2 Σετ Παράλληλα Συστήματα και Προγραμματισμός

Μάθημα: Παράλληλα Συστήματα και Προγραμματισμός (ΜΥΕ023)
Διδάσκων: Δημακόπουλος Βασίλειος
Ονοματεπώνυμο: Καζακίδης Θεοχάρης
ΑΜ: 4679
Εξάμηνο: Εαρινό
Ακαδημαϊκό Έτος: 2022-2023

Το δεύτερο σετ ασκήσεων αφορά στο προγραμματισμό εφαρμογών σε επίπεδο GPU με την βοήθεια του συστήματος parallax. Ζητείται η αναζήτηση πληροφοριών για συσκευές CUDA που υπάρχουν στο σύστημα ,ακόμη ζητείται η θόλωση εικόνας (Gaussian Blur) μέσω της GPU με την συνάρτηση gaussian_blur_omp_gpu() με αριθμό νημάτων πολλαπλάσιο του 32 και αξιοποίηση όλων των πυρήνων της συσκευής.

Όλες οι μετρήσεις που θα παρουσιαστούν έγιναν στο παρακάτω σύστημα του parallax:

Όνομα υπολογιστή	**-**
Επεξεργαστής	**Intel(R) Xeon(R) Gold 6130 CPU @ 2.10GHz**
Πλήθος πυρήνων	**64**
Μεταφραστής	**gcc (GCC) 11.2.1**

Άσκηση 1

→Το πρόβλημα

Στην άσκηση αυτή ζητείται η αναζήτηση πληροφοριών για συσκευές CUDA που υπάρχουν στο σύστημα. Συγκεκριμένα μέσα σε ένα κενό αρχείο που ονομάζεται cuinfo.cu ,το οποίο χρησιμοποιεί CUDA Runtime API θα τροποποιηθεί με τέτοιο τρόπο ώστε να μας εμφανίζει τις ζητούμενες πληροφορίες μου αναφέρονται στην εκφώνηση, με σημαντικότερη πληροφορία να απευθύνεται για το πόσο είναι το πλήθος των πυρήνων. Οι πληροφορίες αυτές είναι οι εξής :

- ID Συσκευής
- Όνομα Συσκευής
- Υπολογισμός Ικανότητας CUDA
- Πλήθος SMs
- Μέγιστο Πλήθος Νημάτων Ανά Μπλοκ
- Συνολική Global Μνήμη
- Συνολική Shared Μνήμη Ανά Μπλοκ
- Εκτιμόμενο Συνολικό Πλήθος Πυρήνων

και έπειτα να τα εκτυπώσουμε στο τερματικό

→Κώδικας που προστέθηκε στο αρχείο cuinfo.cu

Ο κώδικας που προστέθηκε στο αρχείο cuinfo.cu συμπληρώθηκε μέσα στο μπλοκ της επανάληψης for και συγκεκριμένα στο σημείο TODO… :

```
for (i = 0; i < num_devs; i++)
{
        /* TODO: Retrieve and pretty-print all the necessary information */
}
```

```
for (i = 0; i < num_devs; i++)
{
        cudaGetDeviceProperties(&dev_prop, i);//1

        printf("ID Συσκευής: %d\n", i);//2
        printf("Όνομα Συσκευής: %s\n", dev_prop.name);//3
        printf("Υπολογισμός Ικανότητας CUDA: %d.%d\n", dev_prop.major,
dev_prop.minor);//4
        printf("Πλήθος SMs: %d\n", dev_prop.multiProcessorCount);//5
        printf("Μέγιστο Πλήθος Νημάτων Ανά Μπλοκ: %d\n",
dev_prop.maxThreadsPerBlock);//6
        printf("Συνολική Global Μνήμη: %f GB\n", dev_prop.totalGlobalMem /
(1024.0 * 1024.0 * 1024.0));//7
        printf("Συνολική Shared Μνήμη Ανά Μπλοκ: %f MB\n",
dev_prop.sharedMemPerBlock / (1024.0 * 1024.0));//8
        printf("Εκτιμόμενο Συνολικό Πλήθος Πυρήνων:
%d\n",dev_prop.multiProcessorCount * dev_prop.maxThreadsPerMultiProcessor *
dev_prop.warpSize);//9
}
```

Όπου συγκεκριμένα θα επεξηγήσω για κάθε //αριθμός τι κάνει η κάθε εντολή που προστέθηκε:

- //1: Συνάρτηση cudaGetDeviceProperties την οποία καλούμε, όπου ως ορίσματα έχει τον αριθμό i των συσκυών που είναι μέσα στην for και το struct dev_prop που περιέχει τα πεδία με τα properties των συσκευών

Gets the flags for the current device.

cudaGetDeviceProperties (cudaDeviceProp* prop , int device)
Returns information about the compute-device.

Οπου αυτή η εντολή βρέθηκε https://docs.nvidia.com/cuda/cuda-runtime-api/group__CUDART__DEVICE.html#group__CUDART__DEVICE_1g1bf9d625a931d657e08db2b4391170f0 όπου είναι μέσα στο documentation της NVIDIA για την CUDA TOOLKIT DOCUMENATION

- **//2: Το i της συσκευής που είναι μέσα στη for και εκτυπώνεται στην οθόνη από την printf.**
- **//3 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε ένα πεδίο του struct αυτού και συγκεκριμένα το όνομα της συσκευής dev_prop.name.**
- **//4 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε τα πεδίο του struct αυτού και συγκεκριμένα τους μέγιστους/ελάχιστους αριθμούς της συσκευής dev_prop.major/minor..**
- **//5 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε ένα πεδίο του struct αυτού και συγκεκριμένα το πλήθος των SM της συσκευής dev_prop. multiProcessorCount.**
- **//6 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε ένα πεδίο του struct αυτού και συγκεκριμένα το μέγιστο πλήθος νημάτων ανά μπλοκ της συσκευής dev_prop. maxThreadsPerBlock.**
- **//7 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε ένα πεδίο του struct αυτού και συγκεκριμένα συνολική global μνήμη της συσκευής και την διαιρώ με 1024*1024*1024 για να το εκτυπώσω σε GB dev_prop.totalGlobalMem.**
- **//8 Από το struct dev_prop που είναι όρισμα της συνάρτησης cudaGetDeviceProperties παίρνουμε ένα πεδίο του struct αυτού και συγκεκριμένα συνολική shared μνήμη της συσκευής και την διαιρώ με 1024*1024 για να το εκτυπώσω σε MB dev_prop.sharedMemPerBlock.**
- **//9 Δεν βρήκα κάποια συνάρτηση ή κάποιο πεδίο του struct που η CUDA να μας δίνει απευθείας το πλήθος πυρήνων της συσκευής. Άρα το μόνο που μπορεί να γίνει είναι μια εκτίμηση αυτού του πλήθους. Συγκεκριμένα ,μέσω του μεταξύ πολλαπλασιασμού του πλήθους SM(πεδίο dev_prop.multiProcessorCount) και του πλήθους νημάτων ανά SM(dev_prop.maxThreadsPerMultiProcessor)πεδίο παίρνουμε το μέγιστο αριθμό πυρήνων που μπορεί να έχει η GPU δηλαδή ένα σενάριο οπού όλα τα νήματα δέσμευσαν με ιδανικό τρόπο το μέγιστο που μπορούν σε πλήθος πυρήνων. Έτσι πολλαπλασίασα το γινόμενο που προαναφέρθηκε με των αριθμό warp του συστήματος των οποίο το βρήκα από το πεδίο dev_prop.warpSize και ο αριθμός που προκύπτει είναι μια εκτίμηση των πυρήνων ης συσκευής (Στην CUDA, οι ομάδες νημάτων με διαδοχικούς δείκτες νημάτων ομαδοποιούνται σε στρεβλώσεις(warp).Μία πλήρης στρέβλωση εκτελείται σε έναν μόνο πυρήνα CUDA. Κατά τον χρόνο εκτέλεσης, ένα μπλοκ νημάτων χωρίζεται σε έναν αριθμό warps για εκτέλεση στους πυρήνες ενός SM. Γενικότερα το μέγεθος ενός warp εξαρτάται από το υλικό.)**

→Τι τυπώνει?

```
[ex23003@parallax ~]$ nvcc cuinfo.cu -o cuinfo
[ex23003@parallax ~]$ ./cuinfo
 ID Συσκευής: 0
 Όνομα Συσκευής: Tesla P40
 Υπολογισμός Ικανότητας CUDA: 6.1
 Πλήθος SMs: 30
 Μέγιστο Πλήθος Νημάτων Ανά Μπλοκ: 1024
 Συνολική Global Μνήμη: 22.381958 GB
 Συνολική Shared Μνήμη Ανά Μπλοκ: 0.046875 MB
 Εκτιμόμενο Συνολικό Πλήθος Πυρήνων: 1966080
[ex23003@parallax ~]$
```

Άσκηση 2

→Το πρόβλημα

Στην άσκηση αυτή ζητείται συμπληρωθεί η συνάρτηση gaussian_blur_omp_gpu() ώστε η θόλωση της εικόνας να γίνεται στην GPU και ακόμη να πειραματιστούμε με διαφορετικό πλήθος νημάτων (πολλαπλάσιο 32) και η ομάδες νημάτων να είναι έχουν τέτοιο πλήθος ώστε να αξιοποιούν όλου τους πυρήνες της συσκευής.

→Κώδικας που προστέθηκε στο αρχείο gaussian_blur.c

```
#pragma omp target teams distribute parallel for num_threads(256) num_teams(15) private(i,j) collapse(2) map(to: imgin->red[:2250000], imgin->green[:2250000], imgin->blue[:2250000]) map(tofrom: imgout->red[:2250000], imgout->green[:2250000], imgout->blue[:2250000]
```

→Χρησιμοποιήθηκε το **target** για να εκτελεστεί η εντολή από την GPU ,**teams** για να δημιουργηθεί ομάδα νημάτων,**distribute** για διαμοιρασμό των επαναλήψεων των βρόγχων,private(i,j) ιδιωτικές μεταβλητές των νημάτων,**collapse** για την δημιουργία εμφωλευμένων βρόγχων σε έναν βρόγχο και **map** για την μεταφορά δεδομένων μεταξύ συσκευής και GPU(συγκεκριμένα περνάω τους πίνακες με τα χρώματα)

	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
SEQUENTIAL	22.212531	19.882209	18.934232	19.059115	20.022.022
ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΣΕ ΜΙΑ ΕΝΤΟΛΗ					
OMP GPU					
32 Νήματα /120 Ομάδες	1.018526	1.017754	1.010492	1.020032	1.016.701
64 Νήματα /60 Ομάδες	1.782111	1.772454	1.787598	1.816430	1.789.648
128 Νήματα /30 Ομάδες	3.402519	3.396708	3.404913	3.396265	3.400.101
256 Νήματα /15 Ομάδες	6.720053	6.679370	6.770347	6.897521	6.766.823


```
#pragma omp target map(to: imgin->red[:2250000], imgin->green[:2250000], imgin->blue[:2250000]) map(tofrom: imgout->red[:2250000], imgout->green[:2250000], imgout->blue[:2250000])
{
        #pragma omp teams num_teams(120)
        {
                #pragma omp distribute parallel for num_threads(32) private(i,j) collapse(2)
```

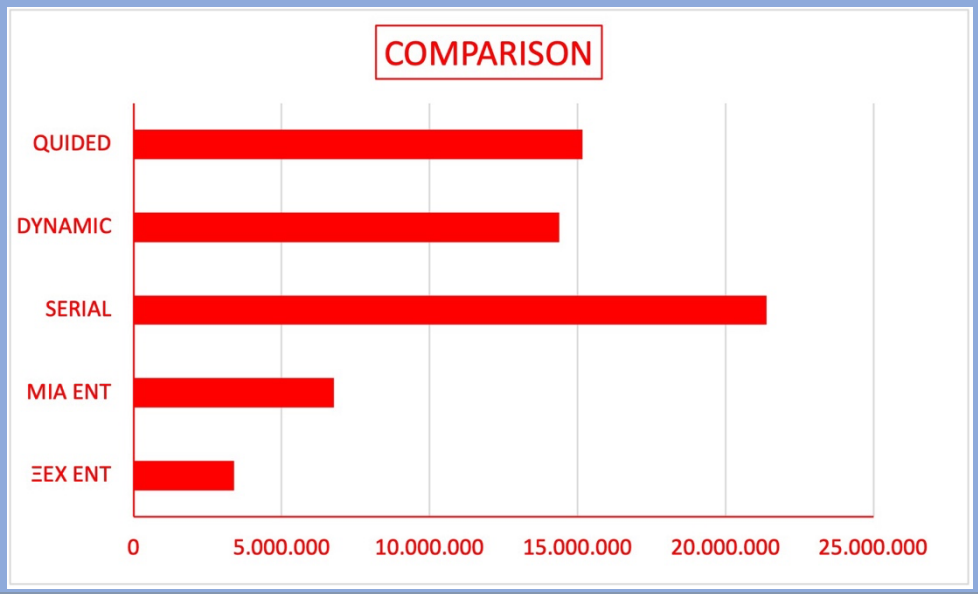
→ Χρησιμοποιήθηκε παραλληλοποίηση σε διαφορετικά block του κώδικα από έξω προς τα μέσα έτσι ώστε η παραλληλοποίηση να γίνεται σε διαφορετικά στάδια

	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
SEQUENTIAL	19.0111835	18.971288	19.138891	18.953911	19.018.818
ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΣΕ ΞΕΧΩΡΙΣΤΕΣ ΕΝΤΟΛΕΣ					
OMP GPU					
32 Νήματα /120 Ομάδες	3.391391	3.404792	3.391061	3.391878	3.394.781
64 Νήματα /60 Ομάδες	3.376572	3.396755	3.409402	3.407375	3.397.526
128 Νήματα /30 Ομάδες	3.396265	3.402519	3.396708	3.404913	3.400.101
256 Νήματα /15 Ομάδες	3.397426	3.382210	3.388512	3.372619	3.385.192


```
//TODO num_threads(128) num_teams(30)
//      num_threads(256) num_teams(15)
//      num_threads(64) num_teams(60)
//      num_threads(32) num_teams(120)
```

→Ομάδες νημάτων πολλαπλάσια του 32 και οι αντίστοιχες ομάδες για αξιοποίηση όλων των πυρήνων του συστήματος

ΣΕΙΡΙΑΚΟ 1ης ΑΣΚΗΣΗΣ	1η Εκτέλεση	2η Εκτέλεση	3η Εκτέλεση	4η Εκτέλεση	Μ.Ο. Εκτέλεσης
STATIC	21.338295	21.303755	21.661350	21.264037	21.3918592
DYNAMIC	14.469651	14.356198	14.365510	14.350689	14.385512
GUIDED	14.711575	15.265012	15.830573	14.897475	15.17615875
ΜΕΓΑΛΥΤΕΡΟΙ ΧΡΟΝΟΙ ΜΕ ΠΑΡΑΠΑΝΩ ΥΛΟΠΟΙΗΣΗ ΑΝΑ ΥΛΟΠΟΙΗΣΗ					
ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΣΕ ΜΙΑ ΕΝΤΟΛΗ(256 Νήματα /15 Ομάδες)	6.720053	6.679370	6.770347	6.897521	6.766.823
ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗ ΣΕ ΞΕΧΩΡΙΣΤΕΣ ΕΝΤΟΛΕΣ(128 Νήματα /30 Ομάδες)	3.396265	3.402519	3.396708	3.404913	3.400.101



→Συμπέρασμα

Παρατηρούμε ότι από τις μεταξύ εντολές τις παραπάνω υλοποίηση καλύτερο χρόνο γενικά δίνει η υλοποίηση με την παραλληλοποίηση σε ξεχωριστές εντολές. Γενικότερα όμως σε σύγκριση με την προηγούμενη άσκηση καλύτερο χρόνο δίνουν οι υλοποιήσεις τις δεύτερης ασκήσεως σε σχέση με τις σειριακές με ξεκάθαροι διαφορά στους χρόνους όπου φαίνεται και στο σχήμα παραπάνω. Έτσι συμπεραίνουμε ότι τα προγράμματα που τρέξαμε στη GPU είχαν πολύ καλύτερο χρόνο άρα η GPU είναι κατά πολύ καλύτερη.