ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Παράλληλες Αρχιτεκτονικές Υπολογισμού για Μηχανική Μάθηση

Ε.ΔΕ.Μ^2

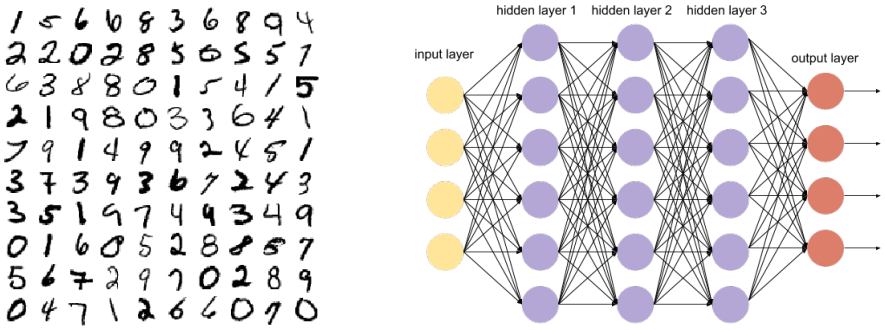
Ακαδημαϊκό Έτος 2020-21

http://www.cslab.ece.ntua.gr/courses/parml

ΕΡΓΑΣΙΑ
Επιτάχυνση εκπαίδευσης νευρωνικού δικτύου σε αρχιτεκτονικές κοινής μνήμης με OpenMP και CUDA

1 Σκοπός της Εργασίας

Στα πλαίσια αυτής της εργασίας θα εξοικειωθείτε με τον παράλληλο προγραμματισμό σε αρχιτεκτονικές κοινής μνήμης μέσα από την επίλυση ενός προβλήματος μηχανικής μάθησης. Συγκεκριμένα, θα καταπιαστείτε με την αναγνώριση χειρόγραφων αριθμητικών ψηφίων και θα εκπαιδεύσετε νευρωνικό δίκτυο για την επίλυσή του. Η εκπαίδευση θα γίνει με χρήση CPU και GPU. Ως δεδομένα εκπαίδευσης για το μοντέλο σας θα χρησιμοποιήσετε τη βάση δεδομένων MNIST, η οποία περιλαμβάνει 70000 εικόνες χειρόγραφων αριθμητικών ψηφίων με 784 χαρακτηριστικά. Η αρχιτεκτονική του μοντέλου που θα εκπαιδεύσετε δίνεται στο Σχήμα 1.



Σχήμα 1: Χειρόγραφα αριθμητικά ψηφία (αριστερά), νευρωνικό δίκτυο με 3 κρυφά επίπεδα (δεξιά)

Ο σκελετός της εργασίας βρίσκεται στον κατάλογο `/home/parml/shared/ex1_omp_cuda`. Η μεταγλώττιση και εκτέλεση των προγραμμάτων θα γίνεται στο μηχάνημα `termi1`, που ανήκει στην ουρά `termis` (δείτε και υποενότητα 3.2).

2 Ζητούμενα

Στα πλαίσια αυτής της εργασίας θα καταπιαστείτε με την παραλληλοποίηση του αλγορίθμου πολλαπλασιασμού πινάκων (GEneral Matrix Multiply, GEMM), ο οποίος καταναλώνει μεγάλο ποσοστό του χρόνου εκτέλεσης της εκπαίδευσης νευρωνικών δικτύων. Ξεκινώντας από μία απλοϊκή αρχική υλοποίηση, στο τέλος της εργασίας θα έχετε πετύχει μία αρκετά αποδοτική υλοποίηση του αλγορίθμου GEMM για CPUs και GPUs, η οποία θα οδηγήσει σε επιτάχυνση της εκπαίδευσης.

2.1 Παραλληλοποίηση σε CPU

Σας δίνονται σειριακές υλοποιήσεις τριών διαφορετικών παραλλαγών του πολλαπλασιασμού πινάκων στο αρχείο `linalg.c`. Συγκεκριμένα, σας δίνονται οι `dgemm`, `dgemm_ta` και `dgemm_tb`, που υπολογίζουν αντίστοιχα τα $A \cdot B$, $A^T \cdot B$ και $A \cdot B^T + C$, όπου A, B και C πίνακες εισόδου. Αφού τις μελετήσετε, ζητείται:

1. η παραλληλοποίησή τους με χρήση του προγραμμτιστικού μοντέλου OpenMP και
2. η υλοποίησή τους με χρήση της βιβλιοθήκης OpenBLAS.

Για το Ερώτημα 1 θα χρειαστεί να συμπληρώσετε τις `dgemm`, `dgemm_ta` και `dgemm_tb`, ενώ για το Ερώτημα 2 θα χρειαστεί να μελετήσετε τη διεπαφή (API) της βιβλιοθήκης OpenBLAS και συγκεκριμένα της συνάρτησης `cblas_dgemm(...)`. Θεωρήστε ότι οι πίνακες είναι αποθηκευμένοι στη μνήμη κατά γραμμές (row-major).

2.2 Παραλληλοποίηση σε GPU

Σας δίνονται απλοϊκές παράλληλες υλοποιήσεις σε CUDA τριών διαφορετικών παραλλαγών του πολλαπλασιασμού πινάκων στο αρχείο `linalg.cu`. Συγκεκριμένα, σας δίνονται οι `dgemm_gpu`, `dgemm_ta_gpu` και `dgemm_tb_gpu`, που υπολογίζουν αντίστοιχα τα $A \cdot B$, $A^T \cdot B$ και $A \cdot B^T + C$, όπου A, B και C πίνακες εισόδου. Αφού τις μελετήσετε, ζητείται:

1. η βελτιστοποίησή τους με χρήση κοινής μνήμης (shared memory) και
2. η υλοποίησή τους με χρήση της βιβλιοθήκης cuBLAS.

Για το Ερώτημα 1 θα χρειαστεί να συμπληρώσετε τις `dgemm_shmem`, `dgemm_ta_shmem` και `dgemm_tb_shmem` και να μεταγλωττίσετε τον κώδικα με την επιλογή GEMM_OPTIMIZED=1, ενώ για το Ερώτημα 2 θα χρειαστεί να μελετήσετε τη διεπαφή (API) της βιβλιοθήκης cuBLAS και συγκεκριμένα της συνάρτησης `cublasDgemm(...)`. Θεωρήστε ότι οι πίνακες είναι αποθηκευμένοι στη μνήμη κατά γραμμές (row-major) και διαβάστε προσεκτικά πώς θεωρεί η βιβλιοθήκη cuBLAS ότι είναι αποθηκευμένα οι πίνακες στην μνήμη.

3 Υποδείξεις και διευκρινίσεις

3.1 Δομή κώδικα

Για την διευκόλυνσή σας, αλλά και για να υπάρχει ένας κοινός τρόπος μέτρησης του χρόνου εκτέλεσης, σας δίνεται πλήρης και λειτουργικός σκελετός του κώδικα της εργασίας. Ο κώδικας βρίσκεται στον `scirouter`, στο φάκελο `/home/parml/shared/ex1_omp_cuda` και αποτελείται από κάποια script για τη μεταγλώττιση και εκτέλεση στην ουρά `termis` καθώς και τον υποφακέλο `/src` ο οποίος περιέχει τον κώδικα. Οι προσθήκες που ζητούνται να γίνουν αφορούν τα αρχεία `src/linalg.c/cu`, όπου υπάρχει η ένδειξη "FILLME". Σας παρέχονται επιπλέον τα κατάλληλα Makefile για την μεταγλώττιση και τη σύνδεση του κώδικά σας, ένα για CPU κι ένα για GPU. Πληκτρολογώντας `make -f Makefile.cpu` δημιουργούνται δύο εκτελέσιμα, για το Ερώτημα 2.1, και αντίστοιχα, πληκτρολογώντας `make -f Makefile.gpu` δημιουργούνται δύο εκτελέσιμα για το Ερώτημα 2.2.

3.2 Περιβάλλον ανάπτυξης

Θα τρέξετε τον κώδικά σας σε διαφορετικά μηχανήματα του εργαστηρίου. Για το Ερώτημα 2.1 που αφορά σε CPU, θα χρησιμοποιήσετε τα μηχανήματα `termi1` ή `termi2` της ουράς `termis`, ενώ για το Ερώτημα 2.2. που αφορά σε GPU, θα χρησιμοποιήσετε το μηχάνημα `dungani` της ουράς `serial` με εγκατεστημένη κάρτα γραφικών γενιάς 3.5 (NVIDIA Tesla K40). Για περισσότερες πληροφορίες σχετικά με τα λεπτομερή τεχνικά χαρακτηριστικά της GPU, μπορείτε να εκτελέσετε το πρόγραμμα `deviceQuery` ποιυ βρίσκεται στον κάταλογο `/usr/local/cuda/samples/1_Utilities/deviceQuery` όντας στο μηχάνημα `dungani`. ΠΡΟΣΟΧΗ - με την default σειρά η Tesla K40 δεν είναι η πρώτη, καθώς το dungani έχει περισσότερες από 1 GPU. Για τη χρήση της Tesla K40 στο dungani χρειάζεται η εντολή `export CUDA_DEVICE_ORDER=PCI_BUS_ID` στο μηχάνημα (script υποβολής) και `cudaSetDevice(0)` στην main, τα οποία είναι ήδη συμπληρωμένα. Η χρήση των παραπάνω μηχανημάτων θα γίνεται μέσω του συστήματος υποβολής εργασιών Torque. Για τη μεταγλώττιση των προγραμμάτων σας, θα πρέπει να χρησιμοποιήσετε το κατάλληλο μηχάνημα, μέσω του συστήματος Torque, ως εξής:

```
$ qsub compile_on_cpu.sh
$ qsub compile_on_gpu.sh
```

Για την εκτέλεση των προγραμμάτων σας, θα πρέπει επίσης να χρησιμοποιήσετε το κατάλληλο μηχάνημα μέσω του συστήματος Torque, ως εξής:

```
$ qsub run_omp.sh
$ qsub run_blas.sh
$ qsub run_cuda.sh
$ qsub run_cublas.sh
```

3.3 Έλεγχος ορθότητας

Για τον έλεγχο ορθότητας των υλοποιήσεών σας σας δίνονται τα πρόγραμματα `test_omp`, `test_blas`, `test_cuda` και `test_cublas`, τα οποία και θα εκτελείτε κάθε φορά που κάνετε αλλαγές στον κώδικα. Αν τυπωθεί η ένδειξη FAILED τότε θα πρέπει να επαναξετάσετε τις υλοποιήσεις σας.

4 Πειράματα και μετρήσεις επιδόσεων

4.1 Σενάριο μετρήσεων και διαγράμματα

Το μηχάνημα στο οποίο θα εκτελέσετε τα πειράματά σας για το Ερώτημα 2.1 αποτελείται από δύο επεξεργαστές Intel Xeon X5650 (6 πυρήνες + 2 hyper-threads ανά πυρήνα, συνολικά 12 πυρήνες + 24 hyper-threads). Το μηχάνημα στο οποίο θα εκτελέσετε τα πειράματά σας για το Ερώτημα 2.2 αποτελείται από έναν επεξεργαστή Intel i7-4820K (4 πυρήνες + 2 hyper-threads ανά πυρήνα, συνολικά 8 hyper-threads) και μία κάρτα γραφικών NVIDIA Tesla K40.

Κλιμακωσιμότητα σε CPU

Αρχικά, σας ζητείται να εξετάσετε και να σχολιάσετε την κλιμακωσιμότητα της εκπαίδευσης στη CPU με τις υλοποιήσεις OpenMP και OpenBLAS. Συγκεκριμένα, ζητείται ένα διάγραμμα χρόνου εκτελεσης καθώς μεταβάλλεται ο αριθμός των νημάτων, καθώς και ένα διάγραμμα επιτάχυνσης (speedup) ως προς τη σειριακή υλοποίηση (χωρίς OpenMP). Σε κάθε διάγραμμα να απεικονίζονται αποτελέσματα και από τις δύο υλοποιήσεις.

Σύγκριση επιδόσεων σε CPU και GPU

Επιπλέον, σας ζητείται να εξετάσετε και να σχολιάσετε την επίδοση της εκπαίδευσης στη CPU και στη GPU. Συγκεκριμένα, ζητείται να μετρήσετε και να απεικονίσετε σε διάγραμμα με μπάρες το συνολικό χρόνο εκτέλεσης της εκπαίδευσης αλλά και το χρόνο που αναλώνεται σε πολλαπλασιασμούς πινάκων για τα παρακάτω σενάρια εκπαίδευσης:

- σειριακή έκδοση της GEMM για CPUs
- παράλληλη έκδοση της GEMM για CPUs με OpenMP
- παράλληλη έκδοση της GEMM για CPUs με OpenBLAS
- παράλληλη naive έκδοση της GEMM για GPUs με CUDA
- παράλληλη shmem έκδοση της GEMM για GPUs με CUDA
- παράλληλη έκδοση της GEMM για GPUs με cuBLAS

Για κάθε σενάριο εκπαίδευσης θα απεικονίσετε μία μοναδική μπάρα με τον συνολικό χρόνο εκτέλεσης στην οποία θα δείχνετε και τι κομμάτι του χρόνου αναλώνεται σε κάθε μία από τις τρεις παραλλαγές του πολλαπλασιασμού πινάκων.