ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ
ΕΡΓΑΣΤΗΡΙΟ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ
www.cslab.ece.ntua.gr

Συστήματα Παράλληλης Επεξεργασίας
2021–2022

Άσκηση 4:
Παράλληλος προγραμματισμός σε επεξεργαστές γραφικών

Προθεσμίες παράδοσης
Τελική αναφορά: 26/1/2022[1] [2]

1 Πολλαπλασιασμός πίνακα με πίνακα

Ο πολλαπλασιασμός πίνακα με πίνακα (Dense Matrix-Matrix multiplication, DMM) είναι ένας από τους πιο σημαντικούς υπολογιστικούς πυρήνες αλγεβρικών υπολογισμών. Έστω οι πίνακες εισόδου **A** και **B** με διαστάσεις $M \times K$ και $K \times N$ αντίστοιχα. Ο πυρήνας DMM υπολογίζει τον πίνακα εξόδου $\mathbf{C} = \mathbf{A} \cdot \mathbf{B}$ διαστάσεων $M \times N$. Σε αναλυτική μορφή κάθε στοιχείο του πίνακα εξόδου υπολογίζεται ως:

$$C_{i,j} = \sum_{k=1}^{K} A_{ik} \cdot B_{kj}, \forall i \in [1, M], \forall j \in [1, N]$$

2 Ζητούμενα

2.1 Υλοποίηση για επεξεργαστές γραφικών (GPUs)

Στην άσκηση αυτή θα υλοποιήσετε τον αλγόριθμο DMM για επεξεργαστές γραφικών χρησιμοποιώντας το προγραμματιστικό περιβάλλον CUDA. Ξεκινώντας από μία απλοϊκή αρχική υλοποίηση, στο τέλος της άσκησης θα έχετε πετύχει μία αρκετά αποδοτική υλοποίηση του αλγορίθμου DMM για τους επεξεργαστές γραφικών. Υποθέστε ότι οι πίνακες είναι αποθηκευμένοι στη μνήμη κατά γραμμές (row-major format). Οι υλοποιήσεις σας θα πρέπει να λειτουργούν για οποιαδήποτε τιμή των διαστάσεων M, N, K, εφόσον τα αντίστοιχα δεδομένα του προβλήματος χωράνε στην κύρια μνήμη της GPU (global memory).

Βασική υλοποίηση

Υλοποιήστε τον αλγόριθμο DMM, αναθέτοντας σε κάθε νήμα εκτέλεσης τον υπολογισμό ενός[3] στοιχείου του πίνακα εξόδου.

[1] filename a4-parlabXX-final.pdf
[2] Υποβολή στο portal του μαθήματος
[3] ή και περισσότερων αν το grid δεν μπορεί να καλύψει το μέγεθος του προβλήματος

1. Υπολογίστε τον αριθμό των προσβάσεων στην κύρια μνήμη για τους πίνακες **A** και **B** συναρτήσει των διαστάσεων M, N, K του προβλήματος και των διαστάσεων του μπλοκ νημάτων (`THREAD_BLOCK_X` για 1Δ μπλοκ και `THREAD_BLOCK_X`/`THREAD_BLOCK_Y` για 2Δ μπλοκ).
2. Υπολογίστε το πηλίκο των πράξεων κινητής υποδιαστολής (floating-point operations) προς τον αριθμό των προσβάσεων στην κύρια μνήμη μίας επανάληψης του εσωτερικού βρόχου της υλοποίησής σας. Η επίδοση της υλοποίησής σας περιορίζεται από το εύρος ζώνης της κύριας μνήμης (memory-bound) ή το ρυθμό εκτέλεσης πράξεων κινητής υποδιαστολής των πολυεπεξεργαστικών στοιχείων (compute-bound); Αιτιολογήστε την απάντησή σας.
3. Ποιες από τις προσβάσεις στην κύρια μνήμη συνενώνονται και ποιες όχι με βάση την υλοποίησή σας;
4. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων και καταγράψετε την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων με χρήση του CUDA Occupancy Calculator και την επίδοση του κώδικά σας.

Συνένωση των προσβάσεων στην κύρια μνήμη

Τροποποιήστε την προηγούμενη υλοποίηση ώστε να να επιτύχετε συνένωση των προσβάσεων στην κύρια μνήμη για τον πίνακα **A** προφορτώνοντας τμηματικά στοιχεία του στην τοπική μνήμη των πολυεπεξεργαστικών στοιχείων (shared memory). Μην τροποποιήσετε τον τρόπο ανάθεσης υπολογισμών σε νήματα.

1. Υπολογίστε τον αριθμό των προσβάσεων στην κύρια μνήμη για τον πίνακα **A** συναρτήσει των διαστάσεων M, N, K του προβλήματος, των διαστάσεων του μπλοκ νημάτων (`THREAD_BLOCK_X` για 1Δ μπλοκ και `THREAD_BLOCK_X`/`THREAD_BLOCK_Y` για 2Δ μπλοκ) και των διαστάσεων του μπλοκ υπολογισμού (`TILE_X` για 1Δ μπλοκ και `TILE_X`/`TILE_Y` για 2Δ μπλοκ). Κατά πόσο μειώνονται οι προσβάσεις σε σχέση με την προηγούμενη υλοποίηση;
2. Υπολογίστε το πηλίκο των πράξεων κινητής υποδιαστολής (floating-point operations) προς τον αριθμό των προσβάσεων στην κύρια μνήμη μίας επανάληψης του εσωτερικού βρόχου της υλοποίησής σας. Η επίδοση της υλοποίησής σας περιορίζεται από το εύρος ζώνης της κύριας μνήμης (memory-bound) ή το ρυθμό εκτέλεσης πράξεων κινητής υποδιαστολής των πολυεπεξεργαστικών στοιχείων (compute-bound); Αιτιολογήστε την απάντησή σας.
3. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων/υπολογισμού και καταγράψετε την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων με χρήση του CUDA Occupancy Calculator και την επίδοση του κώδικά σας.

Μείωση των προσβάσεων στην κύρια μνήμη

Αξιοποιήστε την τοπική μνήμη των πολυεπεξεργαστικών στοιχείων (shared memory) για να μειώσετε περαιτέρω τις προσβάσεις στην κύρια μνήμη προφορτώνοντας τμηματικά και στοιχεία του B στην τοπική μνήμη των πολυεπεξεργαστικών στοιχείων (shared memory). Μην τροποποιήσετε τον τρόπο ανάθεσης υπολογισμών σε νήματα.

1. Υπολογίστε τον αριθμό των προσβάσεων στην κύρια μνήμη για τους πίνακες **A** και **B** συναρτήσει των διαστάσεων M, N, K του προβλήματος, των διαστάσεων του μπλοκ νημάτων (`THREAD_BLOCK_X` για 1Δ μπλοκ και `THREAD_BLOCK_X`/`THREAD_BLOCK_Y` για 2Δ μπλοκ) και των διαστάσεων του μπλοκ υπολογισμού (`TILE_X` για 1Δ μπλοκ και `TILE_X`/`TILE_Y` για 2Δ μπλοκ). Κατά πόσο μειώνονται οι προσβάσεις στην κύρια μνήμη σε σχέση με την προηγούμενη υλοποίηση;
2. Υπολογίστε το πηλίκο των πράξεων κινητής υποδιαστολής (floating-point operations) προς τον αριθμό των προσβάσεων στην κύρια μνήμη μίας επανάληψης του εσωτερικού βρόχου της υλοποίησής σας. Η επίδοση της υλοποίησής σας περιορίζεται από το εύρος ζώνης της κύριας μνήμης (memory-bound) ή το ρυθμό εκτέλεσης πράξεων κινητής υποδιαστολής των SM (compute-bound); Αιτιολογήστε την απάντησή σας.

3. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων/υπολογισμού και καταγράψετε την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων με χρήση του CUDA Occupancy Calculator και την επίδοση του κώδικά σας.

Χρήση της βιβλιοθήκης cuBLAS

Χρησιμοποιήστε την συνάρτηση `cublasSgemm()` της βιβλιοθήκης cuBLAS για την υλοποίηση του πολλαπλασιασμού πινάκων. Η βιβλιοθήκη cuBLAS αποτελεί υλοποίηση της BLAS για τις κάρτες γραφικών της NVIDIA. Χρησιμοποιήστε τις κατάλληλες παραμέτρους για τους βαθμωτούς `alpha` και `beta` που απαιτεί η συνάρτηση. Επιπλέον, διαβάστε προσεκτικά πως θεωρεί η βιβλιοθήκη cuBLAS ότι είναι αποθηκευμένοι οι πίνακες στη μνήμη για να καθορίσετε την τιμή της παραμέτρου `trans`.

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 άσκησης, καθώς και η ρουτίνα της σειριακής εκτέλεσης σε CPU του DMM. Ο κώδικας βρίσκεται στον scirouter, στο φάκελο `/home/parallel/pps/2021-2022/a4/dmm-skeleton` και αποτελείται από δύο υποφακέλους `common` και `cuda`. Εσείς θα κάνετε προσθήκες στον υποφάκελο `cuda` ο οποίος περιέχει:

Κυρίως πρόγραμμα: Πρόκειται για το αρχείο `dmm_main.cu`, το οποίο περιέχει την συνάρτηση `main()` του προγράμματος, η οποία είναι υπεύθυνη (α′) για την ανάγνωση των ορισμάτων της γραμμής εντολών, (β′) για την δημιουργία των πινάκων εισόδου/εξόδου, (γ′) για την αρχικοποίηση (παραχωρήσεις μνήμης, παράμετροι πυρήνα), εκτέλεση και χρονομέτρηση του πυρήνα στην GPU, και (δ′) για τον έλεγχο της εγκυρότητας των αποτελεσμάτων.

Υλοποιήσεις για CPU: Πρόκειται για το αρχείο `dmm.c`, το οποίο περιέχει την σειριακή υλοποίηση του DMM για CPUs.

Υλοποιήσεις για GPU: Πρόκειται για το αρχείο `dmm_gpu.cu`, το οποίο περιέχει τις υλοποιήσεις του DMM για GPUs.

Βοηθητικά αρχεία: Πρόκειται για τα αρχεία `mat_util.c` και `gpu_util.cu`, τα οποία περιέχουν βοηθητικές συναρτήσεις για το χειρισμό δισδιάστατου πίνακα και για το χειρισμό των GPUs αντίστοιχα.

Σας παρέχονται επιπλέον και τα κατάλληλα `Makefiles` για την μεταγλώττιση και την σύνδεση του κώδικά σας. Πληκτρολογείστε '`make help`', ώστε να δείτε τις διάφορες επιλογές που σας δίνονται κατά την μεταγλώττιση. Η προεπιλεγμένη ρύθμιση για την μεταγλώττιση είναι σε λειτουργία debug (όλες οι βελτιστοποιήσεις του μεταγλωττιστή απενεργοποιημένες).

Τα σημεία του κώδικα, στα οποία θα πρέπει να επέμβετε είναι σημειωμένα με την επιγραφή '`FILLME:`' μαζί με μία σύντομη περιγραφή. Οι ζητούμενες προσθήκες θα γίνουν στα αρχεία `dmm_gpu.cu` και `dmm_main.cu`. Τέλος, για να δείτε τον τρόπο χρήσης του τελικού εκτελέσιμου, εκτελέστε '`./dmm_main`' από την γραμμή εντολών και θα παρουσιαστεί ένα σύντομο μήνυμα βοήθειας για την σωστή χρήση του.

3.2 Περιβάλλον ανάπτυξης

Θα τρέξετε τον κώδικά σας σε μηχάνημα του εργαστηρίου (`dungani`) με εγκατεστημένη κάρτα γραφικών γενιάς 3.5 (NVIDIA Tesla K40c, αρχιτεκτονική Kepler). Για περισσότερες πληροφορίες σχετικά με τα τεχνικά χαρακτηριστικά της GPU, πληκτρολογείστε '`make query`' όντας στο μηχάνημα. Η χρήση του μηχανήματος για την εκτέλεση της άσκησης θα γίνεται μέσω του συστήματος υποβολής εργασιών Torque, όπως και στις προηγούμενες ασκήσεις. Συγκεκριμένα, η υποβολή στο σύστημα Torque θα γίνεται ως εξής:

```
$ qsub -q serial -l nodes=dungani:ppn=1 myjob.sh
```

ΠΡΟΣΟΧΗ: Στο μηχάνημα dungani υπάρχουν εγκατεστημένες κι άλλες κάρτες γραφικών πέραν της NVIDIA Tesla K40c που έχει dev_id = 2. Θα πρέπει να φροντίσετε ο κώδικας σας να τρέχει πάντα σε αυτή (το οποίο μπορείτε να επιβεβαιώσετε μέσω του .sh αλλά και μεσα στον κώδικα).

4 Πειράματα και μετρήσεις επιδόσεων

4.1 Σενάριο μετρήσεων και διαγράμματα

Σκοπός των μετρήσεων είναι (α′) η μελέτη της επίδρασης του μεγέθους του μπλοκ νημάτων/υπολογισμού στην επίδοση των υλοποιήσεών σας και (β′) η σύγκριση της επίδοσης όλων εκδόσεων του πυρήνα DMM. Για την διευκόλυνσή σας σας δίνεται ένα script μετρήσεων `run_dmm.sh` το οποίο μπορείτε να χρησιμοποιήσετε ως βάση. Συγκεκριμένα, σας ζητούνται τα παρακάτω σύνολα μετρήσεων:

- Για κάθε έκδοση πυρήνα που υλοποιήσατε (naive, coalesced, shmem) να καταγράψετε πώς μεταβάλλεται η επίδοση για διαφορετικές διαστάσεις του μπλοκ νημάτων (`THREAD_BLOCK_X/Y`) και υπολογισμού (`TILE_X/Y`) για διαστάσεις πινάκων $M = N = K = 2048$.
- Για κάθε μία από τις τέσσερις εκδόσεις πυρήνων (naive, coalesced, shmem και cuBLAS) να καταγράψετε την επίδοση για μεγέθη πινάκων $M, N, K \in [256, 512, 1024, 2048]$. Για τις naive, coalesced και shmem υλοποιήσεις επιλέξτε τις βέλτιστες διαστάσεις μπλοκ νημάτων και υπολογισμών.
- **Bonus: Υπολογισμός transfer overhead**. Σε καθένα από τα παραπάνω μεγέθη πινάκων $M, N, K \in [256, 512, 1024, 2048]$ υπολογίστε και το χρόνο που απαιτείται για την μεταφορά δεδομένων (ίδια για όλα τα kernel) και την επίδοση αυτής. Για αυτό χρειάζεται η προσθήκη δικών σας timer στην dmm_main. Σχολιάστε συγκρίνοντας με την αντίστοιχη επίδοση εκτέλεσης που βρήκατε παραπάνω πότε πιστεύετε πως το μέγεθος προβλήματος δικαιολογεί τη χρήσης της συγκεκριμένης GPU. Θα μπορούσατε να γενικεύσετε για οποιαδήποτε M, N, K σε οποιαδήποτε GPU αν είναι γνωστά τα $\mathrm{Rmax}(\mathrm{Gflops/s})$ και $\mathtt{transfer\_bw}(\mathrm{Gbytes/s})$, υποθέτοντας πως $\mathtt{dmm\_performance} = \mathrm{Rmax}$?

ΠΡΟΣΟΧΗ: Στο Makefile που σας δίνεται, η προεπιλεγμένη ρύθμιση για την μεταγλώττιση είναι σε λειτουργία debug (όλες οι βελτιστοποιήσεις του μεταγλωττιστή απενεργοποιημένες). Για να μετρήσετε τις επιδόσεις των πυρήνων θα πρέπει να απενεργοποιήσετε αυτή την λειτουργία και να μεταγλωττίσετε εξαρχής (‘`make clean`’) τον κώδικά σας με ‘`make DEBUG=0`’.

5 Τελική αναφορά

Στην τελική αναφορά για την άσκηση σας ζητείτε για κάθε μία από τις υλοποιήσεις: (α′) να παραθέσετε τον αντίστοιχο κώδικά με επεξήγηση, (β′) να απαντήσετε στα σχετικά ερωτήματα, (γ′) να απεικονίσετε την επίδοσή της με βάση το σενάριο μετρήσεων που ζητείται και (δ′) να ερμηνεύσετε την συμπεριφορά της επίδοσής της.