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
2013–2014

Άσκηση 3:
Παράλληλος προγραμματισμός σε επεξεργαστές γραφικών

Προθεσμίες παράδοσης

Επίδειξη προγραμμάτων	*Ημ/νία εξέτασης*
Τελική αναφορά	*Πέρας εξεταστικής*

1 Εύρεση ελαχίστων μονοπατιών σε γράφο

Το πρόβλημα της εύρεσης των ελαχίστων μονοπατιών μεταξύ όλων των κόμβων ενός γράφου (All-Pairs Shortest Path – APSP) εμφανίζεται σε σειρά εφαρμογών που κάνουν χρήση αναπαραστάσεων γράφων, όπως δίκτυα επικοινωνιών, μεταφορών κ.ά. Δεδομένου ενός σταθμισμένου κατευθυνόμενου γράφου $G = (V, E)$ με συνάρτηση βάρους $w : E \to \mathbb{R}$, το πρόβλημα APSP ορίζεται ως η ανεύρεση του μονοπατιού με το ελάχιστο βάρος για κάθε ζεύγος $u, v \in V$. Ως βάρος ενός μονοπατιού $u \rightsquigarrow v$ ορίζεται το άθροισμα των βαρών των ακμών που απαρτίζουν το μονοπάτι. Η λύση του προβλήματος απαιτείται συνήθως σε μορφή πίνακα $|V| \times |V|$, όπου η εγγραφή στην γραμμή u και στήλη v αποτελεί το βάρος του ελαχίστου μονοπατιού από τον κόμβο u στον v.

1.1 Ο αλγόριθμος Floyd-Warshall

Ο αλγόριθμος Floyd-Warshall επιλύει το πρόβλημα APSP βασιζόμενος στην ιδέα του δυναμικού προγραμματισμού σε χρόνο $\Theta(|V|^3)$. Επιτρέπονται ακμές με αρνητικό βάρος, αρκεί να μην υπάρχουν κύκλοι με συνολικό αρνητικό βάρος. Η ιδέα του αλγορίθμου Floyd-Warshall βασίζεται στην παρατήρηση ότι εάν ένας κόμβος k βρίσκεται στο ελάχιστο μονοπάτι p_{ij} από τον κόμβο i στον κόμβο j, τότε και τα μονοπάτια p_{ik} και p_{kj} είναι ελάχιστα. Η απλή μορφή του Floyd-Warshall παρουσιάζεται στον Αλγόριθμο 1, όπου W είναι ο αρχικός πίνακας γειτνίασης του γράφου G, ο οποίος στο πέρας του αλγορίθμου περιέχει τις ελάχιστες αποστάσεις μεταξύ όλων των κόμβων.

1.2 Θέματα επίδοσης

Η πολυπλοκότητα του αλγορίθμου Floyd-Warshall είναι σημαντική και η απλοϊκή του μορφή (Αλγόριθμος 1) δυσχεραίνει την εκτέλεσή του σε σύγχρονες αρχιτεκτονικές με κρυφή μνήμη, παρά την ύπαρξη χρονικής τοπικότητας (επαναλήξεις για κάθε k), καθώς το σχήμα πρόσβασης στον πίνακα W δεν ευνοεί την χωρική τοπικότητα. Για τον λόγο αυτό έχει προταθεί η υλοποίηση με χρήση ψηφίδων (tiles), η οποία επιτυγχάνει σημαντική βελτίωση στην επίδοση του αλγορίθμου σε σύγχρονες αρχιτεκτονικές υπολογιστών. Ο Αλγόριθμος 2 παρουσιάζει την «tiled» εκδοχή του αλγορίθμου Floyd-Warshall.

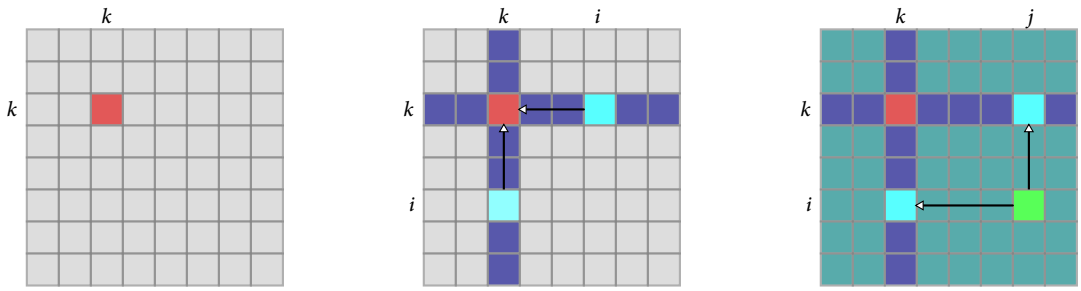
```
1: procedure FloydWarshall(W::inout, N::in)
   W: N × N πίνακας γειτνίασης
   N: πλήθος κόμβων
2:     for k ← 1 to N do
3:         for i ← 1 to N do
4:             for j ← 1 to N do
5:                 w_ij ← min(w_ij, w_ik + w_kj)
6:             end for
7:         end for
8:     end for
```

Αλγόριθμος 1: Ο αλγόριθμος Floyd-Warshall για την επίλυση του προβλήματος APSP.

```
1: procedure FWI(A::inout, B::in, C::in, N::in)
   A, B, C: N × N υποπίνακες γειτνίασης
   N: πλήθος κόμβων
2:     for k ← 1 to N do
3:         for i ← 1 to N do
4:             for j ← 1 to N do
5:                 a_ij ← min(a_ij, b_ik + c_kj)
6:             end for
7:         end for
8:     end for

9: procedure FloydWarshallTiled(W::inout, N::in, B::in)
   W: N × N πίνακας γειτνίασης
   N: πλήθος κόμβων
   B: μέγεθος «tile»
10:    T ← εξαγωγή «tiles»
11:    for k ← 1 to ⌈N/B⌉ do
12:        FWI(T_kk, T_kk, T_kk, B)
13:        for i ← 1 to ⌈N/B⌉ do
14:            if i ≠ k then
15:                FWI(T_ik, T_ik, T_kk, B)
16:                FWI(T_ki, T_kk, T_ki, B)
17:        end for
18:        for i ← 1 to ⌈N/B⌉ do
19:            if i = k then
20:                continue
21:            for j ← 1 to ⌈N/B⌉ do
22:                if j ≠ k then
23:                    FWI(T_ij, T_ik, T_kj, B)
24:            end for
25:        end for
26:    end for
```

Αλγόριθμος 2: Η «tiled» έκδοση του αλγορίθμου Floyd-Warshall. Ο υπολογισμός χωρίζεται σε τρεις φάσεις, κάθε μία από τις οποίες μπορεί να εκτελεστεί παράλληλα. Η συνάρτηση FWI υπολογίζει το APSP για τον πίνακα A βασιζόμενο στις τιμές των B και C. Προφανώς, FWI(W,W,W,N) = FloydWarshall(W,N).



(α′) Πρώτη φάση (β′) Δεύτερη φάση (γ′) Τρίτη φάση

Σχήμα 1: Οι τρεις φάσεις της «tiled» έκδοσης του αλγορίθμου Floyd-Warshall.

Η υλοποίηση του Floyd-Warshall με χρήση «tiles» χωρίζεται σε τρεις φάσεις, ο οποίες απεικονίζονται σχηματικά στο Σχήμα 1:

1. υπολογισμός του APSP για το «tile» T_{kk},
2. υπολογισμός του APSP για τα «tiles» T_{ik} και T_{kk} βάσει του T_{kk},
3. υπολογισμός του APSP για τα υπόλοιπα «tiles» T_{ij} βάσει των T_{ik} και T_{kj}.

Με εξαίρεση την πρώτη φάση, το μέγεθος του συνόλου εργασίας του αλγορίθμου Floyd-Warshall σε κάθε φάση είναι ίσο με το μέγεθος τριών «tiles». Εάν φροντίσουμε το μέγεθος των «tiles» έτσι, ώστε τρία από αυτά να χωρούν στην κρυφή μνήμη L1 του επεξεργαστή, θα έχουμε πετύχει σημαντική βελτίωση της χωρικής τοπικότητας του αλγορίθμου με αντίστοιχα ωφέλη και στην επίδοσή του.

1.3 Παραλληλοποίηση

Η παραλληλοποίηση του απλού αλγορίθμου Floyd-Warshall είναι αρκετά ευθεία, καθώς οι δύο εσωτερικοί βρόχοι μπορούν να εκτελεστούν παράλληλα με τον υπολογισμό να συγχρονίζεται στο τέλος κάθε επανάληψης του εξωτερικού βρόχου.

Η παραλληλοποίηση της «tiled» έκδοσης του αλγορίθμου είναι και αυτή αρκετά εύκολη, αρκεί να τηρηθεί η σειρά των φάσεων που περιγράψαμε προηγουμένως. Αυτο απαιτεί ένα καθολικό συγχρονισμό των νημάτων (barrier) μετά το πέρας κάθε φάσης.

2 Υλοποίηση Floyd-Warshall σε επεξεργαστές γραφικών (GPUs)

Στην άσκηση αυτή θα πρέπει να υλοποιήσετε τον αλγόριθμο Floyd-Warshall 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 για τους επεξεργαστές γραφικών.

2.1 Βασική υλοποίηση

Υλοποιήστε τον απλή έκδοση του αλγορίθμου Floyd-Warshall (Αλγόριθμος 1) για τους επεξεργαστές γραφικών χρησιμοποιώντας το προγραμματιστικό περιβάλλον CUDA[1]. Πειραματιστείτε με μονοδιάστατα και δισδιάστατα μπλοκ και σχολιάστε την επίδοση.

- Εκτελέστε το πρόγραμμά σας για διάφορα μεγέθη γράφων εισόδου, με πλήθος ακμών από 1024 έως 8192. Πώς διακυμαίνεται η επίδοση;
- Πόσο καλύτερος είναι ο κώδικάς σας από την «tiled» υλοποίηση του Floyd-Warshall με OpenMP?

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

2.2 Tiled υλοποίηση

Υλοποιήστε την «tiled» εκδοχή του αλγορίθμου Floyd-Warshall (Αλγόριθμος 2) για τους επεξεργαστές γραφικών κάνοντας χρήση μόνο της κύριας μνήμης.

- Πόσοι διαφορετικοί πυρήνες απαιτούνται τουλάχιστον για την υλοποίηση;
- Εκτελέστε το πρόγραμμά σας για διάφορα μεγέθη γράφων εισόδου, με πλήθος ακμών από 1024 έως 8192. Πώς διακυμαίνεται η επίδοση;
- Πόσο καλύτερος είναι τώρα ο κώδικάς σας από την «tiled» υλοποίηση του Floyd-Warshall με OpenMP?

2.3 Χρήση της μοιραζόμενης on-chip μνήμης

Υλοποιήστε την «tiled» εκδοχή του αλγορίθμου Floyd-Warshall (Αλγόριθμος 2) για τους επεξεργαστές γραφικών κάνοντας χρήση της μοιραζόμενης on-chip μνήμης ανά SM, ώστε να μειώσετε τις προσβάσεις στην κύρια μνήμη.

- Ποια δεδομένα απαιτούνται σε κάθε φάση για τον υπολογισμό του APSP?
- Ποιο είναι το ιδανικό μέγεθος «tile»; Τι περιορισμοί προκύπτουν;
- Εκτελέστε το πρόγραμμά σας για διάφορα μεγέθη γράφων εισόδου, με πλήθος ακμών από 1024 έως 8192. Πώς διακυμαίνεται η επίδοση;
- Πόσο καλύτερος είναι τώρα ο κώδικάς σας από την «tiled» υλοποίηση του Floyd-Warshall με OpenMP?

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 των διαφορετικών εκδόσεων Floyd-Warshall με χρήση OpenMP. Ο κώδικας που σας παρέχετε αποτελείται από τέσσερα (4) μέρη:

Κυρίως πρόγραμμα: Πρόκειται για το αρχείο `apsp_main.c`, 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, (β′) για την δημιουργία του γράφου εισόδου, (γ′) για την εκτέλεση και χρονομέτρηση του εκάστοτε πυρήνα (CPU ή GPU), (δ′) για τον έλεγχο της εγκυρότητας των αποτελεσμάτων.

Υλοποίηση του γράφου: Πρόκειται για τα αρχεία `graph.h` και `graph.c`. Περιέχουν όλες τις λειτουργίες σχετικά με την δομή του γράφου (κατασκευή, αντιγραφή, καταστροφή κ.λπ.), καθώς επίσης και τις υλοποιήσεις (απλή και «tiled») του Floyd-Warshall για τις CPUs.

Πυρήνες για CPU: Πρόκειται για το αρχείο `cpu_kernels.c`, το οποίο υλοποιεί απλά την κοινή διεπαφή κλήσης πυρήνων (βλ. `apsp_kernel_fn_t` στο αρχείο `graph.h`). Εσωτερικά, καλεί τις κατάλληλες ρουτίνες που ορίζονται στο `graph.c`.

Πυρήνες για GPU: Πρόκειται για το αρχείο `gpu_kernels.cu`, το οποίο περιέχει τις υλοποιήσεις των πυρήνων για GPUs σε συνδυασμό με ό,τι απαραίτητο χρειάζεται για την κλήση τους (καταχώρηση μνήμης στην GPU, αντιγραφή από/προς την GPU, ρύθμιση παραμέτρων πυρήνα, κ.λπ.).

Βοηθητικές συναρτήσεις: Πρόκειται για τα αρχεία `alloc.c`, `error.c`, `timer.c` και `gpu_util.cu`, τα οποία περιέχουν βοηθητικές συναρτήσεις για δυναμική παραχώρηση μνήμης δισδιάστατου 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. Πληκτρολογείστε '`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 μόνο στο αρχείο `gpu_kernels.cu`. Τέλος, για να δείτε τον τρόπο χρήσης του τελικού εκτέλεσιμου, εκτελέστε '`./apsp_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 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.

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 λειτουργίας εξομοίωσης, ωστόσο, σας προτείνουμε να χρησιμοποιήσετε το περιβάλλον ανάπτυξης CUDA 2.3, καθότι στις επόμενες εκδόσεις η υποστήριξη της συγκεκριμένης λειτουγίας 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 Διαδικασία μετρήσεων

Σκοπός των μετρήσεων της άσκησης αυτής είναι η σύγκριση της επίδοσης των τριών εκδόσεων του APSP για GPUs με την έκδοση «tiled» για CPUs. Η κατάλληλη επιλογή του μεγέθους των «tiles» για τις εκδόσεις των GPUs, καθώς και τα μεγέθη και σχήματα των μπλοκ νημάτων αφήνονται ελεύθερα για δικό σας πειραματισμό. Επιλέξατε αυτά που σας δίνουν την καλύτερη επίδοση και προσπαθήστε να εξηγήσετε τους λόγους πίσω από τυχόν διαφορές επίδοσης που παρατηρήσατε.

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.

4.2 Μεταγλώττιση

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`'.

Προσοχή: Όταν απενεργοποιείτε την λειτουργία debug, απενεργοποιείται και ο έλεγχος του αποτελέσματος του πυρήνα για λόγους συντομίας. Επομένως, προτού τρέξετε την τελική έκδοση, θα πρέπει να έχετε σιγουρευτεί ότι ο πυρήνας σας έχει το σωστό αποτέλεσμα.

4.3 Χρήση υπολογιστών εργαστηρίου

Η χρήση των υπολογιστών του εργαστηρίου (ουρά clones) 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 parlab -l nodes=1:ppn=24:cuda myjob.sh
```