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
2010 – 2011

Άσκηση 2:
Παράλληλος προγραμματισμός σε επεξεργαστές γραφικών

Προθεσμίες παράδοσης

Επίδειξη προγραμμάτων	*21 Δεκ. 2010*
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 $\mathbf{x}$ 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 γενικού σκοπού (CPUs)

Στο βήμα αυτό της άσκησης θα πρέπει να υλοποίησετε δύο εκδόσεις του αλγορίθμου DMV για επεξεργαστές γενικού σκοπού:

1. Σειριακό πρόγραμμα.
2. Παράλληλο πρόγραμμα για αρχιτεκτονικές μοιραζόμενης μνήμης με χρήση OpenMP.

Το βήμα αυτό θα σας χρησιμέψει περισσότερο ως σημείο αναφοράς και σύγκρισης με την υλοποίηση για τους επεξεργαστές γραφικών. Δεν απαιτείται κάποια βελτιστοποίηση ούτε στην σειριακή ούτε στην παράλληλη έκδοση.

2.2 Υλοποίηση για επεξεργαστές γραφικών (GPUs)

Σε αυτό το βήμα, 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 συγκεκριμένους επεξεργαστές γραφικών.

Βασική υλοποίηση

Υλοποιήστε τον αλγόριθμο DMV για τους επεξεργαστές γραφικών χρησιμοποιώντας το προγραμματιστικό περιβάλλον CUDA, 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.
2. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων και καταγράψετε την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) και την επίδοση του κώδικά σας.
3. Γιατί ο κώδικάς σας είναι απογοητευτικά αργός;

Συνένωση των προσβάσεων στην κύρια μνήμη (memory access coalescing)

Τροποποιήστε 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 του κώδικα για την CPU.
2. Ποιες από τις προσβάσεις στην κύρια μνήμη συνενώνονται τώρα και ποιες όχι;

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 εκτελείται τους υπολογισμούς σε αυτό. Θα πρέπει να προσέξετε, οι προσβάσεις στην κύρια μνήμη για την προφόρτωση του **x** να μπορούν να συνενωθούν, ώστε μπορέσετε να αποκομίσετε οφέλη από αυτή την βελτιστοποίηση.

1. Καταγράψετε την επίδοση και συγκρίνετέ την με την επίδοση του κώδικα για την CPU.
2. Πειραματιστείτε με διάφορα μεγέθη μπλοκ νημάτων και καταγράψετε την χρησιμοποίηση των πολυεπεξεργαστικών στοιχείων (SMs) και την επίδοση του κώδικά σας.
3. Πώς συσχετίζεται ο χρόνος εκτέλεσης με το μέγεθος του μπλοκ νημάτων και του πλήθους των μπλοκ; Δώστε μία σύντομη εξήγηση.

3 Υποδείξεις και διευκρινίσεις

3.1 Παράμετροι εισόδου, αρχικοποιήσεις και έλεγχοι αποτελεσμάτων

1. Ο αλγόριθμός σας αρκεί να χειρίζεται τετραγωνικούς πίνακες $N \times N$, όπου το μέγεθος N θα πρέπει να δίνεται ως παράμετρος από την γραμμή εντολών.
2. Η υλοποίηση για τους επεξεργαστές γραφικών θα πρέπει να δέχεται ως είσοδο και το μέγεθος του μπλοκ νημάτων. Χρησιμοποιήστε μονοδιάστατα μπλοκ νημάτων και μονοδιάστατο πλέγμα. Το μέγεθος του πλέγματος θα καθορίζεται από το μέγεθος του προβλήματος και από το μέγεθος του μπλοκ ως εξής
$$G_s = \lceil \frac{N}{B_s} \rceil,$$
όπου G_s και B_s είναι τα μεγέθη του πλέγματος και του μπλοκ, αντίστοιχα, ενώ N είναι το μέγεθος του προβλήματος.
3. Τα στοιχεία του πίνακα **A** και του διανύσματος εισόδου **x** θα πρέπει να αρχικοποιούνται με ψευδοτυχαίους αριθμούς στο διάστημα $[0, M)$, όπου το M είναι παράμετρος της μεταγλώττισης. Για να έχετε επαναληψιμότητα των αποτελεσμάτων ανάμεσα σε διαφορετικές εκτελέσεις του κώδικά σας, θα πρέπει να αρχικοποιήσετε την γεννήτρια των ψευδοτυχαίων αριθμών που θα χρησιμοποιήσετε με σταθερό φύτρο (δείτε `srand()`, `srand48()`).
4. Όλοι οι υπολογισμοί θα πρέπει να γίνουν με αριθμούς κινητής υποδιαστολής απλής ακρίβειας (`float`).

5. Σε κάθε βήμα της άσκησης είναι σημαντικό να ελέγχετε το αποτέλεσμα των υπολογισμών στην κάρτα γραφικών με το αποτέλεσμα στην CPU, ώστε να αναγνωρίσετε από νωρίς τυχόν λάθη στην υλοποίηση του αλγορίθμου. Δύο διανύσματα εξόδου y και y'θα πρέπει να θεωρούνται ίσα, όταν ισχύει το παρακάτω:

$$|y'_i - y_i| < \epsilon, \forall i \in [0, N], \epsilon > 0.$$

Επειδή οι GPUs που διατίθενται στο εργαστήριο δεν διαθέτουν πλήρη υποστήριξη του προτύπου IEEE 754 για τους αριθμούς κινητής υποδιαστολής, ειδικά όσον αφορά τα σφάλματα στρογγυλοποίησης, χρησιμοποιήστε σχετικά μεγάλο ϵ.

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 twins, όπου υπάρχει εγκατεστημένο το περιβάλλον ανάπτυξης CUDA 2.3. Ωστόσο τα twins δεν έχουν κάρτα γραφικών, οπότε θα πρέπει να μεταγλωττίζετε και να εκτελείτε κατάλληλα τον κώδικά σας για λειτουργία εξομοίωσης (emulation mode). Παρακάτω δίνεται ένα παράδειγμα μεταγλώττισης και σύνδεσης ενός προγράμματος που καλεί τον πυρήνα DMV για την GPU. Υποθέτουμε ότι το αρχείο `dmv_gpu.cu` περιέχει τον κώδικα του υπολογιστικού πυρήνα που πρόκειται να τρέξει στην GPU, ενώ το `dmv_main.cu` περιέχει την `main()` και καλεί τον πυρήνα για την GPU[1].

```
bkk@twin7 dmv$ nvcc -c -O3 -ccbin /usr/bin/gcc-4.3 -deviceemu \
> -I/usr/local/cuda/include dmv_gpu.cu
bkk@twin7 dmv$ nvcc -c -O3 -ccbin /usr/bin/gcc-4.3 -deviceemu \
> -I/usr/local/cuda/include dmv_main.cu
```

Είναι σημαντικό κατά την διαδικασία της μεταγλώττισης να δηλώσουμε στον μεταγλωττιστή με τον διακόπτη '`-I`' πού να ψάξει να βρει τα αρχεία επικεφαλίδας του περιβάλλοντος ανάπτυξης (`cuda.h`), ενώ με τον διακόπτη '`-ccbin`' δηλώνουμε στον `nvcc` να χρησιμοποιήσει τον GCC 4.3 ως back-end compiler και όχι τον GCC 4.4 που είναι ο προεπιλεγενος GCC compiler στα twins. Αυτό είναι απαραίτητο, ώστε να μεταγλωττιστεί σωστά ο κώδικάς σας. Τέλος, ο διακόπτης '`-deviceemu`' παράγει κώδικα για λειτουργία εξομοίωσης. Η διαδικασία της σύνδεσης μπορεί να γίνει με τον συνήθη τρόπο με χρήση του GCC:

```
bkk@twin7 dmv$ gcc-4.3 -L/usr/local/cuda/lib64 -lcudart \
> -o dmv_main dmv_main.o dmv_gpu.o
```

Για να αποφύγετε την διαδικασία να θυμάστε όλους αυτούς τους διακόπτες κατά την μεταγλώττιση και την σύνδεση, σας προτείνουμε να χρησιμοποιήσετε `Makefile`.

Κατά την δεύτερη φάση της υλοποίησης θα τρέξετε τον κώδικά σας σε μηχάνημα του εργαστηρίου (`warp.cslab.ece.ntua.gr`) με εγκατεστημένες δύο κάρτες γραφικών nVidia GeForce 8800 Ultra. Θα πρέπει να μεταγλωττίσετε ξανά τον κώδικά σας χωρίς τον διακόπτη '`-deviceemu`' αυτή την φορά. Επίσης, στην φάση αυτή, θα χρειαστείτε και τον διακόπτη `--ptxas-options=-v`, με τον οποίο θα τυπωθεί η χρήση καταχωρητών (register usage) για κάθε πυρήνα που πρόκειται να τρέξει στην GPU. Η πληροφορία αυτή είναι χρήσιμη, ώστε να υπολογίσετε τον βαθμό χρησιμοποίησης των SMs (δείτε επίσης το CUDA Occupancy calculator).

[1] Υποθέτουμε ότι το '$' είναι το πρωτεύον (primary) prompt, ενώ το '>' το δευτερεύον (secondary).

Στο σπίτι

Μπορείτε να δουλεύετε και στο δικό σας σύστημα είτε με χρήση της λειτουργίας εξομοίωσης, όπως στα twins, είτε –εάν έχετε CUDA-enabled κάρτα γραφικών– απευθείας στην κάρτα γραφικών σας. Εάν πρόκειται να χρησιμοποιήσετε την χρήση λειτουργίας εξομοίωσης, ωστόσο, σας προτείνουμε να χρησιμοποιήσετε το περιβάλλον ανάπτυξης CUDA 2.3, καθότι στις επόμενες εκδόσεις η υποστήριξη της συγκεκριμένης λειτουγίας έχει εγκαταλειφθεί[2]. Εναλλακτικά, θα πρέπει να δοκιμάσετε άλλου είδους προσομοιωτές επεξεργαστών γραφικών, π.χ., Ocelot. Τέλος, εάν χρησιμοποιήσετε το περιβάλλον CUDA 2.3, θα πρέπει να έχετε εγκατεστημένο και τον GCC 4.3 και να ακολουθήσετε την διαδικασία μεταγλώττισης που περιγράφηκε για την εργασία στο εργαστήριο.

3.3 Πειράματα και μετρήσεις επιδόσεων

Για κάθε έκδοση του αλγορίθμου θα πρέπει να μετράτε τον χρόνο εκτέλεσης (σε s) μίας σειράς επαναλήψεων του αλγορίθμου, π.χ., 100 επαναλήψεις. Ο αριθμός αυτός θα είναι παράμετρος της μεταγλώττισης. Επίσης, θα πρέπει να υπολογίσετε και την επίδοση του αλγορίθμου σε χρήσιμα Gflop/s. Η επίδοση σε Gflop/s για τον συγκεκριμένο αλγόριθμο για K επαναλήψεις είναι

$$P = \frac{2N^2 \cdot K \cdot 10^{-9}}{t_K},$$

όπου N είναι το μέγεθος του προβλήματος και t_K ο χρόνος εκτέλεσης των K επαναλήψεων του αλγορίθμου.

Για την μέτρηση του χρόνου χρησιμοποιήστε την κλήση συστήματος `gettimeofday()` όπως φαίνεται στο παρακάτω ψευδοκώδικα:

```
gettimeofday(...);
for (i = 0; i < K; ++i) {
  my_gpu_kernel<<<G,B>>>(...);
  cudaThreadSynchronize();
}
gettimeofday(...);
```

[2]Η υποστήριξη της λειτουργίας εξομοίωσης εγκαταλήφθηκε στο CUDA 3.0, ενώ στο CUDA 3.1 έχει εξαλειφθεί η λειτουργία αυτή.