Παράλληλα Συστήματα
Εργασία Σεπτεμβρίου 2017-2018

Λέκκας Νικόλαος (1115201600089)

Στην εργασία αυτή στόχος μας είναι η εφαρμογή ενός φίλτρου συνέλιξη σε μια εικόνα τύπου Greyscale ή RGB. Έχουμε ένα φίλτρο που αναπαρίσταται σε πίνακα με πλάτος s. Οι συνολικές διαστάσεις δηλαδή του πίνακα αυτού θα είναι ((2*s)+1). Επίσης έχουμε την εικόνα η οποία αναλόγως αν είναι Greyscale είναι ένας buffer από μονούς χαρακτήρες ενώ αν είναι RGB είναι buffer συνόλου 3ων χαρακτήρων που το καθένα συμβολίζει το red, green και blue.
Η διαδικασία που ακολουθούμε για την επίτευξη της συνέλιξης είναι να "βάζουμε" το φίλτρο πάνω σε κάθε pixel της εικόνας και να εκτελέσουμε την συνάρτηση φίλτρου Io (i, j) $= \sum_{p=-s}^{s} \sum_{q=-s}^{s} Ii\,(i-p, j-q) * h(p,q)$όπου Io είναι η εικόνα εξόδου, Ii η εικόνα εισόδου και h το φίλτρο μας. Το φίλτρο ύστερα από το διάβασμα των δεδομένων του θα βρει το άθροισμα των ψηφίων του και θα διαιρέσει κάθε ψηφίο με αυτό το άθροισμα. Η εικόνα που θα βγει λοιπόν από την εφαρμογή του φίλτρου στην πρώτη εικόνα εισαγωγή θα είναι το αποτέλεσμα μιας διαδικασίας που θα μετατρέψει την εικόνα αλλάζοντας της κάποιο χαρακτηριστικό. Πχ blur, sharpen etc.
Το φίλτρο αυτό μπορεί να εφαρμοστεί πολλαπλές φορές στην εικόνα και να την αλλοιώσει ή να την βελτιώσει αρκετά. Αν διπλασιαστεί, θα διπλασιαστεί το height και το width, αντίστοιχα αν διαιρεθεί κατά 2, θα διαιρεθούν τα μεγέθη. Στόχος αυτής της άσκησης είναι αυτή η διαδικασία να γίνεται παράλληλα στον υπολογιστή. Να μπορέσουμε τη διαδικασία διαβάσματος και εγγραφής της εικόνας(parallel I/o) καθώς και η διαδικασία της συνέλιξης να γίνεται σε πολλαπλούς επεξεργαστές την ίδια στιγμή. Υπάρχουν 2 μοντέλα που ακολουθούμε για την επίτευξη αυτή της παραλληλίας. Το ένα είναι το MPI το οποίο είναι μια αρχιτεκτονική που βασίζεται στη διαχείριση κατανεμημένης μνήμης (distributed memory) ώστε να έχει κάθε επεξεργαστής τη δικιά του μνήμη και να έχει άμεση πρόσβαση σε αυτή. Με το MPI τίθεται το πρόβλημα της επικοινωνίας των επεξεργαστών καθώς κάποιος επεξεργαστής μπορεί να χρειάζεται κάποια δεδομένα από τη μνήμη ενός άλλου επεξεργαστή. Ευτυχώς το MPI έχει τις εντολές MPI_SEND, MPI_RECV και τις παρεμφερείς τους που είναι υπεύθυνες στην ανταλλαγή μηνυμάτων μεταξύ επεξεργαστών ώστε να έχουν πρόσβαση στην μνήμη των άλλων. Επίσης το MPI προσφέρει μεγάλη κλιμάκωση και επεκτασιμότητα καθώς είναι πολύ πιο εύκολο να προσθέσεις νέους επεξεργαστές στο κύκλωμα οικονομικά και έχουμε μεγάλη αποτελεσματικότητα δηλαδή με σταθερό μέγεθος και αύξηση των επεξεργαστών έχουμε επιτάχυνση. Από την άλλη υπάρχει το OpenMP όπου η αρχιτεκτονική του βασίζεται σε κοινή μνήμη (shared memory). Στην αρχιτεκτονική αυτή όλοι οι επεξεργαστές συνδέονται σε μία κοινή μνήμη που βρίσκονται όλα τα δεδομένα. Έτσι σε αντίθεση με την αρχιτεκτονική της κατανεμημένης μνήμης δε χρειάζεται η ανταλλαγή μηνυμάτων για την πρόσβαση των επεξεργαστών σε κάποια δεδομένα. Η σύνδεση μνήμης και επεξεργαστών γίνεται με ένα δίκτυο διασύνδεσης. Τα κύρια αρνητικά της κοινής μνήμης είναι 2. Το ένα είναι το κόστος της επεκτασιμότητας σε σχέση με την κατανεμημένη μνήμη και το άλλο το race condition που δημιουργείται όταν πολλοί επεξεργαστές προσπαθούν να λάβουν ταυτόχρονα δεδομένα από τη μνήμη. Στην αρχιτεκτονική OpenMP η λειτουργία της παραλληλίας γίνεται πιο αυτόνομα από το σύστημα και ο προγραμματιστής αποφασίζει τις επαναλήψεις και τα blocks που θέλει να χωριστούν παράλληλα σε threads και ύστερα ο υπολογιστής

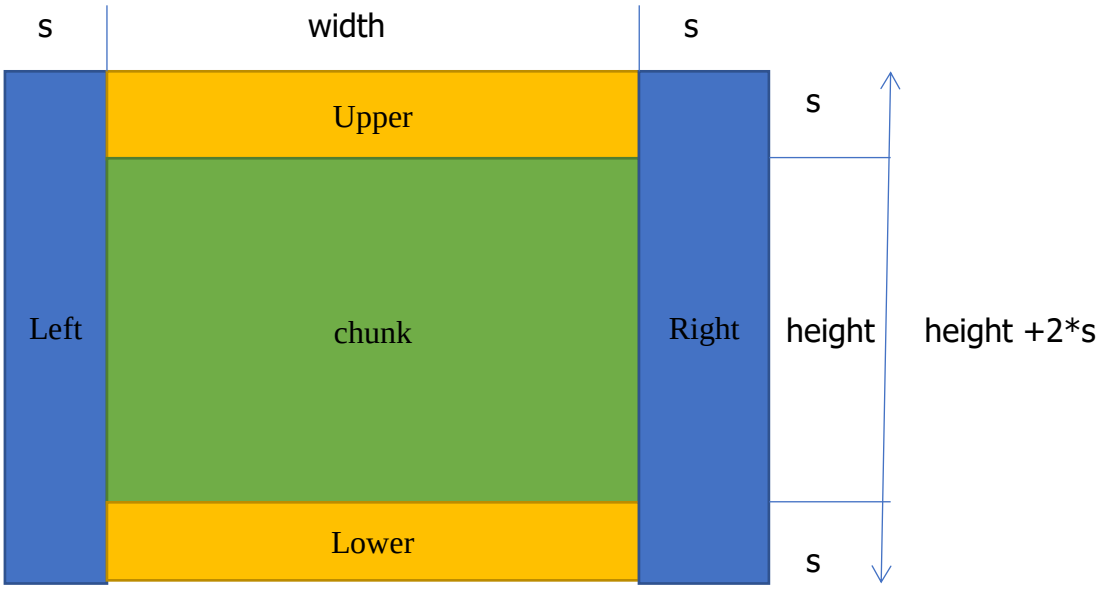
ασχολείται για το πως θα γίνει αυτός ο διαχωρισμός. Στο MPI ο προγραμματιστής είναι υπεύθυνος στην οργάνωση και τον χωρισμό των δεδομένων του στα threads με κατάλληλο τρόπο ώστε να επιτυγχάνει γρήγορη ταχύτητα και ισορροπία φόρτου δεδομένων μεταξύ των threads.

Για την οργάνωση του κώδικα χρησιμοποιήθηκαν ειδικά structs τα οποία περιλαμβάνουν δεδομένα για τα μεγέθη των πινάκων των εικόνων καθώς και για τα χρώματα των pixels(RGB, Greyscale). Επειδή υπάρχουν 2 επιλογές(Greyscale, RGB) χρησιμοποιήθηκε το MYDATA που αλλάζει μεταξύ Greyscale(char) και RGB (3 char(red,green,blue)) με τη χρήση #IFDEF #ENDIF και #DEFINE με σκοπό την αποφυγή επανάληψης του κώδικα.
Ο κώδικας επίσης έχει χωριστεί σε συναρτήσεις ώστε να μπορέσουμε όσο το δυνατό γίνεται να διαχωρίσουμε τις λειτουργίες που εκτελούνται και να κάνουμε τον κώδικα όσο το δυνατό γίνεται ευανάγνωστο και κατανοητό.
Πιο συγκεκριμένα υπάρχει η συνάρτηση convolute που είναι υπεύθυνη για την διαδικασία της συνέλιξης του φίλτρου στην εικόνα(ίσως πολλαπλές φορές). Χρησιμοποιείται η μεταβλητή path ώστε να γίνει για το grey pixel και αν είναι RGB το red, green, blue(βλέπε color[3]). Για να γίνει παράλληλα ο κώδικας πρέπει να χωριστεί η εικόνα σε κομμάτια. Τα κομμάτια τα ορίζω ως chunks όπου κάθε chunk έχει το δικό του ύψος και πλάτος, το offsetx και offsety που δίνει την θέση του στην όλη εικόνα, το img_width και img_height που ειναι ο διαστάσεις όλης της εικόνας, την εικόνα εισαγωγής(in) και εξαγωγής(out) καθώς και τα x,y που αριθμεί τα threads σε δισδιάστατο πίνακα. Για να βρούμε τις διαστάσεις του chunk χρησιμοποιούμε πρώτους παράγοντες με τη συνάρτηση split_chunks όπου διαιρώντας τα μεγέθη και το size των threads με τους πρώτους παράγοντες επιστρέφουμε τα μεγέθη του κάθε chunk.

Αφού χωρίσουμε την εικόνα σε κομμάτια θα αρχίσουμε να κάνουμε συνέλιξη στα pixels του κάθε κομματιού και θα παρατηρήσουμε ότι στα άκρα χρειάζονται πληροφορίες από στήλες και γραμμές της εικόνας που δεν έχει το παρών chunk. Για να το αντιμετωπίσουμε αυτό φτιάχνουμε πίνακες UpperTable , LowerTable, LeftTable και RightTable όπου θα έχουν τα δεδομένα των πινάκων που χρειάζονται για τη συνέλιξη. Τα lower και upper θα εχουν μέγεθος s γραμμές και width στήλες ενώ τα left και right θα έχουν height+2*s γραμμές(όπου το 2*s χρησιμεύει για τις s πληροφορίες που θα βρίσκονται στις γωνίες των εικόνων) και s στήλες ώστε το κάθε chunk να πάρει όλα τα περιφερειακά δεδομένα που χρειάζεται για να κάνει τη σωστή συνέλιξη. Για το διάβασμα των τιμών των 4ων αυτών πινάκων χρησιμοποιούμε την MPI_File_read_at που βασιζόμαστε στο offset που έχουμε αποθηκεύσει στο chunk και για να μεταφέρουμε αυτές τις τιμές χρησιμοποιείται η MPI_iSend και για παραλαβή η MPI_iRecv προσέχοντας ιδιαίτερα στις θέσεις του πίνακα που θα χρειαστούμε να λάβουμε και να στείλουμε με σκοπό τη σωστή προσπέλαση και μετέπειτα τη σωστή συνέλιξη της εικόνας.

Για την μεταφορά του s και του φίλτρου h χρησιμοποιείται η MPI_Bcast η οποία θα μεταφέρει σε όλους τη πληροφορία.

Στην περίπτωση που στη συνέλιξη βρεθούμε εκτός πίνακα τότε βάζουμε ως τιμή την τιμή του κεντρικού pixel στο οποίο εφαρμόζουμε τη συνέλιξη. Για τη διαδικασία του πολλαπλασιασμού ή διαίρεσης της εικόνας χρησιμοποιούμε το resize το οποίο το πολλαπλασιάζουμε στο height και στο width της εικόνας για να βρούμε τα νέα μεγέθη.



Για την εκτέλεση του παράλληλου προγράμματος χρειαζεται:
RGB: mpicc <program_name>.c -o exec -DUSE_RGB -lm
Greyscale: mpicc <program)name>.c -o exec
#(use -DUSE_RGB for RGB) (Nothing for Greyscale)
Run:mpirun -np <numofthreads> ./exec <width> <height> <numofconvolutions> <resize (float)>
πχ. mpirun -np 4 ./exec 1920 2520 2 0.5

Για την εκτέλεση του σειριακού προγράμματος χρειαζεται:
RGB: gcc <program_name>.c -o exec -DUSE_RGB
Greyscale: gcc <program)name>.c -o exec
#(use -DUSE_RGB for RGB) (Nothing for Greyscale)
Run:./exec <width> <height> <numofconvolutions> <resize (float)>
Πχ. ./exec 1920 2520 2 0.5

Το compile μπορεί να επιτευχθεί και με το makefile που έχω συμπεριλάβει

Μετρήσεις
1 διεργασία – 1 convolution – 1 resize Greyscale
CPU_time = 1.369938 (με επικοινωνία)
1 διεργασία – 1 convolution – 1 resize RGB
CPU_time = 1.358874 (με επικοινωνία)
1 διεργασία – 2 convolution – 1 resize Greyscale
CPU_time = 2.729339 (με επικοινωνία)
1 διεργασία – 2 convolution – 1 resize RGB
CPU_time = 8.126979 (με επικοινωνία)
4 διεργασίες – 2 convolution – 1 resize Greyscale
CPU_time = 1.058213 (με επικοινωνία)
4 διεργασίες – 2 convolution – 1 resize RGB
CPU_time = 3.231576 (με επικοινωνία)

9 διεργασίες – 2 convolution – 1 resize Greyscale
CPU_time = 1.624738 (με επικοινωνία)
9 διεργασίες – 2 convolution – 1 resize RGB
CPU_time = 5.087909 (με επικοινωνία)
16 διεργασίες – 2 convolution – 1 resize Greyscale
CPU_time = 1.922746 (με επικοινωνία)
16 διεργασίες – 2 convolution – 1 resize RGB
CPU_time = 4.784326 (με επικοινωνία)
25 διεργασίες – 2 convolution – 0.5 resize Greyscale
CPU_time = 0.478740 (με επικοινωνία)
25 διεργασίες – 2 convolution – 0.5 resize RGB
CPU_time = 1.220356 (με επικοινωνία)
36 διεργασίες – 2 convolution – 2 resize Greyscale
CPU_time = 6.336419 (με επικοινωνία)
36 διεργασίες – 2 convolution – 2 resize RGB
CPU_time = 17.461623 (με επικοινωνία)

Μετά την εκτέλεση και των 2 τρόπων παραλληλοποίησης παρατηρούμε ότι το MPI προσφέρει πολύ καλύτερες επιδόσεις αλλά είναι και πιο περίπλοκο στην διαχείριση του ενώ το OpenMP είναι μεν πιο απλό αλλά προσφέρει μικρότερη επιτάχυνση.
Όσο αυξάνεται το μέγεθος του προβλήματος τόσο πιο αργό είναι το πρόγραμμα όπως και αν έχουμε μεγάλο πλήθος συνελίξεων.
Έτσι παρατηρούμε ότι όσο αυξάνεται το μέγεθος του προβλήματος αυξάνεται ο χρόνος εκτέλεσης. Αυτό μπορούμε να το δούμε άμεσα βλέποντας ότι το RGB που εχει 3 bytes κάνει παραπάνω χρόνο από το Greyscale που είναι 1 byte. Όταν πολλαπλασιάζουμε το μέγεθος τότε αυξάνεται ο χρόνος και όταν διαιρείται το μέγεθος ο χρόνος μειώνεται. Όταν χρησιμοποιούμε παραπάνω διεργασίες το πρόβλημα επίσης γίνεται πιο γρήγορο και χρειάζεται λιγότερος χρόνος. Άρα αφού βελτιώνοντας τη ταχύτητα με περισσότερες διεργασίες και με μεγαλύτερο(μικρή κλιμάκωση) ή με σταθερό μέγεθος(μεγάλη κλιμάκωση) έχουμε σταθερή αποτελεσματικότητα και άρα κλιμάκωση.

Η υβριδική λύση είναι να παραλληλοποιήσουμε τις επαναλήψεις for της convolute με χρήση του OpenMP έτσι ώστε να έχουμε πολλαπλά threads σε κάθε process του MPI. Για την παραλληλοποίηση του προγράμματος με openMP χρειάστηκε να γράψουμε #pragma omp parallel for shared(variables) private(variables) το οποίο θα χωρίσει τις επαναλήψεις της for στα threads του συστήματος επιτυγχάνοντας παραλληλοποίηση.