Κοσμάτου Ανδρομάχη 2822

Αναστασία Ψαρού 2860

Εργασία 5

Histogram Equalization

Στην εργασία αυτή υλοποιήσαμε την εξίσωση ιστογράμματος πάνω στην GPU με τη βοήθεια της CUDA. Η εξίσωση ιστογράμματος βελτιώνει την “αντίθεση” στις ασπρόμαυρες εικόνες. Η διαδικασία αποτελείται από τα εξης βήματα:

1) Δημιουργούμε το ιστόγραμμα της αρχικής εικόνας

Επειδή οι εικόνες μας είναι ασπρόμαυρες και κάθε εικονοστοιχείο είναι unsigned char άρα 8 bit, κάθε εικόνα περιγράφεται από ένα ιστόγραμμα μεγέθους 256\*sizeof(int).

Στον kernel d\_histogram χρησιμοποιούμε ατομικές εντολές για να υπολογίσουμε τις τιμές του ιστογράμματος της αρχικής εικόνας. Αυτό είναι απαραίτητο καθώς πολλά νήματα γράφουν και διαβάζουν από αυτόν το ιστόγραμμα παράλληλα και πολλές φορές από την ίδια θέση.

Για να μειώσουμε τον χρόνο της καθυστέρησης από τις ατομικές εντολές, βάζουμε τον πίνακα του ιστογράμματος στην shared memory,

ώστε κάθε block να είναι ανεξάρτητο από τα άλλα. Μόλις τελειώσουν όλα τα threads τον υπολογισμό που τους έχει ανατεθεί, τα νήματα με

threadIdx.x από 0 εώς 255 προσθέτουν τα αποτελέσματα από τους επιμέρους πίνακες του κάθε block, και τα αποθηκεύουν στον αντίστοιχο πίνακα της global memory.

2) Υπολογίζουμε την σωρευτική πυκνότητα πιθανότητας του κάθε στοιχείου του ιστογράματος. Το ιστόγραμμα έχει 256 θέσεις άρα και ο πίνακας που θα χρειαστούμε για να αποθηκεύσουμε τις τιμές της σωρευτικής πυκνότητας πιθανότητας (cdf) θα έχει 256 θέσεις.

Αρχικά περνάμε το ιστόγραμμα από την global στην shared memory. Για να υπολογίσουμε την cdf, θα χρησιμοποιήσουμε την τεχνική του prefix sum. Στην πρώτη φάση του αλγόριθμου, διαπερνάμε τον πίνακα από την αρχή εώς το τέλος, υπολογίζοντας κάθε φορά το άθροισμα δύο επιμέρους στοιχείων του πίνακα που απέχουν stride. Για παράδειγμα, στην 1η επανάληψη της λούπας υπολογίζονται τα αθροίσματα arr[0]+arr[1], arr[2]+arr[3] κλπ, στην 2η επανάληψη της λούπας υπολογίζονται τα αθρίσματα arr[1]+arr[3], arr[5]+arr[7] και ούτω καθεξής. Όταν θα έχουν τελειώσει όλα τα νήματα τον υπολογισμό, τότε το τελευταίο στοιχείο του πίνακα θα έχει το άθροισμα όλων των στοιχείων του πίνακα.

Στην δεύτερη φάση του αλγορίθμου,υπολογίζουμε τα επιμέρους αθροίσματα που δεν υπολογίστηκαν στο πρώτο βήμα ολοκληρώνοντας τον αλγόριθμο.

3)Στο τρίτο βήμα της εξίσωσης ιστογράμματος, κατασκευάζουμε έναν πίνακα αναφοράς για να αντιστοιχίσουμε κάθε απόχρωση του γκρι της αρχικής εικόνας στην αντίστοιχη απόχρωση της τελικής εικόνας.

4) Εν τέλει υπολογίζουμε την τελική εικόνα με τη βοήθεια του πίνακα αναφοράς.

Aξίζει να αναφέρουμε ότι αντι να δημιουργήσουμε ένα νήμα για κάθε εικονοστοιχείο της εικόνας, στα kernels που ασχολούμαστε με την αρχική ή την τελική εικόνα (d\_histogram, d\_create\_out) κάθε νήμα μας μπορεί να κάνει παραπάνω από μια δουλειά. Αυτο επιτυγχάνεται με μια λούπα που ξεκινάει από i=threadIdx.x+blockDim.x\*blockIdx.x και σε κάθε επανάληψη το iπροχωράει κατά ένα grid μέχρι να ξεπεράσει το μέγεθος της εικόνας. Για παράδειγμα, αν υπάρχουν 512 νήματα στο grid, το νήμα 0 θα υπολογίσει τα στοιχεία 0, 512, 1024…Με αυτή την τεχνική πετυχαίνουμε τη δυνατότητα κλιμάκωσης και την επαναχρησιμοποίηση νημάτων. Με τη χρήση grid-stride loop μπορούμε να υποστηρίξουμε οποιοδήποτε μέγεθος θέλουμε ακόμη και αν υπερβαίνει το μέγιστο grid size της CUDA device supports. Επίσης, μπορεί να περιοριστεί ο αριθμός των μπλοκ που χρησιμοποιούνται και έτσι να βελτιωθεί η απόδοση. Αυτό συμβαίνει, καθώς νήματα θα επαναχρησιμοποιούνται για πολλούς υπολογισμούς. Επίσης, ένα άλλο πλεονέκτημα είναι το πιο εύκολο debugging που παρέχει ο σειριακός κώδικας. Σε kernels που γράφουμε και διαβάζουμε από πίνακες με διαφορετικά μεγέθη, μπορούμε να κάνουμε launch τον kernel με τον μικρότερο αριθμό νημάτων που θα χρειαστούμε και για τον πιο μεγάλο πίνακα να χρησιμοποιήσουμε grid-stride loop. Έτσι θα αποτρέψουμε πολλά νήματα από το να είναι idle.

Για να δεσμέσουμε μνήμη για την αρχική και την τελική εικόνα καθώς και για τους πίνακες hist, cdf και lut, χρησιμοποιούμε cudaMallocManaged. H cudaMallocManaged σε αντίθεση με την cudaMalloc είναι άμεση προσβάσιμη και από τον host CPU, δηλαδή είναι σαν να έχουμε ένα μοναδικό χώρο μνήμης και όχι δύο(GPU, CPU). Επίσης, με την κλήση της πρώτης δεν χρειάζεται να γίνει κλήση της cudaMemcpy.

Το αποτέλεσμα του nvprof: