Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών
Α.Π.Θ.

1/3/2017

Παράλληλα και Κατανεμημένα Συστήματα Υπολογιστών

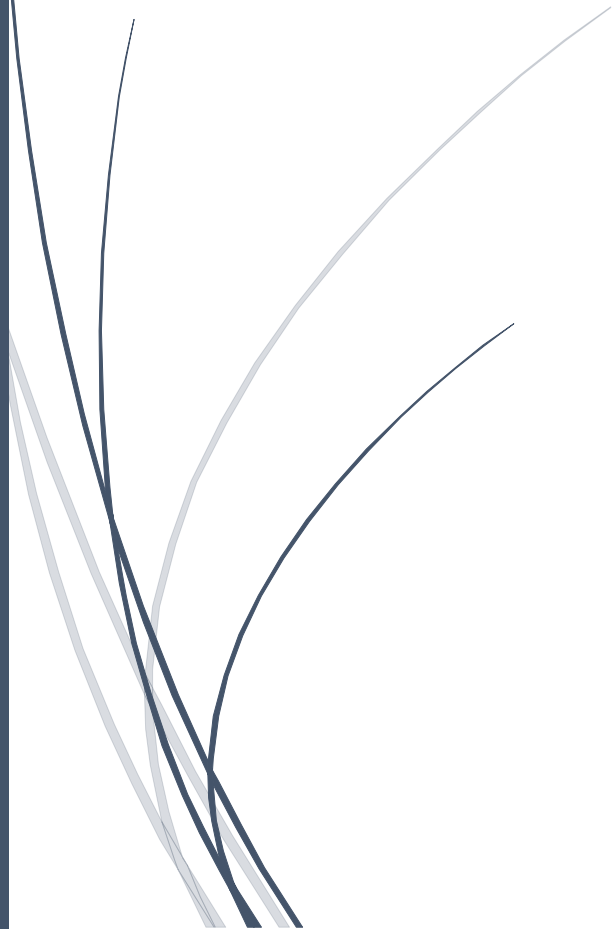
Εργασία 4

Διαμαντή Μαρία 8133

mfdiamanti@ece.auth.gr

Ντζιώνη Δήμητρα 8209

dntzioni@ece.auth.gr



Πίνακας περιεχομένων

1. Εισαγωγή

Το ζητούμενο της τέταρτης εργασίας στο μάθημα των Παράλληλων και Κατανεμημένων Συστημάτων Υπολογιστών ήταν η υλοποίηση του αλγορίθμου backpropagation για την εκπαίδευση ενός νευρωνικού δικτύου παράλληλα. Αρχικά, προτού φτάσουμε στο σημείο της συγγραφής του κώδικα που έπρεπε να παραλληλοποιήσουμε, αναγκαία ήταν η δημιουργία ενός ολοκληρωμένου νευρωνικού δικτύου και η συλλογή των δεδομένων εισόδου και εξόδου για τα οποία θα εκπαιδευτεί αυτό. Τα δεδομένα αυτά αξιοποιούνται μέσω του αλγορίθμου backpropagation σε συνδυασμό με τον αλγόριθμο εκμάθησης stochastic gradient descent, όπως αυτοί αναλύονται στα κεφάλαια 1 και 2 του βιβλίου του Michael Nielsen, το οποίο μας δόθηκε προς μελέτη. Προκειμένου να εκπαιδεύσουμε το δίκτυό μας ως προς δεδομένα υπαρκτά και να μπορέσουμε να εξάγουμε ευκολότερα το συμπέρασμα πως έχει εκπαιδευτεί, αποφασίσαμε να επιλύσουμε το πρόβλημα της αναγνώρισης χειρόγραφων αριθμών από το 0 έως το 9, όπως αυτό παρουσιάζεται στο κεφάλαιο 1. Για το λόγο αυτό, χρησιμοποιήσαμε το σύνολο δεδομένων της MNIST όπως προτεινόταν, η οποία περιέχει δεκάδες χιλιάδες σαρωμένες εικόνες χειρόγραφων ψηφίων, μαζί με τις επιθυμητές εξόδους που πρέπει να φέρει το δίκτυο έπειτα από τη διαδικασία εκμάθησης.

Πιο αναλυτικά, ο αλγόριθμος που υλοποιήσαμε διακρίνεται σε πολλαπλά στάδια. Κατά το πρώτο στάδιο (Feedforward), τα δεδομένα εισόδου που παρουσιάζονται στο πρώτο επίπεδο του δικτύου (layer), προωθούνται προς το τελευταίο, διασχίζοντας ένα προς ένα τα επίπεδά του. Έτσι δημιουργείται ο πίνακας της σταθμισμένης εισόδου z των νευρώνων για κάθε επίπεδο. Αφού και το τελευταίο επίπεδο έχει στη διάθεσή του τον πίνακα αυτό και μετά από κατάλληλους υπολογισμούς, συγκρίνεται η έξοδος του δικτύου με την επιθυμητή έξοδο και προκύπτει ένα σφάλμα δ. Αυτή η σύγκριση πραγματοποιείται για κάθε ένα νευρώνα του τελευταίου επιπέδου και αποτελεί το στάδιο του Output Error. Έπειτα, κατά το Error Backpropagation, προχωρώντας από το τέλος προς την αρχή, με όμοιο τρόπο υπολογίζουμε τον πίνακα σφάλματος δ για κάθε επίπεδο λαμβάνοντας υπόψιν τους πίνακες βαρών w και σφάλματος δ του αμέσως επόμενου επιπέδου. Ο πίνακας σφάλματος δ είναι χρήσιμος στο τελικό στάδιο του αλγορίθμου (Gradient Descent), όπου πραγματοποιείται η ανανέωση των βαρών και των biases κάθε επιπέδου του δικτύου. Τα παραπάνω στάδια επαναλαμβάνονται για πολλαπλά epochs και διαφορετικά training examples έως ότου το συνολικό σφάλμα του αλγορίθμου φτάσει σε μια ικανοποιητική τιμή που έχει ορισθεί από εμάς και υποδηλώνει πως το δίκτυο έχει εκπαιδευτεί.

2. Δημιουργία Νευρωνικού Δικτύου

Αρχικό μέλημά μας ήταν η δημιουργία ενός νευρωνικού δικτύου. Στο σημείο αυτό, λοιπόν, αξίζει να αναφερθούν κάποιες σημαντικές παράμετροι που το απαρτίζουν και ο τρόπος με τον οποίο επιλέχθηκαν αυτές από εμάς. Ο αλγόριθμός μας υλοποιήθηκε δυναμικά ως προς τον αριθμό των layers του δικτύου αλλά και το πλήθος των νευρώνων καθενός layer, επιτρέποντας στο χρήστη να δώσει τις τιμές που επιθυμεί. Συγκεκριμένα, παρέχεται η δυνατότητα εισαγωγής του πλήθους των νευρώνων καθενός layer από αρχείο, σε περίπτωση που τα layers είναι πολλά σε αριθμό. Επιπλέον, για την εισαγωγή των απαραίτητων δεδομένων εκμάθησης που συλλέξαμε από την ιστοσελίδα της MNIST στον κώδικά μας, δημιουργήσαμε τα κατάλληλα αρχεία από τα οποία γίνεται ανάγνωση των δεδομένων, καθώς επίσης και ένα struct, το struct DATA. Το struct αυτό περιέχει τα δεδομένα εισόδου και την επιθυμητή έξοδο του δικτύου για ένα training example. Περισσότερες λεπτομέρειες για το είδος των δεδομένων που παρέχει η MNIST και για τον τρόπο εξαγωγής αυτών από την ιστοσελίδα της αναφέρονται στην παράγραφο 6 που ακολουθεί.

Όσον αφορά τους πίνακες weights, bias, z, delta και activation που είναι απαραίτητοι, όπως ορίζονται από τις εξισώσεις για την υλοποίηση του αλγορίθμου, είναι δισδιάστατοι με την πρώτη τους διάσταση να προσδιορίζει το layer στο οποίο αναφέρονται. Ειδικότερα για τον πίνακα weights, η δεύτερή του διάσταση αποτελεί ένα δείκτη σε ένα δισδιάστατο πίνακα με γραμμές όσες το πλήθος των νευρώνων του layer που υποδεικνύει η πρώτη διάσταση και στήλες όσες το πλήθος των νευρώνων του ακριβώς προηγούμενου layer ως εξής [i*M + j] όπου M = πλήθος νευρώνων του προηγούμενου layer. Να σημειωθεί, επίσης, πως οι πίνακες weights, bias, z και delta δεν υπάρχουν για το input layer, αλλά μόνο ο πίνακας activation.

Πίνακας weights: Αρχικοποιείται με τυχαίες τιμές που έχουν μέση τιμή 0 και απόκλιση r, όπου r = sqrt(1.0 / (double)(neuronsOfLayer[0])) σύμφωνα με ποικίλες αναφορές στο ίντερνετ και έπειτα από δικές μας δοκιμές.

Πίνακας bias: Αρχικοποιείται με τυχαίες τιμές που έχουν μέση τιμή 0 και απόκλιση 1.

Ο τρόπος χρήσης των παραπάνω πινάκων ορίζεται στο link που μας δόθηκε προς μελέτη.

Όπως προαναφέρθηκε, τα στάδια του αλγορίθμου επαναλαμβάνονται για πολλαπλά epochs και διαφορετικά training examples, τα οποία αφορούν στις επαναλήψεις του κύκλου εκμάθησης και στο πλήθος των διαφορετικών δεδομένων εισόδου. Οι τιμές των παραμέτρων αυτών για τις οποίες κάναμε δοκιμές παρουσιάζονται στην ενότητα 6. Ο αλγόριθμος τερματίζεται είτε όταν ολοκληρωθούν οι παραπάνω επαναλήψεις, είτε όταν το συνολικό σφάλμα του δικτύου γίνει μικρότερο της τιμής 0,045. Η τιμή αυτή προκύπτει από τη συνάρτηση κόστους C=0.5*‖y−aL‖^2 ως εξής. Σε περίπτωση που εισάγουμε, για παράδειγμα, την εικόνα με το νούμερο 9 στο δίκτυο, μας αρκεί ο ένατος νευρώνας να φέρει έξοδο μεγαλύτερη ή ίση του 0.7 και οι υπόλοιποι να φέρουν έξοδο μικρότερη ή ίση του 0.3.

3. Pthreads

Το τμήμα του σειριακού κώδικα που παραλληλοποιήθηκε μέσω της χρήσης των pthreads είναι αυτό της αρχικοποίησης των πινάκων weights και bias με τυχαίες τιμές. Συγκεκριμένα, αναπτύξαμε έναν υβριδικό τρόπο παραλληλοποίησης όπου συνδυάσαμε τη χρήση CUDA - pThreads. Αυτό είναι εφικτό σε αυτό το σημείο διότι δεν υπάρχει καμία αλληλεξάρτηση μεταξύ των επιπέδων του δικτύου. Έτσι, παραλληλοποιήθηκε με pThreads ως προς κάθε επίπεδο και με CUDA ως προς κάθε νευρώνα του εκάστοτε επιπέδου. Όσον αφορά τα pThreads , για την αρχικοποίηση των πινάκων weights και bias, καλείται η συνάρτηση generate_data() μέσα στην οποία με τη βοήθεια της pthread_create() δημιουργούνται τόσα νήματα όσα ορίζει ο χρήστης και τα οποία με τη σειρά τους εκτελούν τη συνάρτηση generate_data_parallel(). Τα ορίσματα που πρέπει να περαστούν στη συνάρτηση generate_data_parallel() περνιούνται μέσω του struct generate_data_threads, Ο τρόπος με τον οποίο δημιουργείται ένας πίνακας νημάτων, ορίζεται το attribute τους και ο τρόπος με τον οποίο περνιούνται οι κατάλληλες τιμές μέσα στο struct είναι φανερός στον κώδικα. Προχωρώντας, μέσα στη συνάρτηση generate_data_parallel() κάθε ένα νήμα καλεί το CUDA kernel cudaGenerator, όπου πραγματοποιείται ουσιαστικά η αρχικοποίηση των πινάκων παράλληλα για κάθε ένα νευρώνα. Η λογική «per-thread default stream» που παρουσιάζεται στην ιστοσελίδα της NVIDIA είναι η εξής. Σε κάθε νήμα αντιστοιχεί ένα ξεχωριστό default stream και με αυτόν τον τρόπο αποφεύγεται η σειριακή εκτέλεση του κώδικα καθενός νήματος από το legacy default stream. Η λογική της συγγραφής του CUDA kernel είναι η ήδη γνωστή και αναλύεται στην επόμενη ενότητα. Το μόνο που πρέπει να προστεθεί στον κώδικα για τον υβριδικό τρόπο παραλληλοποίησης είναι το #define CUDA_API_PER_THREAD_DEFAULT_STREAM κάτω από το header <cuda.h> για να είναι εφικτή η μεταγλώττιση.

4. CUDA

Με χρήση της CUDA παραλληλοποιήσαμε με παρόμοιο τρόπο πέντε διαφορετικά αρχεία του κώδικα, καθένα από τα οποία υλοποιεί και ένα διαφορετικό στάδιο του αλγορίθμου. Αυτά είναι τα εξής: functions.c, feedforward.c, outputerrorcalculation.c, backpropagation.c, updateweights.c. Η παραλληλοποίηση πραγματοποιείται για τους νευρώνες ενός layer. Προτού καλέσουμε το εκάστοτε CUDA kernel κάνουμε τις κατάλληλες δεσμεύσεις μνήμης στη GPU προκειμένου να περάσουμε τα απαραίτητα δεδομένα από τη CPU σε αυτήν. Το παραπάνω υλοποιείται μέσω της χρήσης των συναρτήσεων cudaMalloc() και cudaMemcpy(). Μετά την ολοκλήρωση των υπολογισμών στο CUDA kernel, αντιγράφουμε τα επιθυμητά αποτελέσματα των υπολογισμών πίσω στη CPU και έπειτα με τη χρήση της cudaFree() επιτρέπουμε στη GPU να αξιοποιήσει τη μνήμη που δεσμεύσαμε προηγουμένως για κάποιο άλλο σκοπό. Αξίζει να αναφέρουμε ότι με την παραλληλοποίησή μας δημιουργούμε ένα μπλοκ με πολλαπλά νήματα, όσα είναι και το πλήθος των νευρώνων του καθενός layer.

Η επιλογή μας βασίστηκε στο γεγονός ότι ο αριθμός των νευρώνων καθενός layer είναι αρκετά μικρός και δεν υπήρχε πιθανότητα αδυναμίας από το σύστημα να δημιουργήσει τόσα νήματα. Σε άλλη περίπτωση, ο αλγόριθμός μας έχει υλοποιηθεί με τέτοιο τρόπο ώστε να είναι εύκολο να αλλάζει το πλήθος των μπλοκ και των νημάτων. Καταληκτικά, ο λόγος για τον οποίο αξιοποιήθηκε η CUDA εκτενώς είναι το γεγονός πως η GPU είναι σημαντικά γρηγορότερη από τη CPU. Επιπλέον, έπειτα από παλαιότερες δοκιμές μας στο server Διάδη αποδείχθηκε ότι ο ιδανικός αριθμός νημάτων για βέλτιστη απόδοση είναι 8, άρα στη δική μας περίπτωση η χρήση pThreads δε συνιστούσε την κατάλληλη επιλογή.

5. Συλλογή δεδομένων

Προτού παρουσιάσουμε τα αποτελέσματα που εξήγαμε για διαφορετικά μεγέθη δεδομένων εισόδου, αξίζει να αναφερθεί ο τρόπος με τον οποίο αξιοποιήθηκαν τα data sets που παρέχει η MNIST στο ακόλουθο link: http://yann.lecun.com/exdb/mnist/. Για διευκόλυνση, χρησιμοποιήσαμε δύο συναρτήσεις του MATLAB που είναι διαθέσιμες στο ίντερνετ για να εξάγουμε τα δεδομένα των πινάκων train-images-idx3-ubyte και train-labels-idx1-ubyte. Ο 784*60000 train-images-idx3-ubyte πίνακας περιέχει 60000 training examples σε μορφή εικόνων διάστασης 28*28 (κάθε γραμμή του αποτελεί και μια εικόνα χειρόγραφου αριθμού). Από τα 60000 δείγματα εξήγαμε σε δικό μας αρχείο με όνομα input10000x784float.txt τα 10000 από αυτά. Αντίστοιχα, από τον 60000*1 train-labels-idx1-ubyte πίνακα, που περιλαμβάνει τις επιθυμητές εξόδους του δικτύου, εξήγαμε τα πρώτα 10000 στοιχεία σε δικό μας αρχείο με όνομα output10000.txt. Τα αρχεία αυτά βρίσκονται στο φάκελο data. Συνεπώς, προκειμένου να εκτελέσουμε τον αλγόριθμό μας για διαφορετικά μεγέθη δεδομένων εισόδου κάθε φορά διαβάζουμε μέσα στον κώδικα το πλήθος των δεδομένων που επιθυμούμε από τα αρχεία αυτά. Τέλος, η ιστοσελίδα της MNIST παρέχει και ένα πλήθος test data, τα οποία μπορούν να χρησιμοποιηθούν έπειτα για testing ώστε να διαπιστωθεί ότι το δίκτυο εκπαιδεύεται, χρησιμοποιώντας τις τιμές των weights και biases για τις οποίες τερματίστηκε ο αλγόριθμος.

6. Αποτελέσματα - Συμπεράσματα

Τα αποτελέσματα που εξήγαμε για διαφορετικά μεγέθη δεδομένων εισόδου, δηλαδή για διαφορετικό πλήθος training examples παρουσιάζονται παρακάτω. Όπως είναι φανερό από τον πίνακα, το learning rate η, το πλήθος των epochs(ξεκινώντας να μετράμε από το 0 epoch), ο αριθμός των layers, το πλήθος των νευρώνων κάθε layer, καθώς και ο αριθμός των νημάτων παραμένει σταθερός καθ’ όλη τη διάρκεια των δοκιμών. Οι τιμές αυτές προέκυψαν έπειτα από κάποιες δοκιμές έτσι ώστε τα αποτελέσματα να είναι ικανοποιητικά.

Number of training examples	Final error	Total time (to reach this error) (s)	Exit epoch
Number of hidden layers: 3, Number of neurons per hidden 100, Number of epochs: 30, eta: 0.3			
1	0.029772	0.032747	1st
10	0.044459	0.887378	5th
100	0.044973	14.922428	10th
250	0.044991	47.418644	12th
500	0.044985	106.698384	14th
750	0.044989	174.077217	16th
1000	0.045000	268.581558	17th
2500	0.044999	547.106539	18th
5000	0.044995	1187.361194	19th
7500	0.044998	1726.888490	20th
10000	0.044999	2093.860421	18th

Οι παραπάνω χρόνοι όπως είναι φανερό αντιπροσωπεύουν τη χρονική στιγμή που αλγόριθμος φτάνει να έχει συνολικό σφάλμα μικρότερο ή ίσο του 0.045.

Όπως είναι λογικό, όσο μεγαλώνει το πλήθος των δεδομένων εισόδου στο δίκτυο, ο χρόνος εκμάθησης αυξάνεται, λόγω της εκτέλεσης πολύ περισσότερων υπολογισμών. Παρόλη την καθυστέρηση στο χρόνο εκμάθησης, για περισσότερα δεδομένα εισόδου το δίκτυο εκπαιδεύεται καλύτερα καθώς μπορεί να αναγνωρίσει περισσότερες περιπτώσεις χειρόγραφων ψηφίων. Επιπλέον, όσο η συχνότητα εμφάνισης των ψηφίων σε ένα συγκεκριμένο δείγμα μεγαλώνει, τόσο λιγότεροι κύκλοι εκμάθησης απαιτούνται (epochs). Αυτός είναι και ο λόγος που στην περίπτωση των 10000 training examples το exit epoch είναι μικρότερο από ότι στις προηγούμενες δύο περιπτώσεις των 5000 και 7500 training examples. Τέλος, όπως εύκολα παρατηρείται, όσα περισσότερα δεδομένα εισόδου θέτουμε τόσο το τελικό σφάλμα πλησιάζει στην τιμή 0.045, σε αντίθεση με την περίπτωση που εισάγουμε ένα μόνο training example.

7. Αναφορές

http://neuralnetworksanddeeplearning.com/chap1.html#implementing_our_network_to_classify_digits

http://neuralnetworksanddeeplearning.com/chap2.html

https://devblogs.nvidia.com/parallelforall/gpu-pro-tip-cuda-7-streams-simplify-concurrency/

http://cs.umw.edu/~finlayson/class/fall16/cpsc425/notes/cuda-random.html

http://yann.lecun.com/exdb/mnist/

http://ufldl.stanford.edu/wiki/index.php/MATLAB_Modules

https://en.wikibooks.org/wiki/Artificial_Neural_Networks/Neural_Network_Basics

http://machinelearningmastery.com/improve-deep-learning-performance/

https://devblogs.nvidia.com/parallelforall/separate-compilation-linking-cuda-device-code/

https://devtalk.nvidia.com/default/topic/546584/passing-two-nested-structures-including-pointer-of-pointers-at-the-child-structure-/?offset=2

https://www.quora.com/How-do-I-decide-the-number-of-nodes-in-a-hidden-layer-of-a-neural-network

http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

http://staff.itee.uq.edu.au/janetw/cmc/chapters/BackProp/index2.html