Συστήματα Υπολογισμού Υψηλών Επιδόσεων (ECE 415)
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Πανεπιστήμιο Θεσσαλίας

5η Εργαστηριακή Άσκηση

Στοιχεία φοιτητών:

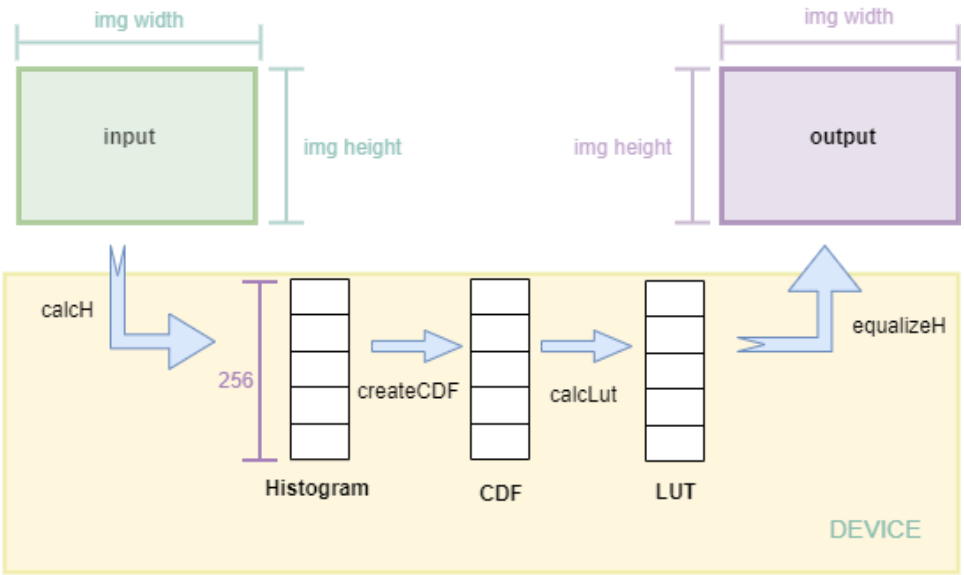
Μπαλτάς Νικόλαος, 2757
Ξωχέλλη Ελένη, 2761

0.Στρατηγική Υλοποίησης

Χωρίσαμε την διαδικασία της εξίσωσης του ιστογράμματος σε 4 μέρη:

1. Υπολογισμός του ιστογράμματος(CalcH)
2. Υπολογισμός της συνάρτησης κατανομής (πίνακας CDF)
3. Υπολογισμός του LUT (lookup table)
4. Εξίσωση του ιστογράμματος(equalize Histogram)

Κάθε ένα από τα μέρη αντιστοιχεί σε μία συνάρτηση και ένα kernel. Όπου ήταν εφικτό χρησιμοποιήθηκε shared memory, registers, streams και το σύνολο των πράξεων γίνεται στο device. Το διάγραμμα απεικονίζει την λογική της υλοποίησης.



Στην πρώτη και τελευταία φάση για να μπορούμε να επεξεργαστούμε εικόνες που ξεπερνούν το μέγιστο μέγεθος grid που μπορούμε να υποστηρίξουμε χρησιμοποιούμε grid - strided loops. Μέσα στο loop μας κινούμαστε κατά **blockDim.x * gridDim.x** που είναι ο συνολικός αριθμός των thread μέσα στο grid.

1. Υπολογισμός Ιστογράμματος – CalcH

Χρησιμοποιώντας την αρχική εικόνα σαν input , μετράμε το πόσες φορές εμφανίζεται η κάθε τιμή φωτεινότητας και δημιουργούμε το ιστόγραμμα. Καθώς υπολογίζουμε το πλήθος των φορών που εμφανίζεται η κάθε τιμή φωτεινότητας ,υπάρχει περίπτωση κάποιο/κάποια thread να προσπαθήσουν να αυξήσουν την ίδια τιμή σε κάποια θέση του πίνακα, δημιουργώντας race condition.

Για την αποφυγή των race conditions χρησιμοποιήσαμε την ατομική εντολή atomicAdd(); Η εντολή αυτή αυξάνει κατά πολύ τον χρόνο εκτέλεσης του kernel σε σχέση με ένα απλό Add.

Δημιουργήσαμε και μία **δεύτερη υλοποίηση** όπου διαιρέσαμε την διαδικασία σε μικρότερα κομμάτια. Με την χρήση streams κατασκευάζονται παράλληλα 2 πίνακες/ιστογράμματα και έπειτα από το άθροισμα προκύπτει ο τελικός. Στο πρώτο stream δίνεται ως input ο μισός πίνακας με διαίρεση δια δύο και στο δεύτερο stream το υπόλοιπο μέρος του πίνακα για την περίπτωση περιττού μεγέθους εικόνας. Η συνάρτηση addH αναλαμβάνει την προσθεση των πινάκων.

Η δεύτερη αυτή υλοποίηση είδαμε ότι σε μεγάλες εικόνες φέρνει καλύτερους χρόνους. Για παράδειγμα στην “planet_surface.pgm” ο χρόνος από 27.05ms έφτασε 21.02ms.

2.Υπολογισμός Συνάρτησης Κατανομής – CreateCDF

Στην αρχική υλοποίηση σε κάθε επανάληψη για N_bins υπολογίζεται το καινούργιο CDF και χρησιμοποιείται για την εύρεση της τιμής στο Look Up Table(LUT).
Για να χωριστεί η διαδικασία σε ένα kernel δημιουργούμε αρχικά έναν πίνακα με όλες τις τιμές του CDF και χρησιμοποιούνται στο επόμενο βήμα από την συνάρτηση CalcLut.

Ο υπολογισμός των τιμών CDF δεν γίνεται διαδοχικά για περιορισμό της αναμονής του προηγούμενου αθροίσματος. Τα αθροίσματα δηλαδή εκτελούνται μερικώς (partial sum) με την χρήση **παράλληλου scan** Brent - Kung για γρηγορότερη υλοποίηση. Ο αλγόριθμος έχει 2 φάσεις και αξιοποιεί shared memory για ταχύτερα loads.

3.Υπολογισμός Πίνακα Αναφοράς – CalcLut

Βασιζόμενοι στον πίνακα του προηγούμενου βήματος δημιουργούμε μόνο μέσω μαθηματικών υπολογισμών πίνακα αναφοράς ίδιου μεγέθους. Κάθε load χρησιμοποιείται για μια σειρά πράξεων πριν την εγγραφή οπότε η συγκεκριμένη συνάρτηση είναι αρκετά αποδοτική.

Για να αποφύγουμε στο επόμενο βήμα τους ελέγχους κατά το πέρασμα των νέων τιμών στον πίνακα, οι ακραίες τιμές <0 και >255 σε αυτό το σημείο αναζητούνται αντικαθίστανται.

4.Δημιουργία Τελικής Εικόνας – EqualizeHist

Χρησιμοποιώντας τον πίνακα LUT που υπολογίσαμε στο προηγούμενο βήμα και grid-strided loop, αντιστοιχίζουμε το κάθε στοιχείο της τελικής εικόνας με την νέα του τιμή που προκύπτει από τον πίνακα LUT με index την εικόνα εισόδου. Όλοι οι έλεγχοι έχουν γίνει σε προηγούμενα βήματα. Η φάση αυτή είναι η πιο χρονοβόρα

5.Παρατηρήσεις σχετικά με την απόδοση

Μεταφορές/χειρισμός μνήμης

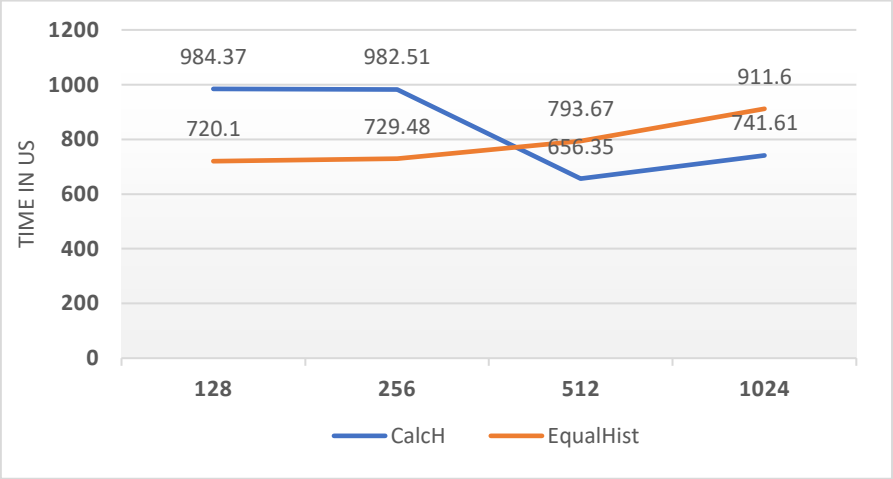
Μεγάλο μέρος του συνολικού χρόνου εκτέλεσης καταλαμβάνουν τα **cudamemcpy** και για την αποφυγή μερικών αποφασίσαμε να μην καλούνται οι συναρτήσεις `contrast_enhancement_g` και `run_cpu_gray_test` που απαιτούν μεταφορά των δεδομένων από το host στο device. Η δημιουργία της τελικής εικόνας γίνεται εξ ολοκλήρου από την main.

Στο kernel EqualizeHist δοκιμάσαμε να περάσουμε τον πίνακα LUT στην **shared memory** έτσι ώστε να μην διαβάζεται από την global memory αλλά από την shared memory.Ο χρόνος εκτέλεσης του συγκεκριμένου kernel αυξήθηκε διότι οι προσβάσεις στην κοινή μνήμη γίνονται με τυχαία σειρά και δεν ήταν coalesced.

Δοκιμάσαμε ακόμα να χρησιμοποιήσουμε **Cudamallocmanaged** αλλά η «χειροκίνητη» χρήση της μνήμης από device σε host και αντίστροφα ήταν πολύ πιο αποδοτική.

Πειραματισμός με μέγεθος grid/block

Για την εκτέλεση των kernel CalcHist και EqualizeHist πειραματιστήκαμε με διαφορετικό αριθμό threads ανά block και μέγεθος grid.



Όπως βλέπουμε στο διάγραμμα οι δύο συναρτήσεις είχαν διαφορετική συμπεριφορά στον χρόνο εκτέλεσης οπότε καταλήξαμε να ορίζουμε τα grid και block dimensions ξεχωριστά για την κάθε μια. Την βέλτιστη απόδοση για το kernel CalcHist την είχαμε με 512 threads ανά block και για το kernel EqualHist με 128.

```
int THREADS_HIST = 512;
int gridDim_HIST = (N + 1) / THREADS_HIST;

int THREADS_EQU = 128;
int gridDim_EQU = (N + 1) / THREADS_EQU;
```

Σημείωση: επισυνάπτονται 2 υλοποιήσεις