ΜΥΕ023-Παράλληλα Συστήματα και Προγραμματισμός

Σετ Ασκήσεων #2

Βασίλειος Βαλεράς

ΑΜ: 4031

E-mail: cs04031@uoi.gr

Το 2ο σετ ασκήσεων αφορά στον παράλληλο προγραμματισμό με CUDA. Στην 1η άσκηση ζητείται η χρήση του CUDA Runtime API με σκοπό την ανάκτηση πληροφοριών σχετικά με την συσκευή CUDA που μας δόθηκε ενώ η δεύτερη στην παραλληλοποίηση της συνάρτησης θόλωσης εικόνας που χρησιμοποιήσαμε και στην 1η άσκηση.

Οι μετρήσεις έγιναν στον σύστημα parallax που μας δόθηκε.

Άσκηση 1

Το πρόβλημα

Στην συγκεκριμένη άσκηση μας δίνεται ένα αρχείο cuinfo.cu το οποίο πρέπει να συμπληρώσουμε χρησιμοποιώντας το CUDA Runtime API με στόχο να ανακαλύψουμε πόσες και ποιες συσκευές CUDA υπάρχουν στο σύστημα καθώς και να εμφανίσουμε στην οθόνη πληροφορίες σχετικά με τις συσκευές που βρέθηκαν.

Η λύση

Εφόσον έχουμε κάνει include το cuda_runtime.h κάνουμε χρήση την συνάρτηση **cudaGetDeviceProperties** η οποία ανακτά τις ιδιότητες της συσκευής CUDA με δείκτη i, και τις αποθηκεύει στην μεταβλητή dev_prop.

Στην συνέχεια κάνουμε τα εξής print:

- Αρχικά τυπώνουμε το όνομα της συσκευής με δείκτη i, κάνοντας χρήση του πεδίου name της μεταβλητής dev_prop.
- Έπειτα με τα πεδία major, minor τυπώνουμε την έκδοση της συσκεής CUDA καθώς και το compute capability της.
- Στην συνέχεια με το πεδίο multiProcessorCount τυπώνουμε τον αριθμό των streaming multiprocessors(SMs) της συσκευής.
- Με το πεδίο maxThreadsPerBlock τυπώνουμε των αριθμό των νημάτων ανά block

- Με το πεδίο totalGlobalMem τυπώνουμε το συνολικό μέγεθος της καθολικής μνήμης
- Με το πεδίο sharedMemPerBlock τυπώνουμε το μέγεθος της κοινόχρηστης μνήμης ανά block

Στο τελευταίο μέρος της άσκησης θα δείξουμε πως μπορούμε να βρούμε το συνολικό αριθμό από cores μιας NVIDIA GPU. Όπως γνωρίζουμε μια συσκευή CUDA διαθέτει ένα πλήθος από SMs , και ο κάθε streaming multiprocessor διαθέτει έναν αριθμό από cores.

Για να βρούμε αυτόν τον συνολικό αριθμό από cores, ο υπολογισμός μας στηρίζεται στο compute capability της CUDA. Το compute capability αντιπροσωπεύει την αρχιτεκτονική και τις δυνατότητες της NVIDIA GPU.

Μέσα λοιπόν σε μια printf, υπάρχουν conditional statements και σε κάθε συνθήκη αντιστοιχεί μια συγκεκριμένη έκδοση υπολογιστικής ικανότητας. Για παράδειγμα αν το πεδίο major είναι 2 και το minor 1, το compute capability (υπολογιστική ικανότητα) της συσκευής είναι 2.1. Εάν η συνθήκη αυτή ισχύει, τότε αυτό σημαίνει ότι η συγκεκριμένη GPU ανήκει στην αρχιτεκτονική Fermi(SM 2.x). Ο αντίστοιχος αριθμός πυρήνων CUDA για την αρχιτεκτονική Fermi είναι: number of SMs * 48 (πρακτικά dev_prop.multiProcessorCount * 48).Με αυτόν τον τρόπο βρίσκουμε τον συνολικό αριθμό από cores.Ομοίως, οι άλλες συνθήκες υπολογίζουν τον συνολικό αριθμό από cores με βάση την συγκεκριμένη έκδοση compute capability. Εάν καμία συνθήκη δεν ταιρίαζει, τότε χρησιμοποιούμε το default (για πλεόν) ότι κάθε SM έχει 128 cores.Αυτό συμβαίνει για compute capability 8.1 και μετά.

Στην δικιά μας περίπτωση η υπολογιστική ικανότητα της συσκευής CUDA είναι 6.1 , όποτε κάθε streaming multiprocessor έχει 128 cores με αποτέλεσμα (αφού έχουμε 30 SMs) να έχουμε 3840 cores συνολικα.

Άσκηση 2

Το πρόβλημα

Στην συγκεκριμένη άσκηση σκοπός μας ήταν όπως και στην 1η άσκηση να υλοποιήσουμε μια εκδοχή της συνάρτησης gaussian blur, αλλά αυτή τη φορά η παραλληλοποίηση (οι υπολογισμοί) να γίνονται στην gpu.

Σαν γενικό πρόβλημα η συνάρτηση στην σειριακή εκδοχή της έχει, ότι εφαρμόζει τον αλγόριθμο χρησιμοποιώντας 4 for loops, το ένα μέσα στο άλλο. Τα 2 πρώτα για iteration σε κάθε pixel της εικόνας και τα άλλα 2 για την εφαρμογή του αλγορίθμου.

Μέθοδος παραλληλοποίησης

Η παράλληλη έκδοση της συνάρτησης gaussian blur στοχεύει στη βελτίωση της απόδοσης κατανέμοντας την εργασία σε πολλαπλές ομάδες και νήματα.

Ο αλγόριθμος χωρίζεται σε ομάδες με κάθε ομάδα να εκτελεί ένα μέρος της εργασίας. Γίνεται ανάθεση διαφορετικών επαναλήψεων του εξωτερικού βρόχου σε διαφορετικές ομάδες.

Ο εσωτερικός βρόχος υπεύθυνος για την επεξεργασία κάθε pixel, παραλληλίζεται χρησιμοποιώντας πολλαπλά νήματα σε κάθε ομάδα.

Αρχικά, χρησιμοποιούμε την οδηγία target ώστε η επεξεργασία να γίνει στην συσκευή CUDA. Έπειτα κάνουμε mapping τα δεδομένα εισόδου, με tofrom για την εικόνα εξόδου (καθώς θέλουμε όχι μόνο να μεταφέρουμε το struct στην μνήμη της CUDA αλλά και να επιστρέψουμε το πλέον ανανεωμένο struct με την εικόνα εξόδου στην CPU) και απλά με to για το struct με την εικόνα εισόδου.

Έπειτα στον εξωτερικό βρόχο χρησιμοποιούμε collapse(2), υποδεικνύοντας ότι τόσο ο εξωτερικός όσο και ο εσωτερικός βρόχος μπορούν να εκτελεστούν παράλληλα. Χρησιμοποιούμε private, ώστε κάθε νήμα να έχει το δικό του copy από k,m,wightsum,redsum,greensum και bluesum για ασφάλεια από race conditions.Τέλος με την οδηγία num_teams , thread_limit ορίζουμε τα νήματα και τις ομάδες.

Πειραματικά αποτελέσματα - μετρήσεις

Σύμφωνα με την εκφώνηση της άσκησης θα πρέπει να χρησιμοποιήσουμε όλους τους δυνατούς συνδυασμούς ομάδων και νημάτων , με την προϋπόθεση ότι χρησιμοποιούμε όλα τα cores στην CUDA και κάθε νήμα είναι πολλαπλάσιο του 32.

Έτσι όλοι οι δυνατοί συνδυασμοί είναι οι εξής:

- 120 ομάδες – 32 νήματα σε κάθε ομάδα
- 60 ομάδες – 64 νήματα σε κάθε ομάδα
- 48 ομάδες – 80 νήματα σε κάθε ομάδα
- 40 ομάδες – 96 νήματα σε κάθε ομάδα
- 30 ομάδες – 128 νήματα σε κάθε ομάδα

Έτσι κάθε πείραμα με συγκεκριμένο αριθμό νημάτων εκτελέστηκε 4 φορές και υπολογίστηκαν οι μέσοι όροι. Τα αποτελέσματα δίνονται στον παρακάτω πίνακα(οι χρόνοι σε sec):

	A	B	C	D	E	F	G
1	Threads	Teams	1st ex	2nd ex	3d ex	4th ex	AVG time
2	32	120	3,325022	3,317554	3,320937	3,320661	3,3210435
3	64	60	3,319588	3,322975	3,318633	3,315354	3,3191375
4	80	48	3,32556	3,323535	3,324913	3,331192	3,3263
5	96	40	3,327201	3,324633	3,330815	3,325811	3,327115
6	128	30	3,31838	3,319065	3,328538	3,319353	3,321334
7							

Η σειριακή εκτέλεση μετά από μετρήσεις έχει χρόνο 18,982146 seconds,

Να σημειωθεί ότι χωρίς να χρησιμοποιηθούν οι εντολές ανάθεσης αριθμού νημάτων και ομάδων η απόδοση είναι πολύ καλύτερη καθώς αφήνουμε το λειτουργικό να κάνει αξιοποίηση των πόρων(ακολουθούν εικόνες).

```
#pragma omp target map(tofrom: imgout->red[0:width*height], imgout->green[0:width*height], imgout->blue[0:width*height]) \
                    map(to: imgin->red[0:width*height], imgin->green[0:width*height], imgin->blue[0:width*height])
{
    #pragma omp teams
    {
        #pragma omp distribute parallel for collapse(2) private(k, m, weightSum, redSum, greenSum, blueSum)
        for (i = 0; i < height; i++)
        {
```

```
<<< Gaussian Blur (h=1500,w=1500,r=8) >>>
Total execution time (sequential): 0.000000
Total execution time (omp loops): 0.000000
Total execution time (omp tasks): 0.000000
Total execution time (omp device): 0.746532
[ex23016@parallax ~]$
```

Σχόλια

Συνολικά παρατηρούμε τεράστια βελτίωση στην παραλληλοποίηση από 19sec περίπου στα 3. Ειδικά χωρίς να πειράξουμε αριθμό threads και teams , έχουμε χρόνο θόλωσης εικόνας στα 0,75 sec!

Παρατηρούμε επίσης σχεδόν ίδιους χρόνους σε διαφορετικούς συνδυασμούς νημάτων και ομάδων. Αυτό μπορεί να οφείλεται σε ανισορροπία φόρτου ή περιορισμούς υλικού.