστε να δούμε που θα ταξινομούσε το δίκτυο την εικόνα.



General structure of the CNN we use for this project.

Μεταφορά Παραμέτρων Νευρωνικού Δικτύου στην GPU

Το `volume`, χρησιμοποιείται για την αναπαράσταση ενεργοποιήσεων (δηλαδή των καταστάσεων) μεταξύ των διαφορετικών επιπέδων του CNN. Έχει τρεις διαστάσεις. Η ερμηνεία τους εξαρτάται από το επίπεδο που τα παράγει. Στο πρώτο επίπεδο, το volume περιέχει τα δεδομένα των εικόνων που θέλουμε να ταξινομήσουμε (όπου βάθος - depth είναι τα τρία χρωματικά κανάλια). Στο τελευταίο επίπεδο το volume περιέχει τις πιθανότητες μία εικόνα να ανήκει σε μία κατηγορία. Τα βάρη - weights αναπαρίστανται σαν array μίας διάστασης με μέγεθος [πλάτοςύψοςβάθος].

```
typedef struct volume {
    int width;
    int height;
    int depth;
    double *weights;
} volume_t;
```

Η αρχιτεκτονική του νευρωνικού δικτύου ορίζεται ως εξής.

```
#define NUM_LAYERS 11

#define NUM_CLASSES 10
typedef struct network {
    volume_t *layers[NUM_LAYERS + 1];

    conv_layer_t *l0;
    relu_layer_t *l1;
    pool_layer_t *l2;
    conv_layer_t *l3;
    relu_layer_t *l4;
    pool_layer_t *l5;
    conv_layer_t *l6;
    relu_layer_t *l7;
    pool_layer_t *l8;
    fc_layer_t *l9;
    softmax_layer_t *l10;
} network_t;
```

Αποτελείται από 11 επίπεδα (NUM_LAYERS 11) , στο `volume_t *layers[]` είναι αποθηκευμένα οι είσοδοι/ έξοδοι κάθε επιπέδου. Το `layers[0]` είναι η εικόνα εσόδου στο νευρωνικό δίκτυο, μεγέθους 32x32x3 και στο `net->layers[0]->weights` βρίσκονται αποθηκευμένα τα pixel της εικόνας.

Τα `layers[1]-layers[11]` περιλαμβάνουν τις εξόδους του κάθε επιπέδου αντίστοιχα.

Οι μεταβλητές `l0 - l10` περιλαμβάνουν τις παραμέτρους του κάθε επιπέδου ανάλογα με τη λειτουργία του καθενός.

- `l0,l3` και `l6` συνελκτικά επίπεδα, `conv_layer_t`
- `l1,l4` και `l7` επίπεδα ReLU, `relu_layer_t`
- `l2,l5` και `l8` επίπεδα Pooling, `pool_layer_t`
- `l9` πλήρως διασυνδεδεμένο επίπεδο, `fc_layer_t`
- `l10` επίπεδο Softmax, `softmax_layer_t`

Η συνάρτηση `make_network()` δεσμεύει χώρο μνήμης για το νευρωνικό δίκτυο και αρχικοποιεί βασικές παραμέτρους.

Για να γίνει η μεταφορά των δεδομένων στην κάρτα γραφικών - device, χρειάζεται να κάνουμε *allocate* τη δομή `net` στο device.

```
network_t *make_network() {
    network_t *net = (network_t *) malloc(sizeof(network_t));
#pragma acc enter data create (net[0:1])
    net->layers[0] = make_volume(32, 32, 3, 0.0);
    net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);
//Code...
}
```

Η *directive* `enter data create` χρησιμοποιείται για τον ορισμό μιας μεταβλητής στη συσκευή και μπορεί να χρησιμοποιηθεί αμέσως μετά την εκχώρηση μιας δυναμικής μεταβλητής μέσω `new`, `malloc` ή `allocate` για να ταιριάζει με τη δημιουργία της μεταβλητής του *host*. Παρομοίως, η *directive* `exit data delete` τοποθετείται λίγο πριν τις εντολές `freed`, `deleted` ή `deallocated` στον *host*. [robFab]

Για κάθε μεταβλητή μέλος του **net**, η απαραίτητη δέσμευση μνήμης στο *device* γίνεται μέσω των αντίστοιχων συναρτήσεων `make_*`()).

Στη συνάρτηση `make_volume()` τοποθετούμε `enter data create` μετά το `malloc` για το `new_vol` και ξεχωριστά για την μεταβλητή `weights` καθώς πρόκειται για δυναμική μεταβλητή. Τέλος, με την `update device(new_vol[0:1],new_vol->weights[0:(width * height * depth)])` αντιγράφουμε στη συσκευή τα δεδομένα του `new_vol`.

```
volume_t *make_volume(int width, int height, int depth, double value) {
    volume_t *new_vol = malloc(sizeof(struct volume));
    #pragma acc enter data create(new_vol[0:1])
    new_vol->weights = malloc(sizeof(double) * width * height * depth);
    #pragma acc enter data create(new_vol->weights[0:(width * height * depth)])
    new_vol->width = width;
    new_vol->height = height;
    new_vol->depth = depth;

    for (int x = 0; x < width; x++) {
        for (int y = 0; y < height; y++) {
            for (int d = 0; d < depth; d++) {
                volume_set(new_vol, x, y, d, value);
            }
        }
    }
    #pragma acc update device(new_vol[0:1],new_vol->weights[0:(width * height * depth)])
    return new_vol;
}
```

Για την αποδέσμευση της μνήμης, στη συνάρτηση `free_volume()` προσθέτουμε `exit data delete(v->weights,v)`

```
void free_volume(volume_t *v) {
    free(v->weights);
    free(v);
    #pragma acc exit data delete(v->weights,v)
}
```

Για κάθε συνάρτηση `make_*`()), τοποθετούμε αντίστοιχα *directive* μετά από κάθε `malloc` και αντιγράφουμε τα δεδομένα στο τέλος της συνάρτησης. Όπου έχουμε κλήση της `make_volume()` δεν χρειάζεται κάποιο επιπλέον *directive*.

Convolutional (layers.c:21-54)

```
conv_layer_t *make_conv_layer(int input_width, int input_height, int input_depth, int filter_width, int num_filters,
    int stride, int pad) {
    conv_layer_t *l = (conv_layer_t *) malloc(sizeof(conv_layer_t));
    #pragma acc enter data create(l[0:1])
    l->output_depth = num_filters;

    //other parameters....
    #pragma acc update device(l[0:1])

    l->filters = malloc(sizeof(volume_t *) * num_filters);
    #pragma acc enter data create(l->filters[0:num_filters])
    for (int i = 0; i < num_filters; i++) {
```

```

    l->filters[i] = make_volume(l->filter_width, l->filter_height,
        l->input_depth, 0.0);
}

l->bias = 0.0;
#pragma acc update device(l->bias)
l->biases = make_volume(1, 1, l->output_depth, l->bias);

return l;
}

```

ReLU (layers.c:167-179)

```

relu_layer_t *make_relu_layer(int input_width, int input_height, int input_depth) {
    relu_layer_t *l = (relu_layer_t *) malloc(sizeof(relu_layer_t));
#pragma acc enter data create(l[0:1])
    l->input_depth = input_depth;
    l->input_width = input_width;
    l->input_height = input_height;

    l->output_width = l->input_width;
    l->output_height = l->input_height;
    l->output_depth = l->input_depth;
#pragma acc update device(l[0:1])
    return l;
}

```

Pool (layers.c:196-213)

```

pool_layer_t *make_pool_layer(int input_width, int input_height, int input_depth, int pool_width, int stride) {
    pool_layer_t *l = (pool_layer_t *) malloc(sizeof(pool_layer_t));
#pragma acc enter data create(l[0:1])
    l->pool_width = pool_width;
    //other parameters...
#pragma acc update device(l[0:1])
    return l;
}

```

Fully Connected (layers.c:261-284)

```

fc_layer_t *make_fc_layer(int input_width, int input_height, int input_depth, int num_neurons) {
    fc_layer_t *l = (fc_layer_t *) malloc(sizeof(fc_layer_t));
#pragma acc enter data create(l[0:1])
    l->output_depth = num_neurons;
    //other parameters...
#pragma acc update device(l[0:1])
    l->filters = (volume_t **) malloc(sizeof(volume_t *) * num_neurons);
#pragma acc enter data create(l->filters[0:num_neurons])
    for (int i = 0; i < l->output_depth; i++) {
        l->filters[i] = make_volume(1, 1, l->num_inputs, 0.0);
    }

    l->bias = 0.0;
#pragma acc update device(l->bias)
    l->biases = make_volume(1, 1, l->output_depth, l->bias);
}

```

```

    return l;
}

```

Softmax (layers.c:331-347)

```

softmax_layer_t *make_softmax_layer(int input_width, int input_height, int input_depth) {
    softmax_layer_t *l = (softmax_layer_t*) malloc(sizeof(softmax_layer_t));
#pragma acc enter data create(l[0:1])
    l->input_depth = input_depth;
    // other parameters...
#pragma acc update device(l[0:1])
    l->likelihoods = (double*) malloc(sizeof(double) * l->output_depth);
#pragma acc enter data create(l->likelihoods[0:l->output_depth])

    return l;
}

```

Στην συνάρτηση `free_network()` προσθέτουμε *directives* `exit data delete` πριν από κάθε `free`.

```

void free_network(network_t *net) {
    for (int i = 0; i < NUM_LAYERS + 1; i++){
        free_volume(net->layers[i]);
    }

    // Free each conv layer's filters and biases
    for (int f = 0; f < net->l0->output_depth; f++) {
        free_volume(net->l0->filters[f]);
    }
#pragma acc exit data delete(net->l0->filters)
    free(net->l0->filters);
    free_volume(net->l0->biases);

    for (int f = 0; f < net->l3->output_depth; f++) {
        free_volume(net->l3->filters[f]);
    }
#pragma acc exit data delete(net->l3->filters)
    free(net->l3->filters);
    free_volume(net->l3->biases);

    for (int f = 0; f < net->l6->output_depth; f++) {
        free_volume(net->l6->filters[f]);
    }
#pragma acc exit data delete(net->l6->filters)
    free(net->l6->filters);
    free_volume(net->l6->biases);

    // Free FC layer filters and biases
    for (int f = 0; f < net->l9->output_depth; f++) {
        free_volume(net->l9->filters[f]);
    }
#pragma acc exit data delete(net->l9->filters)
    free(net->l9->filters);
    free_volume(net->l9->biases);
}

```

```
// Free softmax layer likelihoods
#pragma acc exit data delete(net->l10->likelihoods)
    free(net->l10->likelihoods);

#pragma acc exit data delete(net->l0,net->l1,net->l2,net->l3,net->l4,net->l5,net->l6,net->l7,net->l8,net->l9,net->l10)
    free(net->l0);
    //free l1-l10...
#pragma acc exit data delete(net)
    free(net);
}
```

Επιπλέον, στις συναρτήσεις `copy_volume`, `conv_load` και `fc_load` έχουν προστεθεί τα απαραίτητα *directives* για ενημέρωση των παραμέτρων.

```
conv_load (layers.c:128-165)
void conv_load(conv_layer_t *l, const char *file_name) {
    int filter_width, filter_height, depth, filters;

    FILE *fin = fopen(file_name, "r");
    //Code...
    //Update Weights and Biases on Device
    int we=filter_width*filter_height*depth;
    for(int f = 0; f < filters; f++) {
#pragma acc update device(l->filters[f]->weights[0:we])
    }

#pragma acc update device(l->biases->weights[0:l->output_depth])

    fclose(fin);
}

fc_load (layers.c:305-329)
void fc_load(fc_layer_t *l, const char *filename) {
    FILE *fin = fopen(filename, "r");
    //...Code...

    //Update Weights and Biases on Device
    for(int f = 0; f < l->output_depth; f++) {
#pragma acc update device(l->filters[f]->weights[0:l->num_inputs])
    }
#pragma acc update device(l->biases->weights[0:l->output_depth])
    fclose(fin);
}
```

Εκτέλεση Κώδικα

Χωρίς -acc

RUNNING BENCHMARK ON 1200 PICTURES...

Making network...

16 972 microseconds

Με -acc

RUNNING BENCHMARK ON 1200 PICTURES...

Making network...

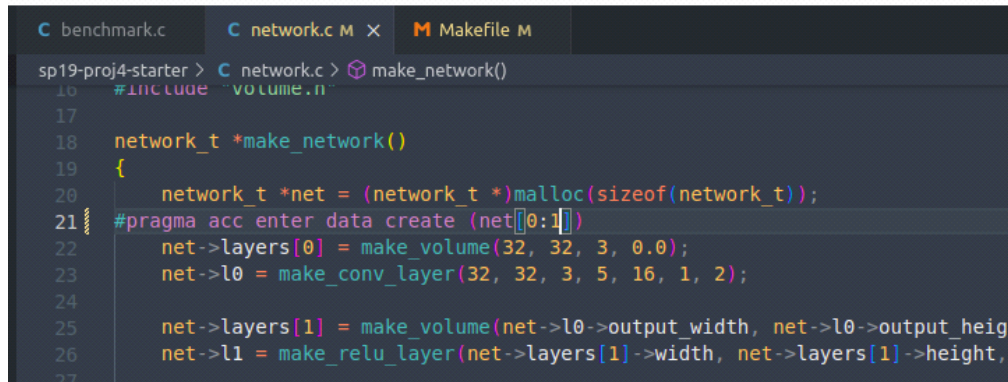
167 979 microseconds

Make_network

Makefile:

CC=nvc

CFLAGS?=-Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect



```
16 #include "volume.h"
17
18 network_t *make_network()
19 {
20     network_t *net = (network_t *)malloc(sizeof(network_t));
21     #pragma acc enter data create (net[0:1])
22     net->layers[0] = make_volume(32, 32, 3, 0.0);
23     net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);
24
25     net->layers[1] = make_volume(net->l0->output_width, net->l0->output_height, 3, 0.0);
26     net->l1 = make_relu_layer(net->layers[1]->width, net->layers[1]->height, 3, 0.0);
27 }
```

Make_network()

Κάνει allocate τα επίπεδα

- Να γίνουν allocate και στο device

*_load()

\$ make benchmark

nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -c benchmark.c

run_classification:

141, Memory zero idiom, loop replaced by call to __c_mzero8

nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -c network.c

make_network:

22, Generating enter data create(net[:1])

nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -c layers.c

conv_forward:

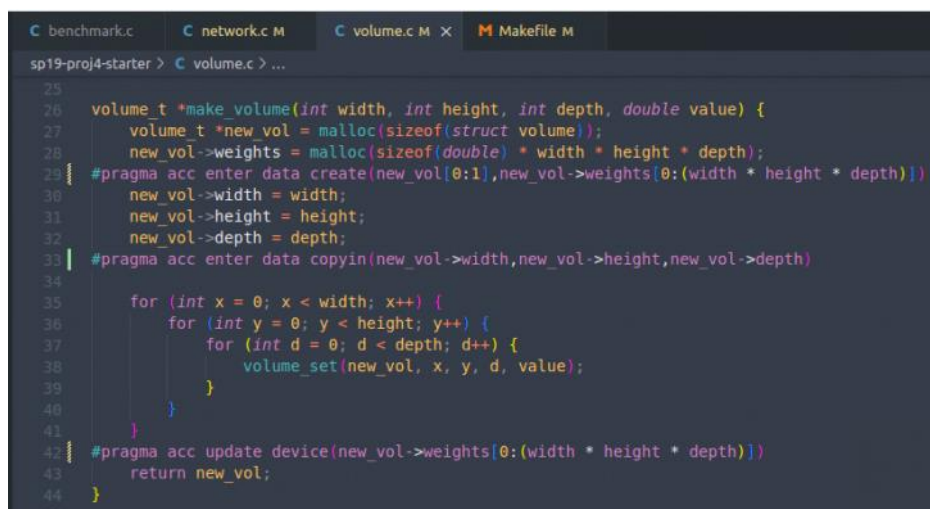
112, FMA (fused multiply-add) instruction(s) generated

fc_forward:

286, FMA (fused multiply-add) instruction(s) generated

nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -c volume.c

nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -o benchmark benchmark.o network.o layers.o volume.o -lm



```
25
26 volume_t *make_volume(int width, int height, int depth, double value) {
27     volume_t *new_vol = malloc(sizeof(struct volume));
28     new_vol->weights = malloc(sizeof(double) * width * height * depth);
29     #pragma acc enter data create(new_vol[0:1], new_vol->weights[0:(width * height * depth)])
30     new_vol->width = width;
31     new_vol->height = height;
32     new_vol->depth = depth;
33     #pragma acc enter data copyin(new_vol->width, new_vol->height, new_vol->depth)
34
35     for (int x = 0; x < width; x++) {
36         for (int y = 0; y < height; y++) {
37             for (int d = 0; d < depth; d++) {
38                 volume_set(new_vol, x, y, d, value);
39             }
40         }
41     }
42     #pragma acc update device(new_vol->weights[0:(width * height * depth)])
43     return new_vol;
44 }
```

```
$ make benchmark
```

```
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mno vect -c volume.c
```


```
make_volume:
```

```
30, Generating enter data create(new_vol->weights[:depth*(height*width)],new_vol[:1])
```

```
32, Generating enter data copyin(new_vol->depth,new_vol->width,new_vol->height)
```

```
43, Generating update device(new_vol->weights[:depth*(height*width)])
```

```
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mno vect -o benchmark benchmark.o network.o layers.o volume.o -lm
```



```
C benchmark.c C network.c M C volume.c M X Makefile M
sp19-proj4-starter > C volume.c > ...
59
60 void free_volume(volume_t *v) {
61     free(v->weights);
62     free(v);
63     #pragma acc exit data delete(v->weights[0:(v->height*v->width*v->depth)],v[0:1])
64 }
65
```

```
$ make benchmark
```

```
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mno vect -c volume.c
```

```
make_volume:
```

```
30, Generating enter data create(new_vol->weights[:depth*(height*width)],new_vol[:1])
```

```
32, Generating enter data copyin(new_vol->depth,new_vol->width,new_vol->height)
```

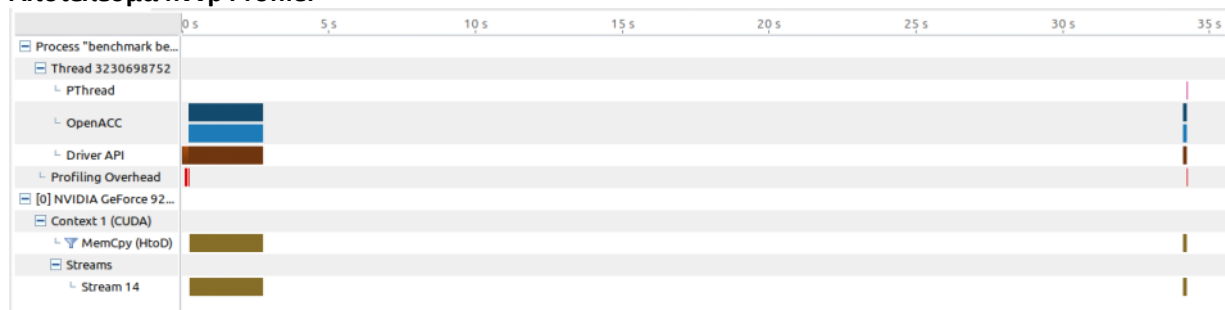
```
43, Generating update device(new_vol->weights[:depth*(height*width)])
```

```
free_volume:
```

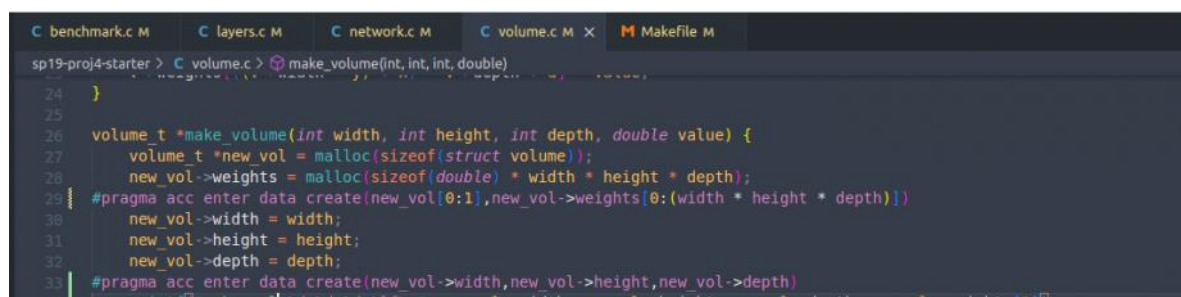
```
64, Generating exit data delete(v[:1],v->weights[:v->depth*(v->width*v->height)])
```

```
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mno vect -o benchmark benchmark.o network.o layers.o volume.o -lm
```

Αποτέλεσμα nnvp Profiler



Φαίνεται να αντιγράφει δεδομένα από host to Device στην αρχή και στο τέλος του προγράμματος
Τα block δεξιά αντιστοιχούν στο delete(...)



```
C benchmark.c M C layers.c M C network.c M C volume.c M X Makefile M
sp19-proj4-starter > C volume.c > make_volume(int, int, int, double)
24 }
25
26 volume_t *make_volume(int width, int height, int depth, double value) {
27     volume_t *new_vol = malloc(sizeof(struct volume));
28     new_vol->weights = malloc(sizeof(double) * width * height * depth);
29     #pragma acc enter data create(new_vol[0:1],new_vol->weights[0:(width * height * depth)])
30     new_vol->width = width;
31     new_vol->height = height;
32     new_vol->depth = depth;
33     #pragma acc enter data create(new_vol->width,new_vol->height,new_vol->depth)
34     printf("make vol %d %d %d %f\n", new_vol->width, new_vol->height, new_vol->depth, new_vol->weights[0]);

```

```

C benchmark.c M  C layers.c M  C network.c M  C volume.c M X  Makefile M
sp19-proj4-starter > C volume.c > make_volume(int, int, int, double)
24 }
25
26 volume_t *make_volume(int width, int height, int depth, double value) {
27     volume_t *new_vol = malloc(sizeof(struct volume));
28     new_vol->weights = malloc(sizeof(double) * width * height * depth);
29     #pragma acc enter data create(new_vol[0:1], new_vol->weights[0:(width * height * depth)])
30     new_vol->width = width;
31     new_vol->height = height;
32     new_vol->depth = depth;
33     #pragma acc enter data create(new_vol->width, new_vol->height, new_vol->depth)
34     printf("make_vol| %d %d %d %f\n", new_vol->width, new_vol->height, new_vol->depth, new_vol->weights[0]);
35
36     for (int x = 0; x < width; x++) {
37         for (int y = 0; y < height; y++) {
38             for (int d = 0; d < depth; d++) {
39                 volume_set(new_vol, x, y, d, value);
40             }
41         }
42     }
43     #pragma acc update device(new_vol->width, new_vol->height, new_vol->depth, new_vol->weights[0:(width * height * depth)])
44     return new_vol;
45 }
46

```

```

19 network_t *make_network()
20 {
21     network_t *net = (network_t *)malloc(sizeof(network_t));
22     #pragma acc enter data create (net[0:1])
23     net->layers[0] = make_volume(32, 32, 3, 0.0);
24     printf("cpu: %d %d %d %f\n", net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0]);
25
26     #pragma acc update self(net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0:(32*32*3)])
27     printf("update: %d %d %d %f\n", net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0]);
28     net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);

```

Output::

```

Making network...
make_vol 32 32 3 0.000000
cpu: 32 32 3 0.000000
update:32 32 3 0.000000

```

```

network_t *make_network()
{
    network_t *net = (network_t *)malloc(sizeof(network_t));
    #pragma acc enter data create (net[0:1])
    net->layers[0] = make_volume(32, 32, 3, 0.0);
    // net->layers[0]->width=22;//TEST
    // printf("cpu: %d %d %d %f\n", net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0]);//TEST
    // Den Xreiazetai...?
    // #pragma acc update self(net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0:(32*32*3)])
    // printf("update: %d %d %d %f\n", net->layers[0]->width, net->layers[0]->height, net->layers[0]->depth, net->layers[0]->weights[0]);//TEST
    net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);
    net->layers[1] = make_volume(net->l0->output_width, net->l0->output_height, net->l0->output_depth, 0.0);
    // net->layers[1]->width=55;//TEST
    // printf("l[1]cpu: %d %d %d %f\n", net->layers[1]->width, net->layers[1]->height, net->layers[1]->depth, net->layers[1]->weights[0]);//TEST
    // #pragma acc update self(net->layers[1]->width, net->layers[1]->height, net->layers[1]->depth, net->layers[1]->weights[0:(32*32*3)])
    // printf("l[1]update: %d %d %d %f\n", net->layers[1]->width, net->layers[1]->height, net->layers[1]->depth, net->layers[1]->weights[0]);//TEST
    net->l1 = make_relu_layer(net->layers[1]->width, net->layers[1]->height, net->layers[1]->depth);

```

Make_conv_layer

```

C benchmark.c sp19-proj4-nvc M C network.c M C layers.c X C benchmark.c sp19-proj4-nvc C volume.c M
sp19-proj4-nvc > C layers.c > @ make_conv_layer(int, int, int, int, int, int, int)
17 #include "layers.h"
18 #include "volume.h"
19
20 conv_layer_t *make_conv_layer(int input_width, int input_height, int input_depth, int filter_width, int num_filters,
21                               int stride, int pad) {
22     conv_layer_t *l = (conv_layer_t *) malloc(sizeof(conv_layer_t));
23     #pragma acc enter data create(l[0:1])
24     l->output_depth = num_filters;
25     l->filter_width = filter_width;
26     l->input_depth = input_depth;
27     l->input_width = input_width;
28     l->input_height = input_height;
29     l->filter_height = l->filter_width;
30     l->stride = stride;
31     l->pad = pad;
32
33     l->output_width = (l->input_width + l->pad * 2 - l->filter_width) /
34                     l->stride + 1;
35     l->output_height = (l->input_height + l->pad * 2 - l->filter_height) /
36                      l->stride + 1;
37     #pragma acc update device(l[0:1])
38     l->output_height=88;
39     #pragma acc update self(l[0:1])
40     printf("l->output_height %d\n", l->output_height);
41
42
43     l->filters = malloc(sizeof(volume_t) * num_filters);
44     #pragma acc enter data create(l->filters[0:num_filters])
45     for (int i = 0; i < num_filters; i++) {
46         l->filters[i] = make_volume(l->filter_width, l->filter_height,
47                                   l->input_depth, 38.0);
48         #pragma acc update device(l->filters[i][0:1], l->filters[i]->weights[0:(l->filter_width*l->filter_height*l->input_depth)])
49     }
50     l->filters[0]->width=111;
51     volume_set(l->filters[0], 2.0, 6.23);
52     printf("l->filters %d\n", l->filters[0]->width);
53     printf("volume_get(l,0,0,0) %3.2f\n", volume_get(l->filters[0], 2.0, 6));
54     for (int i = 0; i < num_filters; i++) {
55         #pragma acc update self(l->filters[i][0:1], l->filters[i]->weights[0:(l->filter_width*l->filter_height*l->input_depth)])
56     }
57
58     printf("l->filters %d\n", l->filters[0]->width);
59     printf("volume_get(l,0,0,0) %3.2f\n", volume_get(l->filters[0], 2.0, 6));
60
61 }

```

Δεν χρειάζεται update τα weights γίνεται στην make_volume των filters όμως πρέπει να γίνει allocate για τα filters.

```

oliad@osu:~/Documents/Programming_2021/Berkeley/sp19-proj4-nvc$ make benchmark
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -g -c layers.c
make_conv_layer:
24, Generating enter data create(l[0:1])
38, Generating update device(l[0:1])
40, Generating update self(l[0:1])
44, Generating enter data create(l->filters[:num_filters])
52, Accelerator clause: upper bound for dimension 0 of array 'filters' is unknown
Generating update device(l->filters->[0:1][0:1], l->filters->weights[l->input_depth*(l->filter_width*l->filter_height))
60, Accelerator clause: upper bound for dimension 0 of array 'filters' is unknown
Generating update self(l->filters->[0:1][0:1], l->filters->weights[l->input_depth*(l->filter_width*l->filter_height))
conv forward:
135, FMA (fused multiply-add) instruction(s) generated
fc_forward:
309, FMA (fused multiply-add) instruction(s) generated
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -Mnovect -g -o benchmark benchmark.o network.o layers.o volume.o -lm
oliad@osu:~/Documents/Programming_2021/Berkeley/sp19-proj4-nvc$ ./benchmark benchmark
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
cpu: 22 32 3 88.000000
gpu: 32 32 3 0.000000
l->output_height 32
l->filters 111
volume_get(l,0,0,0) 23.00
l->filters 5
volume_get(l,0,0,0) 38.00
update:32 32 3 0.000000
133878 microseconds

```

```

sp19-proj4-nvc > C layers.c > @ make_conv_layer(int, int, int, int, int, int, int)
17 #include "layers.h"
18 #include "volume.h"
19
20 conv_layer_t *make_conv_layer(int input_width, int input_height, int input_depth, int filter_width, int num_filters,
21                               int stride, int pad) {
22     conv_layer_t *l = (conv_layer_t *) malloc(sizeof(conv_layer_t));
23     #pragma acc enter data create(l[0:1])
24     l->output_depth = num_filters;
25     l->filter_width = filter_width;
26     l->input_depth = input_depth;
27     l->input_width = input_width;
28     l->input_height = input_height;
29     l->filter_height = l->filter_width;
30     l->stride = stride;
31     l->pad = pad;
32
33     l->output_width = (l->input_width + l->pad * 2 - l->filter_width) /
34                     l->stride + 1;
35     l->output_height = (l->input_height + l->pad * 2 - l->filter_height) /
36                      l->stride + 1;
37     //copy parameters to device
38     #pragma acc update device(l[0:1])
39     l->output_height=88;
40     #pragma acc update self(l[0:1])
41     printf("l->output_height %d\n", l->output_height);
42
43     l->filters = malloc(sizeof(volume_t) * num_filters);
44     #pragma acc enter data create(l->filters[0:num_filters])
45     for (int i = 0; i < num_filters; i++) {
46         l->filters[i] = make_volume(l->filter_width, l->filter_height,
47                                   l->input_depth, 38.0);
48         #pragma acc update device(l->filters[i][0:1], l->filters[i]->weights[0:(l->filter_width*l->filter_height*l->input_depth)])
49     }
50     l->filters[0]->width=111;
51     volume_set(l->filters[0], 2.0, 6.23);
52     printf("l->filters %d\n", l->filters[0]->width);
53     printf("volume_get(l,0,0,0) %3.2f\n", volume_get(l->filters[0], 2.0, 6));
54     for (int i = 0; i < num_filters; i++) {
55         #pragma acc update self(l->filters[i][0:1], l->filters[i]->weights[0:(l->filter_width*l->filter_height*l->input_depth)])
56     }
57
58     printf("l->filters %d\n", l->filters[0]->width);
59     printf("volume_get(l,0,0,0) %3.2f\n", volume_get(l->filters[0], 2.0, 6));
60
61 }

```

Αντιγράφω
! στο device

Αντιγράφω
ΤΙΣ ΠΑΡΑΜΕΤΡΟΥΣ ΣΤΟ
DEVICE

TEST

Για Update Self (10)

```
#pragma acc update self(net->input_width,net->input_height,net->input_depth,net->input_depth,net->output_width,net->output_height,net->
```



```

l0->output_depth)
#pragma acc update self((net->l0->filter_width,net->l0->filter_width,net->l0->output_depth,net->l0->pad,net->l0->stride)
for (int j=0; j<l6; j++){
#pragma acc update self((net->l0->filters[j])[0:l])
for (int i = 0; i < 75; i++){
#pragma acc update self((net->l0->filters[j]->weights[i])
}
}

```

```

sp19-proj4-nvc-2 > C network.c > make_network()
22 network_t *net = (network_t *)malloc(sizeof(network_t));
23 #pragma acc enter data create (net[0:l])
24 net->layers[0] = make_volume(32, 32, 3, 0.0);
25 net->l0 = make_conv_layer(32, 32, 3, 5, 16, 1, 2);
26
27 printf("Net: l0\n");
28 printf("Input(%d,%d,%d) \n", ++net->l0->input_width, ++net->l0->input_height, ++net->l0->input_depth);
29 printf("Output(%d,%d,%d) \n", ++net->l0->output_width, ++net->l0->output_height, ++net->l0->output_depth);
30 printf("filter(%d,%d,%d) pad(%d) stride(%d) \n",
31 ++net->l0->filter_width, ++net->l0->filter_width, ++net->l0->output_depth, ++net->l0->pad, ++net->l0->stride);
32 for (int f=0; f<l6; f++){
33     int count=5; // (net->l0->filters[f]->width)*(net->l0->filters[f]->height)*(net->l0->filters[f]->depth);
34     printf("Filter %d:(%d,%d,%d)\n", f,
35 ++net->l0->filters[f]->width, ++net->l0->filters[f]->height, ++net->l0->filters[f]->depth);
36     for (int i = 0; i < count; i++){
37         net->l0->filters[f]->weights[i] = (double)rand() / RAND_MAX;
38         printf("%3.2f ", net->l0->filters[f]->weights[i]);
39     }
40     putchar('\n'); putchar('\n');
41 }
42 #pragma acc update self((net->l0->input_width,net->l0->input_height,net->l0->input_depth,net->l0->output_width,net->l0->output_height,net->l0->output_depth,net->l0->filter_width,net->l0->filter_width,net->l0->output_depth,net->l0->pad,net->l0->stride)
43 for (int j=0; j<l6; j++){
44     #pragma acc update self((net->l0->filters[j])[0:l])
45     for (int i = 0; i < 75; i++){
46         #pragma acc update self((net->l0->filters[j]->weights[i])
47     }
48 }
49
50 printf("Net: l0 Update\n");
51 printf("Input(%d,%d,%d) \n", net->l0->input_width, net->l0->input_height, net->l0->input_depth);
52 printf("Output(%d,%d,%d) \n", net->l0->output_width, net->l0->output_height, net->l0->output_depth);
53 printf("filter(%d,%d,%d) pad(%d) stride(%d) \n",
54 net->l0->filter_width, net->l0->filter_width, net->l0->output_depth, net->l0->pad, net->l0->stride);

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL JUPYTER

```

143032 microseconds
olias@asus:~/Documents/Programming_2021/Berkeley/sp19-proj4-nvc-2$ make benchmark && ./benchmark benchmark
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -g -c network.c
make network:
24, Generating enter data create(net[:l])
41, Generating update self((net->l0->output_depth,net->l0->input_width,net->l0->output_height,net->l0->input_depth,net->l0->input_height,net->l0->output_width,net->l0->output_width,net->l0->output_depth)
>stride,net->l0->pad,net->l0->output_depth,net->l0->filter_width)
44: Accelerator clause: upper bound for dimension 0 of array 'filters' is unknown
48, Generating update self((net->l0->filters->weights[i])
nvc -Wall -tp=native -c99 -mp -Minfo=all -acc -g -o benchmark benchmark.o network.o layers.o volume.o -lm
RUNNING BENCHMARK ON 1200 PICTURES...
Making network...
Net: l0

```

Ποιά είναι το upper bound σε αυτές τις περιπτώσεις?

Conv Load();

```

sp19-proj4-nvc-2 > C layers.c > conv_load(conv_layer_t*, const char *)
144     volume_set[l->filters[f], x, y, d, val];
145 }
146 }
147 }
148 }
149 }
150 for (int d = 0; d < l->output_depth; d++) {
151     double val;
152     fscanf(fin, "%lf", &val);
153     volume_set[l->biases, 0, 0, d, val];
154 }
155 //Update Weights and Biases on Device
156 int we=filter_width*filter_height*depth;
157 for (int f = 0; f < filters; f++) {
158     #pragma acc update device(l->filters[f]->weights[0:we])
159 }
160 }
161 #pragma acc update device(l->biases->weights[0:l->output_depth])
162 }
163 fclose(fin);
164 }

```

conv_load:

```

159, Generating update device(l->filters->weights[:we])
163, Generating update device(l->biases->weights[:l->output_depth])

```

Προσθήκες για TEST

benchmark.c	fdump_volume()	: για εκτύπωση volume σε αρχείο Αντίστοιχο dump_volume
layers.c	conv_fprint	: εκτύπωση convolutional layer σε αρχείο αντίστοιχο conv_load
	fc_fprint	: εκτύπωση fully connected layer σε αρχείο Αντίστοιχο fc_load
layers.h	conv_fprint fc_fprint	Δηλώσεις

[Problems]

```
sp19-proj4-nvc > C layers.c > make_conv_layer(int, int, int, int, int, int, int)
32 #pragma acc enter data create (l->output_depth, l->filter_width, l->input_depth, l->input_width,
33     l->output_width = (l->input_width + l->pad * 2 - l->filter_width) /
34     l->stride + 1;
35     l->output_height = (l->input_height + l->pad * 2 - l->filter_height) /
36     l->stride + 1;
37     l->filters = malloc(sizeof(volume_t *) * num_filters);
38 #pragma acc enter data create (l->output_width, l->output_height, l->filters[0:num_filters])
39 l->output_height=33;
40 #pragma acc data update self(l->output_height)
41 printf("%d\n", l->output_height);
42
43     for (int i = 0; i < num_filters; i++) {
44         l->filters[i] = make_volume(l->filter_width, l->filter_height,
45         l->input_depth, 0.0);
46     }
47
```

Printf output:

33

Τυπώνει την τιμή cpu

Volume structs και Batches

```
typedef struct volume {  
    int width;  
    int height;  
    int depth;  
    double *weights;  
} volume_t;
```

Οι όγκοι- **Volume** χρησιμοποιούνται για την αναπαράσταση των *ενεργοποιήσεων* (δηλαδή των καταστάσεων) μεταξύ των διαφόρων επιπέδων του CNN. Έχουν τρεις διαστάσεις. Η ερμηνεία του περιεχομένου τους εξαρτάται από το επίπεδο που τα παρήγαγε. Πριν από την πρώτη επανάληψη, ο Όγκος περιέχει τα δεδομένα της εικόνας που θέλουμε να ταξινομήσουμε (όπου βάθος - depth είναι οι τρεις χρωματικές διαστάσεις). Μετά το τελευταίο επίπεδο του CNN, ο όγκος έχει τις πιθανότητες μίας εικόνας να ανήκει σε συγκεκριμένη κατηγορία.

```
typedef volume_t** batch_t;
```

Οργανώνουμε τα δεδομένα ως " ομάδες - batches" όγκων. Κάθε batch αποτελείται από έναν αριθμό δειγμάτων, καθένα από τα οποία περιέχει έναν όγκο για κάθε ενδιαμέσο επίπεδο. Έστω ότι έχουμε L επίπεδα και ένα σύνολο από N εικόνων εισόδου. Τότε το **batch[i][n]** περιέχει τον όγκο στο επίπεδο I για την εικόνα εισόδου n. Με τη χρήση batches, μπορούμε να επεξεργαζόμαστε πολλές εικόνες ταυτόχρονα σε κάθε εκτέλεση των forward συναρτήσεων των διαφορετικών επιπέδων.

```
void net_classify(network_t *net, volume_t **input, double **likelihoods, int n) {  
    batch_t *b = make_batch(net, 1);  
    for (int i = 0; i < n; i++) {  
        copy_volume(b[0][0], input[i]);  
        net_forward(net, b, 0, 0);  
        for (int j = 0; j < NUM_CLASSES; j++) {  
            likelihoods[i][j] = b[11][0]->weights[j];  
        }  
    }  
    free_batch(b, 1);  
}
```

<-b[LAYERS][1] περιέχει το output κάθε επιπέδου για [1] εικόνα
Για κάθε εικόνα i
Αντιγράφει στο b[0][0] το input[i]
Net forward pass

Αντιγράφει στο likelihoods τις εξόδους b[11][0]
Likelihoods[image][NUM_CLASSES]

```
void conv_forward(conv_layer_t *l, volume_t **inputs, volume_t **outputs, int start, int end) {  
    for (int i = start; i <= end; i++) {  
        volume_t *in = inputs[i];  
        volume_t *out = outputs[i];  
  
        int stride = l->stride;
```

```

for(int f = 0; f < l->output_depth; f++) {
    volume_t *filter = l->filters[f];
    int y = -l->pad;
    for(int out_y = 0; out_y < l->output_height; y += stride, out_y++) {
        int x = -l->pad;
        for(int out_x = 0; out_x < l->output_width; x += stride, out_x++) {

            // Take sum of element-wise product
            double sum = 0.0;
            for(int fy = 0; fy < filter->height; fy++) {
                int in_y = y + fy;
                for(int fx = 0; fx < filter->width; fx++) {
                    int in_x = x + fx;
                    if(in_y >= 0 && in_y < in->height && in_x >= 0 && in_x < in->width) {
                        for(int fd = 0; fd < filter->depth; fd++) {
                            sum += volume_get(filter, fx, fy, fd) * volume_get(in, in_x, in_y, fd);
                        }
                    }
                }
            }

            sum += l->biases->weights[f];
            volume_set(out, out_x, out_y, f, sum);
        }
    }
}
}
}
}
}

```

Διαφορά Float και double

Execution time **double:**

```
~/sp19-proj4-starter-master$ time ./benchmark benchmark
```

```
RUNNING BENCHMARK ON 1200 PICTURES...
```

```
Making network...
```

```
Loading batches...
```

```
Loading input batch 0...
```

```
Running classification...
```

```
78.250000% accuracy
```

```
24246882 microseconds
```

```
real 0m24,271s
```

```
user 0m24,146s
```

```
sys 0m0,105s
```

```
---
```

Execution time **float:**

```
~/sp19-proj4-gcc-float$ time ./benchmark benchmark
```

```
RUNNING BENCHMARK ON 1200 PICTURES...
```

```
Making network...
```

```
Loading batches...
```

```
Loading input batch 0...
```

```
Running classification...
```

```
78.250000% accuracy
```

```
24716419 microseconds
```

```
real 0m24,728s
```

```
user 0m24,630s
```

```
sys 0m0,076s
```

C Structs στην OpenACC

Η OpenACC υποστηρίζει στατικά arrays και pointers σε δυναμικά arrays σε δομές struct.

```
typedef struct{
    float x, y, z;
}point;

extern point base[1000];

void vecaddgpu( point *restrict r, int n ){
    #pragma acc parallel loop present(base) copyout(r[0:n])
    for( int i = 0; i < n; ++i ){
        r[i].x = base[i].x;
        r[i].y = sqrtf( base[i].y*base[i].y + base[i].z*base[i].z );
        r[i].z = 0;
    }
}
```

Ένας pointer σε βαθμωτό struct αντιμετωπίζεται σαν arrays με ένα μόνο στοιχείο και το ορίζουμε σαν `r[0:1]`

```
typedef struct{
    base[1000];
    int n;
    float *x, *y, *z;
}point;

extern point A;

void vecaddgpu(){
    #pragma acc parallel loop copyin(A) \
    copyout(A.x[0:A.n], A.y[0:A.n], A.z[0:A.n])
    for( int i = 0; i < A.n; ++i ){
        A.x[i] = A.base[i];
        A.y[i] = sqrtf( A.base[i] );
        A.z[i] = 0;
    }
}
```

Αντιγράφουμε το `A` στο device, και με αυτό τον τρόπο αντιγράφονται και τα μέλη του struct το στατικό array `A.base` και το βαθμωτό `A.n`. Τα δυναμικά μέλη `A.x`, `A.y` και `A.z` αντιγράφονται μετά στην συσκευή. Το struct `A` πρέπει να αντιγραφεί πριν τα δυναμικά μέλη του, είτε τοποθετώντας το struct νωρίτερα στο data clause, είτε κάνοντας copy ή create στο device. Εάν το struct δεν είναι preset στο device όταν αντιγράφονται τα δυναμικά μέλη, τότε δεν είναι δυνατή η πρόσβαση στα δυναμικά μέλη, όπως το `A.x[i]`

Ο pointer σε ένα struct αντιμετωπίζεται σαν array με ένα στοιχείο. Εάν το struct περιέχει μέλη pointers, πρέπει πρώτα να αντιγράψουμε το struct στο device και στη συνέχεια να δημιουργήσουμε ή να αντιγράψουμε τα μέλη pointers:

```
typedef struct{
```

```

    int n;
    float *x, *y, *z;
}point;

void vecaddgpu( point *A, float* base ){
    #pragma acc parallel loop copyin(A[0:1]) \
        copyout(A->x[0:A->n], A->y[0:A->n], A->z[0:A->n]) \
        present(base[0:A->n])
    for( int i = 0; i < A->n; ++i ){
        A->x[i] = base[i];
        A->y[i] = sqrtf( base[i] );
        A->z[i] = 0;
    }
}

```

Προσοχή κατά την αντιγραφή struct που περιέχουν pointers στον host. Στο device τα μέλη με pointer ενημερώνονται με τους pointer στο device. Εάν γίνει αντιγραφή αυτών των pointer στο host, δεν θα είναι έγκυροι.

Όταν δημιουργούμε ή αντιγράφουμε μέλη struct στο device δεσμεύουμε χώρο μνήμης για όλο το struct. Δεν υποστηρίζεται δέσμευση μνήμης για μέρος του struct, ή δέσμευση μνήμης για μόνο ένα μέλος.

Τα structs και τα pointer μέλη τους διαχειρίζονται με τα δυναμικά data directive:

```

typedef struct{
    int n;
    float *x, *y, *z;
}point;

void move_to_device( point *A ){
    #pragma acc enter data copyin(A[0:1])
    #pragma acc enter data create(A->x[0:A->n], A->y[0:A->n], A->z[0:A->n])
}

void move_from_device( point* A ){
    #pragma acc enter data copyout(A->x[0:A->n], A->y[0:A->n], A->z[0:A->n])
    #pragma acc enter data delete(A[0:1])
}

void vecaddgpu( point *A, float* base ){
    #pragma acc parallel loop present(A[0:1]) \
        present(A->x[0:A->n], A->y[0:A->n], A->z[0:A->n]) \
        present(base[0:A->n])
    for( int i = 0; i < A->n; ++i ){
        A->x[i] = base[i];
        A->y[i] = sqrtf( base[i] );
        A->z[i] = 0;
    }
}

```

Nsys profiling

<https://docs.nvidia.com/nsight-systems/InstallationGuide/index.html>

Use of Linux Perf: To collect thread scheduling data and IP (instruction pointer) samples, the Linux operating system's perf_event_paranoid level must be 2 or less. Use the following command to check:

```
cat /proc/sys/kernel/perf_event_paranoid
```

If the output is >2, then do the following to temporarily adjust the paranoid level (note that this has to be done after each reboot):

```
sudo sh -c 'echo 2 >/proc/sys/kernel/perf_event_paranoid'
```

To make the change permanent, use the following command:

```
sudo sh -c 'echo kernel.perf_event_paranoid=2 > /etc/sysctl.d/local.conf'
```



Installation
Guide Nsi...

Requirements for x86_64, Power, and Arm SBSA Targets on Linux

When attaching to x86_64, Power, or Arm SBSA Linux-based target from the GUI on the host, the connection is established through SSH.

Use of Linux Perf: To collect thread scheduling data and IP (instruction pointer) samples, the Linux operating system's perf_event_paranoid level must be 2 or less. Use the following command to check:

```
cat /proc/sys/kernel/perf_event_paranoid
```

If the output is >2, then do the following to temporarily adjust the paranoid level (note that this has to be done after each reboot):

```
sudo sh -c 'echo 2 >/proc/sys/kernel/perf_event_paranoid'
```

To make the change permanent, use the following command:

```
sudo sh -c 'echo kernel.perf_event_paranoid=2 > /etc/sysctl.d/local.conf'
```

Kernel version: To collect thread scheduling data and IP (instruction pointer) samples and backtraces, the kernel version must be:

- 3.10.0-693 or later for CentOS and RedHat Enterprise Linux 7.4+
- 4.3 or greater for all other distros including Ubuntu

To check the version number of the kernel on a target device, run the following command on the device:

```
uname -a
```

```
uname -a
```

Note that only CentOS, RedHat, and Ubuntu distros are tested/confirmed to work correctly.

glibc version: To check the glibc version on a target device, run the following command:

```
ldd --version
```

Nsight Systems requires glibc 2.17 or more recent.

CUDA: See above for supported CUDA versions in this release. Use the deviceQuery command to determine the CUDA driver and runtime versions on the system. the deviceQuery command is available in the CUDA SDK. It is normally installed at:

```
/usr/local/cuda/samples/1_Uutilities/deviceQuery
```

Only pure 64-bit environments are supported. In other words, 32-bit systems or 32-bit processes running within a 64-bit environment are not supported.

Nsight Systems requires write permission to the `/var/lock` directory on the target system.

Docker: See Collecting Data within a Docker section of [Profiling in a Docker on Linux Devices](#) for more information.

gprof

Κυριακή, 20 Νοεμβρίου 2022 9:18 μμ

Δημιουργία εικόνας Call Graph με gprof2dot

<https://github.com/jrfonseca/gprof2dot>

```
/path/to/your/executable arg1 arg2
```

```
gprof path/to/your/executable | gprof2dot.py | dot -Tpng -o output.png
```


Dataset

Dataset <http://www.cs.toronto.edu/~kriz/cifar.html>

[notes]Ο Κώδικας

Δομή αρχείων

Ο φάκελος περιλαμβάνει τα παρακάτω αρχεία:

```
Makefile
layers.h
snapshot/
layers_baseline.c
test/
benchmark.c
network.c
volume.c
network.h
volume.h
huge_test.sh
network_baseline.c
layers.c
run_test.sh
```

Τα μόνα αρχεία στα οποία επεμβαίνουμε είναι:

- `layers.c`
- `network.c`
- `Volume.c`

Τα αρχεία `layers_baseline.c`, `network_baseline.c`, και `volume_baseline.c` δεν μεταβάλλονται, υπάρχουν σαν σημείο αναφοράς για μέτρηση της επιτάχυνσης.

Στο project χρησιμοποιούνται διάφοροι τύποι δεδομένων. Με τον όρο τύποι δεδομένων, εννοούμε τις δομές που με την `typedef` έχουν μετονομαστεί ώστε να έχουν νόημα. Επίσης στα header files υπάρχουν περιγραφές για το τι κάνει κάθε συνάρτηση.

Πρώτος τύπος δεδομένων `volume_t` type, περιλαμβάνει έναν τρισδιάστατο πίνακα (ή όγκο) μεταβλητών `double`. Χρησιμοποιείται για την αναπαράσταση των αποτελεσμάτων κάθε επιπέδου, καθώς και για τα βάρη κάποιων επιπέδων.

Στη συνέχεια, έχουμε τα διαφορετικά επίπεδα: `conv`, `relu`, `pool`, `fc`, and `softmax`. Το καθένα από αυτά έχει:

- Μία δομή δεδομένων που περιέχει την περιγραφή των παραμέτρων του επιπέδου. Κάθε επίπεδο έχει συγκεκριμένο αριθμό παραμέτρων που δεν αλλάζει κατά την διάρκεια της εκτέλεσης του προγράμματος. Σε αυτές τις παραμέτρους αποθηκεύονται το μέγεθος των όγκων της εισόδου και της εξόδου, και ορίζονται κατά τον ορισμό του CNN.
- Μια συνάρτηση `*_forward` η οποία εκτελεί την κύρια λειτουργία του επιπέδου. Αυτές οι συναρτήσεις αυτές παίρνουν τη δομή δεδομένων του επιπέδου, και έναν πίνακα pointers των όγκων των εισόδων και των εξόδων. Επιπλέον οι μεταβλητές `start` και `end` δείκτες σε αυτό τον πίνακα. Αυτό επιτρέπει σε κάθε επίπεδο να επεξεργάζεται μια ομάδα εισόδων. Για παράδειγμα, για την επεξεργασία των εισόδων `5,6,7,8,9` ορίζουμε `start = 5` και `end = 9`. Τα αποτελέσματα επιστρέφονται στους αντίστοιχους πίνακες εξόδων.
- Δύο συναρτήσεις, την `make_*` και την `*_load`. Η πρώτη παράγει ένα επίπεδο με συγκεκριμένο σύνολο παραμέτρων, και η δεύτερη φορτώνει τα βάρη του επιπέδου από αρχείο `.txt`.

Τελευταία σημαντική δομή δεδομένων είναι η `network_t`, η οποία περιέχει όλες τις πληροφορίες που περιγράφουν το CNN. Περιλαμβάνει όλα τα επίπεδα, και μία παράμετρο που αφορά στους διάφορους ενδιάμεσους όγκους. Να σημειωθεί πως αυτοί οι Όγκοι (Volumes) για την αποθήκευση δεδομένων (αυτό γίνεται από την δομή `batches` που περιγράφεται παρακάτω). Υπάρχουν ώστε να είναι διαθέσιμες οι διαστάσεις των διαφορετικών όγκων.

Τα δεδομένα ενδιάμεσα αναπαρίστανται ως `batches`. Περιλαμβάνουν τα ενδιάμεσα δεδομένα που σχετίζονται με ένα σύνολο εικόνων εισόδου. Το `batch_t` είναι συντομογραφία του `volume_t***`, πρόκειται δηλαδή για δισδιάστατους πίνακες δεικτών σε Volumes. Η πρώτη διάσταση δηλώνει το επιπέδου ανήκει το Volume (π.χ., V0, V1, κλπ. στο σχήμα παραπάνω) και η δεύτερη διάσταση δηλώνει την είσοδο. Στο αρχικό πρόγραμμα τα `batches` έχουν μέγεθος 1 (π.χ., επεξεργαζόμαστε ένα παράδειγμα κάθε φορά), για παραλληλοποίηση θα χρειαστούν μεγαλύτερα `batches`, τα οποία κάνουμε allocate στη μνήμη με την συνάρτηση `make_batch` και απελευθερώνουμε τη μνήμη με την `free_batch`.

Τέλος, η συνάρτηση `net_forward` δέχεται ένα batch (καθώς και δείκτες `start/end`) και πεφαρμόζει το CNN για κάθε είσοδο από την αρχή μέχρι το τέλος καλώντας τις συναρτήσεις `forward` για κάθε επίπεδο. Αυτή η συνάρτηση χρησιμοποιείται από την `net_classify` για να πάρει ένα σύνολο εικόνων, βάζει το καθένα στον όγκο V0 ενός batch που αποτελείται από ένα στοιχείο, και στη συνέχεια τρέχει το δίκτυο σε αυτά. Αποθηκεύουμε τα `likelihoods` (δηλαδή τις τιμές στο τελευταίο επίπεδο του δικτύου) σε έναν 2D πίνακα τύπου `double` ώστε να δούμε που θα ταξινομούσε το δίκτυο την εικόνα.

Κατανόηση του Κώδικα

Δομή φακέλου

Ο φάκελος περιλαμβάνει τα παρακάτω αρχεία:

```
Makefile
layers.h
snapshot/
layers_baseline.c
test/
benchmark.c
network.c
volume.c
network.h
volume.h
huge_test.sh
network_baseline.c
layers.c
run_test.sh
```

Τα μόνα αρχεία που επεμβαίνουμε είναι:

- **layers.c**
- **network.c**
- **volume.c**

Τα αρχεία `layers_baseline.c`, `network_baseline.c`, και `volume_baseline.c` περιλαμβάνουν την σειριακή έκδοση του κώδικα σαν σημείο αναφοράς για μέτρηση της επιτάχυνσης.

Επισκόπηση Κώδικα

Στον κώδικα χρησιμοποιούνται διάφοροι τύποι δεδομένων. Με τον όρο τύποι δεδομένων, εννοούμε τις δομές που με την `typedef` έχουν μετονομαστεί ώστε να έχουν νόημα. Επίσης στα header files

υπάρχουν περιγραφές για το τι κάνει κάθε συνάρτηση.

Πρώτος τύπος δεδομένων `volume_t`, περιλαμβάνει έναν τρισδιάστατο πίνακα (ή όγκο) μεταβλητών `double`. Χρησιμοποιείται για την αναπαράσταση των αποτελεσμάτων κάθε επιπέδου, καθώς και για τα βάρη κάποιων επιπέδων.

Στη συνέχεια, έχουμε τα διαφορετικά είδη επιπέδων: `conv`, `relu`, `pool`, `fc`, και `softmax`. Για το καθένα από αυτά υπάρχει:

- Μία δομή δεδομένων που περιέχει την περιγραφή των παραμέτρων του επιπέδου. Κάθε επίπεδο έχει συγκεκριμένο αριθμό παραμέτρων που δεν αλλάζει κατά την διάρκεια της εκτέλεσης του προγράμματος. Για παράδειγμα, το μέγεθος των όγκων της εισόδου και εξόδου ενός επιπέδου δηλώνεται στον ορισμό του CNN.
- Μία συνάρτηση `*_forward` οποία εκτελεί την κύρια λειτουργία του επιπέδου. Αυτές οι συναρτήσεις αυτές παίρνουν τη δομή δεδομένων του επιπέδου, και έναν πίνακα `pointers` των όγκων των εισόδων και των εξόδων. Επιπλέον οι μεταβλητές `start` και `end` δείκτες σε αυτό τον πίνακα. Αυτό επιτρέπει σε κάθε επίπεδο να επεξεργάζεται μια ομάδα εισόδων. Για παράδειγμα, μπορεί να ορίσουμε `start = 5` και `end = 9` για την επεξεργασία των εισόδων 5,6,7,8,9 ταυτόχρονα. Τα αποτελέσματα επιστρέφονται στους αντίστοιχους πίνακες εξόδων.
- Δύο συναρτήσεις, την `make_*` και `*_load`. Η πρώτη παράγει ένα επίπεδο και με συγκεκριμένο σύνολο παραμέτρων, και η δεύτερη φορτώνει τα βάρη του επιπέδου από αρχείο `.txt`.

Τελευταία σημαντική δομή δεδομένων είναι η `network_t`, η οποία περιέχει όλες τις πληροφορίες που περιγράφουν το CNN. Αυτό περιλαμβάνει όλα τα επίπεδα, και μία παράμετρο που αφορά στους διάφορους ενδιάμεσους όγκους(`volumes`). Οι όγκοι (`volumes`) δεν χρησιμοποιούνται για την αποθήκευση δεδομένων, αυτό γίνεται με τα `batches`. Χρησιμοποιούνται για να υπάρχουν διαθέσιμες οι διαστάσεις του κάθε όγκου.

Όλα τα ενδιάμεσα δεδομένα αναπαρίστανται ως `batches`. Περιλαμβάνουν τα ενδιάμεσα δεδομένα που σχετίζονται με ένα σύνολο εικόνων εισόδου. Το `batch_t` είναι συντομογραφία του `volume_t***`, πρόκειται δηλαδή για δισδιάστατους πίνακες δεικτών σε `volumes`. Η πρώτη διάσταση δηλώνει το επίπεδο που ανήκει το `volume` (π.χ., `V0`, `V1`, κλπ. στο σχήμα παραπάνω) και η δεύτερη διάσταση δηλώνει την είσοδο. Στο αρχικό πρόγραμμα τα `batches` έχουν μέγεθος 1 (π.χ., επεξεργαζόμαστε ένα παράδειγμα κάθε φορά), για παραλληλοποίηση θα χρειαστούν μεγαλύτερα `batches`, τα οποία κάνουμε `allocate` στη μνήμη με την συνάρτηση `make_batch` και απελευθερώνουμε από τη μνήμη με την `free_batch`.

Τέλος, η συνάρτηση `net_forward` δέχεται ένα `batch` (καθώς και δείκτες `start/end`) και εφαρμόζει το CNN για κάθε είσοδο από την αρχή μέχρι το τέλος καλώντας τις συναρτήσεις `forward` για κάθε επίπεδο. Αυτή η συνάρτηση χρησιμοποιείται από την `net_classify`. Παίρνει ένα σύνολο εικόνων, βάζει κάθε μία από αυτές στο `volume V0` ενός στοιχείου του `batch`, και στη συνέχεια τρέχει το δίκτυο σε αυτά. Αποθηκεύουμε τα `likelihoods` (δηλαδή τις τιμές στο τελευταίο επίπεδο του δικτύου) σε έναν 2D πίνακα τύπου `double` ώστε να δούμε που θα ταξινομούσε το δίκτυο την εικόνα.

Συμπεράσματα

Μελλοντικές Επεκτάσεις

Links

UC Berkeley, CS 61C Spring 2019 - Project 4 <https://inst.eecs.berkeley.edu/~cs61c/sp19/projects/proj4/>

UC Berkeley. CS 61C Project 4: Performance Programming.
<https://inst.eecs.berkeley.edu/cs61c/sp19/projects/proj4/>, 2019

Stanford University, CS231n Deep Learning for Computer Vision <http://cs231n.stanford.edu/https://stanford.edu/~shervine/teaching/cs-230/>

Udacity, cs344 "Intro to Parallel Programming" <https://classroom.udacity.com/courses/cs344>
[<https://www.udacity.com/wiki/cs344>]

Lawrence Livermore National Laboratory, Introduction to Parallel Computing Tutorial
<https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial>

Michael Nielsen, Neural Networks and Deep Learning,
<http://neuralnetworksanddeeplearning.com/chap1.html>

<https://www.cs.toronto.edu/~kriz/>

Convolutional Neural Networks for Object Classification

Alex Krizhevsky

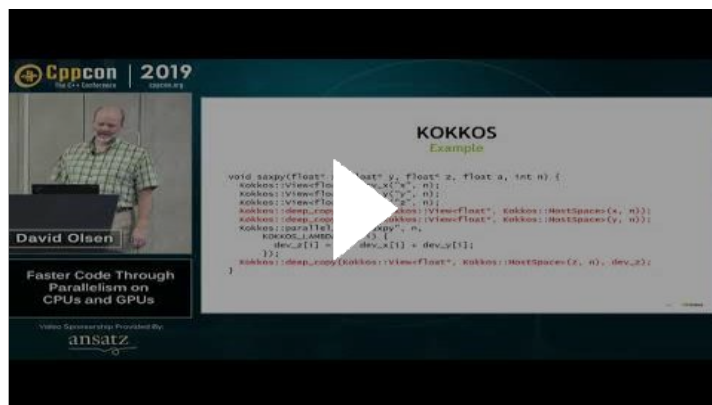
http://www.eecg.toronto.edu/~moshovos/CUDA08/arx/convnet_report.pdf

<https://paperswithcode.com/paper/benchmarking-state-of-the-art-deep-learning>

<https://ai.googleblog.com/2021/09/toward-fast-and-accurate-neural.html>

<https://course.fast.ai/>

[CppCon 2019: David Olsen "Faster Code Through Parallelism on CPUs and GPUs"](#)



Mathworks, "Convolutional Neural Network." <https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html>.

CS231n, "Convolutional Neural Networks for Visual Recognition." <http://cs231n.github.io>.

Courses

[Heterogeneous Parallel Programming \(University of Illinois at Urbana-Champaign\)](#)

[Intro to Parallel Programming](#) (Udacity-olia92@live.com)



Books

Rob Faber. Parallel Programming with OpenACC. Morgan Kaufmann, 2017.

https://github.com/OpenACCUserGroup/openacc_concept_strategies_book

Juckeland, Guido & Chandrasekaran, Sunita. (2017). OpenACC for Programmers: Concepts and Strategies.

Programming Massively Parallel Processors. DOI: <http://dx.doi.org/10.1016/B978-0-12-811986-0.00001-7>

Copyright ©2017 David B. Kirk/NVIDIA Corporation and Wen-mei W. Hwu. Published by Elsevier Inc. All rights reserved

Parallel and High Performance Computing. Robert Robey, Yuliana Zamora. Manning

Papers

<http://yann.lecun.com/exdb/publis/pdf/lecun-89e.pdf>

Alex Krizhevsky, [Learning Multiple Layers of Features from Tiny Images](#), 2009.

E. Cengil, A. Çinar and Z. Güler, "[A GPU-based convolutional neural network approach for image classification](#)," 2017 International Artificial Intelligence and Data Processing Symposium (IDAP), 2017, pp. 1-6, doi: 10.1109/IDAP.2017.8090194.

<https://arxiv.org/abs/1711.03278>

Application of the residue number system to reduce hardware costs of the convolutional neural network implementation
<https://www.sciencedirect.com/science/article/pii/S0378475420301580>

Krizhevsky, A. [Convolutional Deep Belief Networks on CIFAR-10](#)

Krizhevsky, A, [Convolutional Neural Networks for Object Classification in CUDA](#)

Krizhevsky, ImageNet Classification with Deep Convolutional Neural Networks,
<http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf>

Benchmarking State-of-the-Art Deep Learning Software Tools <https://arxiv.org/pdf/1608.07249v7.pdf> ,
<https://paperswithcode.com/paper/benchmarking-state-of-the-art-deep-learning>

Performance Analysis of GPU-based Convolutional Neural Networks
<https://www2.seas.gwu.edu/~howie/publications/GPU-CNN-ICPP16.pdf>

Github

https://github.com/luweiagi/CNN_LeNet

Άλλες Διπλωματικές

E.Chatzidaki, A **CNN** framework accelerated by an FPGA in synthesized C

<https://dias.library.tuc.gr/view/83856>

Σ.Ρήγα, Υλοποίηση συστήματος μηχανικής μάθησης για συστάσεις ταινιών με παράλληλο

προγραμματισμό σε **OpenACC** και αξιολόγηση της επιτάχυνσής του σε βάσεις δεδομένων υψηλού
όγκου <https://nemertes.library.upatras.gr/jspui//handle/10889/12867>

Dataset

CIFAR-10 <https://www.cs.toronto.edu/~kriz/cifar.html>

Fashion mnist <https://github.com/zalandoresearch/fashion-mnist>

For Citations

<https://www.sciencedirect.com/book/9780128119860/programming-massively-parallel-processors>

<https://www.researchgate.net/publication/319979709>

[OpenACC for Programmers Concepts and Strategies/citation/download](#)

<https://www.mendeley.com/search/>

Cuda-gdb

3.3. Using the Debugger

CUDA-GDB can be used in the following system configurations:

3.3.1. Single-GPU Debugging with the Desktop Manager Running

For devices with compute capability 6.0 and higher CUDA-GDB can be used to debug CUDA applications on the same GPU that is running the desktop GUI.

Additionally for devices with compute capability less than 6.0 software preemption can be used to debug CUDA applications on the same GPU that is running the desktop GUI. There are two ways to enable this functionality:

Note: This is a BETA feature available on Linux and is only supported on Maxwell. The options listed below are ignored for GPUs with SM6.0 compute capability and higher.

- Use the following command:
`set cuda software_preemption on`
- Export the following environment variable:
`CUDA_DEBUGGER_SOFTWARE_PREEMPTION=1`

Either of the options above will activate software preemption. These options must be set **prior** to running the application. When the GPU hits a breakpoint or any other event that would normally cause the GPU to freeze, CUDA-GDB releases the GPU for use by the desktop or other applications. This enables CUDA-GDB to debug a CUDA application on the same GPU that is running the desktop GUI, and also enables debugging of multiple CUDA applications context-switching on the same GPU.

```
$ set cuda software_preemption on
$ export CUDA_DEBUGGER_SOFTWARE_PREEMPTION=1
$ printenv CUDA_DEBUGGER_SOFTWARE_PREEMPTION
```

Nvc Compiler

```
$ nvc --help -Minfo
```

```
-M[no]info[=all|accel|ftn|inline|intensity|ipa|loop|lre|mp|opt|par|pcast|pfo|stat|time|vect|stdpar]
```

Generate informational messages about optimizations

all -Minfo=accel,inline,ipa,loop,lre,mp,opt,par,vect,stdpar

accel Enable Accelerator information

ftn Enable Fortran-specific information

inline Enable inliner information

intensity Enable compute intensity information

ipa Enable IPA information

loop Enable loop optimization information

lre Enable LRE information

mp Enable OpenMP information

opt Enable optimizer information

par Enable parallelizer information

pcast Enable PCAST information

pfo Enable profile feedback information

stat Same as -Minfo=time

time Display time spent in compiler phases

vect Enable vectorizer information

stdpar Enable stdpar information

-Minfo=all,pfo αντίστοιχο με -Minfo=all,ccff στον pgcc

This compiler flag instructs the compiler to print information about how it optimized the code and additionally embed this information in the executable for use by tools that support common compiler feedback format

Ubuntu: restore nvidia graphics driver after update

Link: <https://hih-v-104.neurologie.uni-tuebingen.de/ofork/public.pl?Action=PublicFAQZoom;ItemID=100>

FAQ ► Software ► Linux ► Ubuntu: restore nvidia graphics driver after update

To open links in the following description blocks, you might need to press Ctrl or Cmd or Shift key while clicking the link (depending on your browser and OS).

Symptom:

Problem:

This can help, if the gui fails after an ubuntu update.

Solution:

Ctrl-Alt-F1 -> Console

```
sudo apt-get purge nvidia*
sudo apt-get install nvidia-current-updates-dev
/etc/init.d/lightdm restart
```

Ctrl-Alt-F1 -> Console

```
sudo apt-get purge nvidia*
sudo apt-get install nvidia-current-updates-dev
/etc/init.d/lightdm restart
```

Ctrl-Alt-F1 -> Console

```
sudo apt-get purge nvidia*
sudo apt-get install nvidia-driver-470
reboot
```

codes

Thursday, August 11, 2022 4:57 PM

<https://github.com/fan-wenjie/LeNet-5>

Overleaf

Παρασκευή, 12 Αυγούστου 2022 12:24 μμ

https://www.overleaf.com/learn/latex/Multilingual_typesetting_on_Overleaf_using_babel_and_fontspec

```
\usepackage[english,russian,main=french]{babel}

\begin{otherlanguage}{russian}
Lorem Ipsum - это текст-`\textsf{рыба}', часто используемый в
\texttt{печати} и вэб-дизайне. ...
\end{otherlanguage}
```

https://www.overleaf.com/learn/latex/Code_Highlighting_with_minted

```
\usepackage{minted}

\begin{minted}{python}
import numpy as np

def incmatrix(genl1,genl2):
[...]
    return M
\end{minted}
```

C

Σάββατο, 13 Αυγούστου 2022 12:52 μμ

<http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines>
<https://www.open-std.org/jtc1/sc22/wg14/www/docs/n1548.pdf>

Test Create on Device

make_conv_layer

```
40 | l->bias = 0.0;
41 | #pragma acc enter data copyin(l->bias)
42 |
43 | l->filters = malloc(sizeof(volume_t) * num_filters);
44 | // #pragma acc enter data create(l->filters[0:num_filters])
45 | for (int i = 0; i < num_filters; i++) {
46 |     l->filters[i] = make_volume(l->filter_width, l->filter_height,
47 |                               l->input_depth, 0.0); //TEST:0.0 original:0.0
48 | }
49 |
50 | //---> Host Data
51 | fdump_volume(l->filters[15], "/output/convl_filters.txt");
52 | //---> Create and Copy data to Device
53 | #pragma acc enter data copyin(l->filters[0:num_filters])
54 | //---> Copy data to Host
55 | for (int i = 0; i < num_filters; i++) {
56 |     int we=l->filter_width*l->filter_height*l->input_depth;
57 |     #pragma acc update self(l->filters[i]->weights[0:we])
58 |     #pragma acc update self(l->filters[i]->width, l->filters[i]->height, l->filters[i]->depth)
59 | }
60 | fdump_volume(l->filters[15], "/output/convl_filters2.txt");
61 |
62 | l->biases = make_volume(1, 1, l->output_depth, l->bias);
63 | }
```

Make Conv Layer

41: αντιγράφει όλες τις παραμέτρους στη συσκευή

46: η make_volume δημιουργεί τα filters στη συσκευή

Βάζει σε όλα την τιμή 0.0

50: τυπώνουμε τιμές fdump --> πρέπει να βγάζει 0.0 από τον host

52: data copyin(l->filters[0:num_filters]) για αντιγραφή δεδομένων στο device

54-58: αντιγραφή data στον host . Για κάθε filter ξεχωριστά τα weights[0:we], αλλά και κάθε struct member ξεχωριστά.

59: fdump_volume τυπώνουμε τα filters από το device, πρέπει να έχουν την τιμή 8.0

Make_volume():

```
34 | new_vol->height = height;
35 | new_vol->depth = depth;
36 | #pragma acc enter data copyin(new_vol->bias)
37 | //--->Paralle create and calculate on device
38 | #pragma acc enter data create(new_vol->weights[0:(width * height * depth)])
39 | #pragma acc parallel loop collapse(3) present(new_vol)
40 | for (int x = 0; x < width; x++) {
41 |     for (int y = 0; y < height; y++) {
42 |         for (int d = 0; d < depth; d++) {
43 |             volume_set(new_vol, x, y, d, value);
44 |         }
45 |     }
46 | }
47 | // #pragma acc enter data copyin(new_vol->weights[0:(width * height * depth)])
48 |
49 | return new_vol;
50 | }
```

Make Volume

35: δημιουργία και αντιγραφή του struct new_vol στο device.

37: δημιουργία για τα weights ξεχωριστά

38-45: κομμάτι κώδικα που θα τρέξει παράλληλα στη συσκευή, collapse στα 3 loop, και πρέπει να δηλώσουμε ότι το new_vol βρήσκεται στη συσκευή --> present(new_vol)

**volume_set είναι δηλωμένη με #pragma acc routine seq τρέχει σειριακά στο thread

```
//--->Paralle create and calculate on device
// #pragma acc enter data create(new_vol->weights[0:(width * height * depth)])
// #pragma acc parallel loop collapse(3) present(new_vol)
```

//---> Host Data

```
for (int i = 0; i < num_filters; i++) {
    for (int x = 0; x < l->filter_height; x++) {
        for (int y = 0; y < l->input_depth; y++) {
            for (int d = 0; d < l->input_depth; d++) {
                volume_set(l->filters[i], x, y, d, 3.0);
            }
        }
    }
}

fdump_volume(l->filters[15], "/output/convl_filters.txt");
//---> Create and Copy data to Device
// #pragma acc enter data copyin(l->filters[0:num_filters])
//---> Copy data to Host
for (int i = 0; i < num_filters; i++) {
    int we=l->filter_width*l->filter_height*l->input_depth;
    #pragma acc update self(l->filters[i]->weights[0:we])
    #pragma acc update self(l->filters[i]->width, l->filters[i]->height, l->filters[i]->depth)
}

fdump_volume(l->filters[15], "/output/convl_filters2.txt");
```

Volumes are used to represent the activations (i.e., state) between the different layers of the CNN. They all have three dimensions. The interpretation of their content depends on the layer that produced them. Before the first iteration, the Volume holds the data of the image we want to classify (the depth are the three color dimensions). After the last stage of the CNN, the Volume holds the probabilities that an image is part of a specific category.

```
typedef struct volume {  
    int width;  
    int height;  
    int depth;  
    double *weights;  
} volume_t;
```