



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
2011–2012

Άσκηση 3: Παράλληλος προγραμματισμός σε επεξεργαστές γραφικών

Προθεσμίες παράδοσης

Επίδειξη προγραμμάτων

7 Φεβ. 2012

Τελική αναφορά

Ημ/νία γραπτής εξέτασης

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 x και η μήτρα (δισδιάστατος πίνακας) τιμών A . Ο πυρήνας DMV υπολογίζει το διάνυσμα εξόδου $y = Ax$. 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. 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 για τους επεξεργαστές γραφικών χρησιμοποιώντας το προγραμματιστικό περιβάλλον CUDA¹, 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 (σεριακό και OpenMP).
2. Γιατί ο κώδικάς σας είναι απογοητευτικά αργός;

¹Εάν κάποιος το επιθυμεί, μπορεί να υλοποιήσει την άσκηση και σε OpenCL. Ο λόγος που χρησιμοποιήσαμε CUDA οφείλεται στο γεγονός ότι έχει πιο απλό και ευθές API.

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 χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) και την επίδοση του κώδικά σας.
3. Ποιες από τις προσβάσεις στην κύρια μνήμη συνενώνονται τώρα και ποιες όχι σε μια αρχιτεκτονική με compute capability (α') 1.0, (β') ≥ 1.2 και (γ') 2.0;

Χρήση της τοπικής on-chip μνήμης

Χρησιμοποιείτε 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.

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 χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) και την επίδοση του κώδικά σας.
3. Παρατηρείτε κάποια συσχέτιση του χρόνου εκτέλεσης με το μέγεθος του μπλοκ νημάτων και του πλήθους των μπλοκ; Δώστε μία σύντομη εξήγηση.

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 δίνετε πλήρης και λειτουργικός σκελετός του κώδικα της άσκησης, καθώς και οι ρουτίνες της σειριακής και παράλληλης εκτέλεσης του DMV με OpenMP. Ο κώδικας που σας παρέχετε αποτελείται από τέσσερα (4) μέρη:

Κυρίως πρόγραμμα: Πρόκειται για το αρχείο `dmv_main.cu`, 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_gru.cu`, το οποίο περιέχει τις υλοποιήσεις των πυρήνων για GPUs.

Βοηθητικές συναρτήσεις: Πρόκειται για τα αρχεία `alloc.c`, `error.c`, `timer.c` και `gru_util.cu`, τα οποία περιέχουν βοηθητικές συναρτήσεις για δυναμική παραχώρηση μνήμης διαστάτου πίνακα, χειρισμού λαθών, χρονομέτρησης και χειρισμού των GPUs, αντίστοιχα.

Σας παρέχετε επιπλέον και το κατάλληλο `Makefile` για την μεταγλώττιση και την σύνδεση του κώδικά σας. Πληκτρολογήστε `make help`, 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 επί της ουσίας στα αρχεία `dmv_gru.cu` και `dmv_main.cu`. 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.

3.2 Περιβάλλον και διαδικασία ανάπτυξης

Στο εργαστήριο

Στο εργαστήριο η διαδικασία ανάπτυξης της άσκησης χωρίζεται σε δύο φάσεις:

1. Στην φάση ανάπτυξης και αποσφαλμάτωσης (debugging) και
2. στην φάση δοκιμών και πειραμάτων στους επεξεργαστές γραφικών.

Κατά την διάρκεια της πρώτης φάσης μπορείτε να δουλεύετε στα clones (όπως στις προηγούμενες ασκήσεις), όπου υπάρχει εγκατεστημένο το περιβάλλον ανάπτυξης CUDA 2.3. Ωστόσο, τα clones δεν έχουν κάρτα γραφικών, οπότε θα πρέπει να μεταγλωττίζετε και να εκτελείτε κατάλληλα τον κώδικά σας για λειτουργία εξομοίωσης (emulation mode). Αυτό μπορείτε να το πετύχετε εύκολα σε κάποιο clone, εκτελώντας απλά `make EMU=1`. Ο λόγος ύπαρξης αυτής της φάσης είναι καθαρά για έλεγχο λαθών στις υλοποιήσεις σας· οι επιδόσεις των πυρήνων για GPU σε λειτουργία εξομοίωσης δεν έχουν καμία σχέση με την πραγματικότητα, οπότε μην προσπαθήσετε να ερμηνεύσετε τα αποτελέσματα.

Κατά την δεύτερη φάση της υλοποίησης θα τρέξετε τον κώδικά σας σε μηχάνημα του εργαστηρίου (termis) με εγκατεστημένη κάρτα γραφικών τελευταίας γενιάς (nVidia Tesla M2050, αρχιτεκτονική Fermi). Για περισσότερες πληροφορίες σχετικά με τα λεπτομερή τεχνικά χαρακτηριστικά της GPU, πληκτρολογήστε `make query` όντας σε ένα μηχάνημα termi.

Στο σπίτι

Μπορείτε να δουλεύετε και στο δικό σας σύστημα είτε με χρήση της λειτουργίας εξομοίωσης, όπως στα clones, είτε –εάν έχετε CUDA-enabled κάρτα γραφικών– απευθείας στην κάρτα γραφικών σας. Εάν πρόκειται να χρησιμοποιήσετε την λειτουργία εξομοίωσης, ωστόσο, σας προτείνουμε να χρησιμοποιήσετε το περιβάλλον ανάπτυξης CUDA 2.3, καθότι στις επόμενες εκδόσεις η υποστήριξη της συγκεκριμένης λειτουργίας έχει εγκαταλειφθεί. Εναλλακτικά, θα πρέπει να δοκιμάσετε άλλου είδους προσομοιωτές επεξεργαστών γραφικών, π.χ., [Ocelot](#). Τέλος, εάν διαλέξετε κάποια δική σας πλατφόρμα για την ανάπτυξη του κώδικά σας, το πιθανότερο είναι να πρέπει να αλλάξετε κάποιες μεταβλητές του `Makefile` που σας δίνετε. Για περισσότερες πληροφορίες, μπορείτε να επικοινωνήσετε με τους βοηθούς του εργαστηρίου.

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 Ki², 2 Ki, 4 Ki, 7 Ki, 8 Ki, 14 Ki και 16 Ki.

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 περιβάλλοντος `GOMP_CPU_AFFINITY=0-23`. Με αυτό τον τρόπο³ «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 τρεις εκδόσεις πυρήνων (`naive`, `coalesced` και `shmem`) που υλοποιήσατε.

Για την διευκόλυνσή σας, σας δίνετε ένα πρωτότυπο script μετρήσεων τόσο για τις μετρήσεις με GPUs όσο και αυτών με OpenMP. 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 χρησιμοποίηση ολόκληρης της GPU από την επιλογή του μπλοκ;

ΠΡΟΣΟΧΗ: Στο Makefile που σας δίνετε, 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`.

4.2 Χρήση υπολογιστών εργαστηρίου

Η χρήση των υπολογιστών του εργαστηρίου (ουρές `clones` και `termis`) για την εκτέλεση της άσκησης θα γίνεται μέσω του συστήματος υποβολής εργασιών `Torque`, όπως και στις προηγούμενες ασκήσεις.

²1 Ki = 1024 bytes.

³Ο 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.