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
2017–2018

Άσκηση 4:
Παράλληλος προγραμματισμός σε επεξεργαστές γραφικών

Προθεσμία παράδοσης: *9 Ιαν. 2018*

1 Πολλαπλασιασμός πίνακα με διάνυσμα

Ο πολλαπλασιασμός πίνακα με διάνυσμα (Dense Matrix-Vector multiplication, DMV) είναι ένας από τους πιο σημαντικούς υπολογιστικούς πυρήνες αλγεβρικών υπολογισμών που συναντάται σε πληθώρα επιστημονικών εφαρμογών. Έστω το διάνυσμα εισόδου $\mathbf{x}$ και η μήτρα (δισδιάστατος πίνακας) τιμών $\mathbf{A}$. Ο πυρήνας DMV υπολογίζει το διάνυσμα εξόδου $\mathbf{y} = \mathbf{Ax}$. Σε αναλυτική μορφή θεωρώντας τετραγωνικό $N \times N$ πίνακα, το γινόμενο γράφεται ως

$$y_i = \sum_{j=1}^{N} a_{ij} x_j, \forall i \in [1, N]$$

2 Ζητούμενα

2.1 Υλοποίηση για επεξεργαστές γραφικών (GPUs)

Στην άσκηση αυτή θα πρέπει να υλοποιήσετε τον αλγόριθμο DMV για τους επεξεργαστές γραφικών του εργαστηρίου, χρησιμοποιώντας το προγραμματιστικό περιβάλλον CUDA ή/και OpenCL. Ξεκινώντας από μία απλοϊκή αρχική υλοποίηση, στο τέλος της άσκησης θα έχετε πετύχει μία αρκετά αποδοτική υλοποίηση του αλγορίθμου DMV για τους επεξεργαστές γραφικών.

Βασική υλοποίηση

Υλοποιήστε τον αλγόριθμο DMV, αναθέτοντας σε κάθε νήμα εκτέλεσης τον υπολογισμό μιας γραμμής του πίνακα εισόδου.

1. Καταγράψετε την επίδοση και συγκρίνετέ την με την επίδοση του κώδικα για την CPU (σειριακό και OpenMP).
2. Γιατί ο κώδικάς σας είναι απογοητευτικά αργός;

[-1] filename a4-parlabXX-final.pdf
[0] mail to: athena@cslab.ece.ntua.gr, cc: goumas@cslab.ece.ntua.gr

Συνένωση των προσβάσεων στην κύρια μνήμη (memory access coalescing)

Τροποποιήστε κατάλληλα τον κώδικά σας και τον τρόπο αποθήκευσης του πίνακα **A**, ώστε να επιτύχετε συνένωση των προσβάσεων στην κύρια μνήμη για τον πίνακα **A**.

1. Καταγράψετε την επίδοση και συγκρίνετέ την με την επίδοση της βασικής έκδοσης.
2. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων και καταγράψετε την επίδοση του κώδικά σας. Για περιβάλλον ανάπτυξης CUDA καταγαψτε και την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) με χρήση του CUDA Occupancy Calculator.

Χρήση της τοπικής on-chip μνήμης

Χρησιμοποιείστε την τοπική μνήμη των πολυεπεξεργαστικών στοιχείων (shared memory), ώστε να προφορτώνετε (prefetch) τμηματικά το διάνυσμα εισόδου **x** σε αυτή και στην συνέχεια να εκτελείτε τους υπολογισμούς σε αυτό. Θα πρέπει να προσέξετε, οι προσβάσεις στην κύρια μνήμη για την προφόρτωση του **x** να μπορούν να συνενωθούν, ώστε να μπορέσετε να αποκομίσετε οφέλη από αυτή την βελτιστοποίηση. Άλλοι τρόποι χρήσης της τοπικής μνήμης για βελτίωση της επίδοσης είναι επίσης ευσπρόσδεκτοι.

1. Καταγράψετε την επίδοση και συγκρίνετέ την με την επίδοση του κώδικα της προηγούμενης βελτιστοποίησης. Πού οφείλεται η διαφορά, εάν υπάρχει; Εάν δεν υπάρχει, πώς το εξηγείτε βάσει της αρχιτεκτονικής της κάρτας γραφικών που χρησιμοποιήσατε (Fermi); Μπορείτε να φανταστείτε την συμπεριφορά της βελτιστοποίησης αυτής σε παλαιότερες αρχιτεκτονικές χωρίς κρυφή μνήμη;
2. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων και καταγράψετε την επίδοση του κώδικά σας. Για περιβάλλον ανάπτυξης CUDA καταγαψτε και την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) με χρήση του CUDA Occupancy Calculator.
3. Παρατηρείτε κάποια συσχέτιση του χρόνου εκτέλεσης με το μέγεθος του μπλοκ νημάτων και του πλήθους των μπλοκ; Δώστε μία σύντομη εξήγηση.

Χρήση της βιβλιοθήκης cuBLAS (περιβάλλον CUDA) ή clBLAS (περιβάλλον OpenCL)

Τροποποιήστε κατάλληλα τον κώδικά σας ώστε να χρησιμοποιήσετε την συνάρτηση `cublasSgemv()` της βιβλιοθήκης cuBLAS ή `clblasSgemv()` της βιβλιοθήκης clBLAS για την υλοποίηση του ζητούμενου πολλαπλασιασμού. Χρησιμοποιήστε τις κατάλληλες παραμέτρους για τους βαθμωτούς `alpha` και `beta` που απαιτεί η συνάρτηση. Επιπλέον, διαβάστε προσεκτικά πως θεωρεί η κάθε βιβλιοθήκη ότι είναι αποθηκευμένος ο πίνακας στη μνήμη για να καθορίσετε την σωστή τιμή της παραμέτρου `trans`.

1. Καταγράψετε την επίδοση και συγκρίνετέ την με την επίδοση του κώδικα που αναπτύξατε στα προηγούμενα ερωτήματα της εργασίας.

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 οι ρουτίνες της σειριακής και παράλληλης εκτέλεσης του DMV με OpenMP. Ο κώδικας βρίσκεται στον scirouter, στο φάκελο `/home/parallel/pps/2017-2018/a4` και για καθένα από τα προγραμματιστικά περιβάλλοντα (`/cuda` και `/opencl`) αποτελείται από τρία βασικά μέρη:

Κυρίως πρόγραμμα: Πρόκειται για το αρχείο `dmv_main.*`, το οποίο περιέχει την συνάρτηση `main()` του προγράμματος, η οποία είναι υπεύθυνη (α′) για την ανάγνωση των ορισμάτων της γραμμής εντολών και των μεταβλητών περιβάλλοντος (βλ. παρακάτω), (β′) για την δημιουργία του

πίνακα εισόδου και των διανυσμάτων εισόδου/εξόδου, (γ′) για την εκτέλεση και χρονομέτρηση του σειριακού και του παράλληλου με OpenMP πυρήνα, (δ′) για την αρχικοποίηση (παραχωρήσεις μνήμης, παράμετροι πυρήνα), εκτέλεση και χρονομέτρηση του πυρήνα στην GPU, και (ε′) για τον έλεγχο της εγκυρότητας των αποτελεσμάτων.

Υλοποιήσεις για CPU: Πρόκειται για το αρχείο `dmv.c`, το οποίο περιέχει την σειριακή και παράλληλη υλοποίηση του DMV για τις CPUs, καθώς και κάποιες βοηθητικές συναρτήσεις για την αρχικοποίηση του πίνακα και τον χειρισμό των διανυσμάτων.

Υλοποιήσεις για GPU: Πρόκειται για το αρχείο `dmv_gpu.*`, το οποίο περιέχει τις υλοποιήσεις των πυρήνων για GPUs.

Επιπλέον, κάποια βοηθητικά αρχεία τα οποία περιέχουν βοηθητικές συναρτήσεις για δυναμική παραχώρηση μνήμης δισδιάστατου πίνακα, χειρισμού λαθών και χρονομέτρησης υπάρχουν στο `/common`. Σας παρέχετονται επιπλέον και τα κατάλληλα `Makefiles` για την μεταγλώττιση και την σύνδεση του κώδικά σας. Πληκτρολογείστε '`make help`' είτε από το root directory είτε από το από κάποιο από τα `/cuda` και `/opencl`, ώστε να δείτε τις διάφορες επιλογές που σας δίνονται κατά την μεταγλώττιση.

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 `dmv_gpu.*` και `dmv_main.*`. Τέλος, για να δείτε τον τρόπο χρήσης του τελικού εκτέλεσιμου, εκτελέστε '`./dmv_main`' από την γραμμή εντολών και θα παρουσιαστεί ένα σύντομο μήνυμα βοήθειας για την σωστή χρήση του.

ΠΡΟΣΟΧΗ: Στα Makefile που σας δίνονται, 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`'.

3.2 Περιβάλλον ανάπτυξης

Θα τρέξετε τον κώδικά σας σε μηχάνημα του εργαστηρίου (`termis`) με εγκατεστημένη κάρτα γραφικών γενιάς 2.0 (nVidia Tesla M2050, αρχιτεκτονική Fermi). Για περισσότερες πληροφορίες σχετικά με τα λεπτομερή τεχνικά χαρακτηριστικά της GPU, πληκτρολογείστε '`make query`' όντας σε ένα μηχάνημα termi. Η χρήση των υπολογιστών του εργαστηρίου (ουρά termis) για την εκτέλεση της άσκησης θα γίνεται μέσω του συστήματος υποβολής εργασιών Torque, όπως και στις προηγούμενες ασκήσεις. Για την εκτέλεση των μετρήσεων σε πραγματική GPU, θα πρέπει να δεσμεύσετε το κατάλληλο μηχάνημα από το σύστημα Torque ως εξής:

```
$ qsub -q termis -l nodes=1:ppn=24:cuda myjob.sh
```

4 Πειράματα και μετρήσεις επιδόσεων

4.1 Σενάριο μετρήσεων και διαγράμματα

Σκοπός των μετρήσεων είναι (α′) η σύγκριση της επίδοσης των διαφόρων εκδόσεων του πυρήνα DMV για GPUs σε σχέση με την σειριακή και παράλληλη υλοποίησή του για CPUs, και (β′) η μελέτη της επίδρασης στην επίδοση του μεγέθους του μπλοκ για τις υλοποιήσεις σε GPUs. Το μηχάνημα στο οποίο θα εκτελέσετε τα πειράματά σας αποτελείται από δύο επεξεργαστές Intel Xeon X5650 (6 πυρήνες + H/T, συνολικά 12 πυρήνες + H/T) και μία κάρτα γραφικών nVidia Tesla M2050 αρχιτεκτονικής Fermi. Συνοπτικά, σας ζητούνται τα παρακάτω σύνολα μετρήσεων:

Σειριακή έκδοση για CPUs Να καταγράψετε την επίδοση για μεγέθη πινάκων 1 Ki[1], 2 Ki, 4 Ki, 7 Ki, 8 Ki, 14 Ki και 16 Ki.

[1] 1 Ki = 1024 bytes.

Παράλληλη έκδοση για CPUs Να καταγράψετε την επίδοση για τα προαναφερθέντα μεγέθη πινάκων για 1, 2, 6, 12 και 24 νήματα. Για την αξιοπιστία των μετρήσεών σας, θα πρέπει να «συνδέσετε» κάθε νήμα με συγκεκριμένο επεξεργαστή χρησιμοποιώντας την μεταβλητή περιβάλλοντας `GOMP_CPU_AFFINITY=0-23`. Με αυτό τον τρόπο[2] «γεμίζετε» πρώτα τον πρώτο επεξεργαστή (6 πυρήνες), στην συνέχεια τον δεύτερο (12 πυρήνες) και, τέλος, χρησιμοποιείτε και το HyperThreading (24 h/w νήματα).

Παράλληλη έκδοση για GPUs Να καταγράψετε την επίδοση για τα προαναφερθέντα μεγέθη πινάκων και για μεγέθη μπλοκ 16–512 με βήμα 16 για κάθε μία από τις τέσσερις εκδόσεις πυρήνων (naive, coalesced, shmem και cuBLAS) που υλοποιήσατε.

Στην τελική αναφορά σας για την άσκηση, σας ζητείτε να ερμηνεύσετε (σύντομα) την συμπεριφορά της επίδοσης του DMV τόσο για τον κώδικα OpenMP όσο και για τον κώδικα σε GPUs. Για παράδειγμα,

- Πώς εξηγείται η διαφορά επίδοσης στην έκδοση του OpenMP μεταξύ μικρών και μεγάλων πινάκων; Πότε θα θεωρούσατε ένα πίνακα «μεγάλο» για την συγκεκριμένη αρχιτεκτονική;
- Στην μετάβαση από τα 6 στα 12 νήματα, μπορείτε να εξηγήσετε την «κάμψη» στην βελτίωση της επίδοσης της έκδοσης για OpenMP;
- Παρατηρείτε κάποια συσχέτιση της επίδοσης της έκδοσης για GPUs με το μέγεθος του μπλοκ που επιλέγετε; Πώς επηρεάζεται η χρησιμοποιήση ολόκληρης της GPU από την επιλογή του μπλοκ;

[2] Ο τρόπος που ανατίθενται τα CPU IDs στους λογικούς επεξεργαστές ενός συστήματος δεν είναι γενικός. Η σειρά που σας δίνουμε αφορά συγκεκριμένα το μηχάνημα στο οποίο πρόκειται να εκτελέσετε τα πειράματά σας.