

# Παράλληλα Συστήματα

Δημήτριος Γάγγας

1115201400024

Ιωάννης Παπαδόπουλος

1115201400144

14 Οκτωβρίου 2018

## Περιεχόμενα

<b>1</b>	<b>Εισαγώγη</b>	<b>2</b>
1.1	Δομή αρχείων . . . . .	2
1.2	Οδηγίες Μεταγλώττισης . . . . .	2
1.3	Οδηγίες Εκτέλεσης . . . . .	3
1.4	Παραδοχές . . . . .	3
<b>2</b>	<b>Σχεδιασμός Προγράμματος</b>	<b>3</b>
2.1	Παράλληλο I/O . . . . .	3
2.2	Διαχωρισμός σε blocks . . . . .	4
2.3	Επικοινωνία διεργασιών . . . . .	4
2.4	Τλοποίηση υβριδικού MPI+OpenMp . . . . .	5
2.5	Έλεγχος τερματισμού . . . . .	5
<b>3</b>	<b>Μετρήσεις και Συμπεράσματα</b>	<b>6</b>
<b>4</b>	<b>Εικόνες</b>	<b>6</b>
4.1	Πριν και Μετά με 20 επαναλήψεις: . . . . .	6
4.2	Πριν και Μετά με 40 επαναλήψεις: . . . . .	7
<b>5</b>	<b>Βιβλιογραφία</b>	<b>8</b>

# 1 Εισαγώγη

Στην παρούσα εργασία ασχοληθήκαμε με το πρόβλημα της παραλληλοποίησης της συνέλιξης εικόνας. Συγκεκριμένα, ασχολήθηκαμε πρακτικά με την συνέλιξη μιας διδιάστατης εικόνας με ένα φίλτρο συνέλιξης. Μέσω της προσπάθειας ανάπτυξης του με παράλληλο πρόγραμμα, πειραματιστήκαμε με διάφορες μεθόδους παραλληλίας και ήρθαμε στην πράξη αντιμέτωποι με τα συνηθέστερα θέματα που προκύπτουν σε αυτό το είδος προγραμματισμού. Σκοπός της άσκησης είναι η οσο το δυνατόν καλύτερη απόδοση του προγραμμάτου που απόφερει τον ελάχιστο δυνατό χρόνο εκτέλεσης.

Δυστυχώς, δεν υπήρχε δυνατότητα να υλοποιηθεί ο παράλληλος προγραμματισμός σε CUDA λόγω έλλειψης κατοχής καρτών γραφικών Nvidia και από τα 2 μέλη της ομάδας.

Σημείωση : Ελέχθηκε με valgrind και δεν παρουσιάζει κάποιου είδους memory leak.

## 1.1 Δομή αρχείων

Ο κώδικας έχει χωριστεί ως εξής:

- Η υλοποίηση του mpi (mpi.c) βρίσκεται στο κατάλογο mpi, ενώ η υλοποίηση του υβριδικού μοντέλου (open\_mp.c) βρίσκεται στο κατάλογο open\_mp.
- Εντός των δύο αυτών καταλόγων υπάρχει ο φάκελος output\_images, στον οποίο αποθηκεύονται οι φιλτραρισμένες/επεξεργασμένες εικόνες.
- Στον εξωτερικό κατάλογο περιέχεται το makefile.

## 1.2 Οδηγίες Μεταγλώττισης

Δυνατοί τρόποι μεταγλώττισης του προγράμματος:

- **make** δημιουργούνται τα εκτελέσιμα mpi\_exe, open\_mp\_exe μέσα στους αντίστοιχους φακέλους.
- **make mpi(/open\_mp)** δημιουργείται μόνο το εκάστοτε εκτελέσιμο.
- **make clean\_exes** διαγράφονται και τα δύο εκτελέσιμα.
- **make clean\_mpi(/\_open\_mp)** διαγράφεται το εκάστοτε εκτελέσιμο.
- **make clean\_outs** διαγράφονται οι φιλτραρισμένες φωτοφραφίες που πρέχυψαν και με τις δύο μεθόδους.

- **make clean\_mpi\_outs(/clean\_open\_mp\_outs)** διαγράφονται οι φιλτραρισμένες φωτοφραφίες που προέκυψαν με την εκάστοτε μέθοδο.
- **make clean\_all** εκτελούνται οι προαναφερθήσες εντολές make clean\_exes και make clean\_outs.

### 1.3 Οδηγίες Εκτέλεσης

Για την εκτέλεση του εκάστοτε εκτελέσιμου αρχεί η εντολή, μέσα από τον αντίστοιχο κατάλογο.

Τυποδείξεις εκτέλεσεις υπάρχουν στα αρχεία **runCommand.txt** του εκάστοτε φακέλου.

### 1.4 Παραδοχές

- Ο αριθμός των διεργασιών θα πρέπει να είναι τέλειο τετράγωνο. Θα μπορούσαμε να είχουμε αποφύγει τον περιορισμό αυτό αλλά θεωρήσαμε πως δεν εμπλέκεται τόσο στον σκοπό της εργασίας.
- Αν για κάποιο μπλοκ δεν υπάρχει γείτονικο μπλοκ από κάποια πλεύρα δεν στέλνει ούτε λαμβάνει σε κάποιο άλλο μπλοκ την πλευρά αυτή.
- Ο αριθμός των επαναλήψεων/φορών κατα την οποία θα επενεργήσει το φίλτρο συνέλιξης στην εικόνα ειναι δηλωμένος ως #define GENERATION
- Επίσης όπως γνωρίζουμε από Foster καλό είναι να αποφεύγονται οι διπλότυπες πληροφορίες που στη δική μας περίπτωση είναι τα περιμετρικά halo points. Παρόλ' αυτά από τη στιγμή που χρησιμοποιούνται για conglomeration καθιστάται να είναι ο πλέον αποδοτικός τρόπος.

## 2 Σχεδιασμός Προγράμματος

### 2.1 Παράλληλο I/O

Εχει υλοποιηθεί και η λειτουργικότητα του παράλληλου I/O. Στην λειτουργικότητα αυτή, ολές οι διεργασίες βλέπουν σε ένα αρχείο. Στη συνέχεια με βάση το pid τους διαβάζουν στο αντίστοιχο offset που τους αναλογεί.

Αντίστοιχα, στο τέλος οταν δημιουργείται το αρχείο εξόδου γράφουν στο κατάλληλο offset της τιμές του μονοδιάστατου πίνακα τους στο οποίο έχει εφαρμοστεί το φίλτρο συνέλιξης.

Αξίζει επίσης να σημειωθεί οτι με αυτόν τον τρόπο επιτυγχάνεται μικρή κατανάλωση μνήμης, καθώς κάθε μια διεργασία διαβάζει μόνο το αντίστοιχο κομμάτι μνήμης που της αναλογεί.

## 2.2 Διαχωρισμος σε blocks

Ο διαχωρισμός των μπλοκ επιτεύχθηκε με την συνάρτηση **div2blocks()**. Ουσιαστικά λαμβάνει ως άριθμο το πλάτος, το ύψος της εικόνας καθώς επίσης και τον αριθμό των διεργασιών και επιστρέφει τον βέλτιστο τρόπο με τον οποίο θα ισοχατανεμηθεί η είκονα/πίνακας στις διεργασίες. Αν δεν μπορεί να διασπαστεί η εικόνα ομοιόμορφα σε blocks με ίσο αριθμό γραμμών και στηλών τότε επιστρέφει σφάλμα.

Στην απόφαση αυτή οδηγηθήκαμε από το γεγονός ότι μας ενδιαφέρει ως επί το πλείστον η ισοχατανομή του φόρτου εργασίας μεταξύ των διεργασιών έτσι ώστε να αποφευχθεί ενδεχόμενο bottleneck μεταξύ των διεργασιών ανα γενιά.

## 2.3 Επικοινωνία διεργασιών

Για την επικοινωνία μεταξύ των διεργασιών έχουν υλοποιηθεί 2 datatypes ένα για τις γραμμές και ενα για τις στήλες αντίστοιχα για τους 2 τύπους εικόνας. Το κανέ μπλοκ στέλνει στους γείτονες του (βορρά, νότο, ανατολή, δύση) τις γραμμές και τις στήλες του αν και εφόσον υπάρχει γειτονικό μπλοκ από την εκάστοτε πλευρά.

Η στρατηγική της ανταλλαγή των γραμμών και στηλών είναι βασισμένη σύμφωνα με το μοντέλο ghost cell pattern. Ουσιαστικά, με αυτό το μοντέλο καταφέρνουμε να γλυτώσουμε να αποστείλουμε καθώς επίσης και να λάβουμε τα 4 corner halo points για κάθε μπλοκ. Πιο αναλυτικά, η ανταλλαγή των γραμμών και στηλών γίνεται σε **2 φάσεις**.

Σε πρώτη φάση στέλνονται ολές οι στήλες στους γείτονες και περιμένουμε να ληφθούν. Γι' αυτό το λόγο χρησιμοποιήσαμε τις blocking συναρτησεις **MPI\_Send()** και **MPI\_Recv()**.

Σε δεύτερη φάση στέλνονται όλες οι γραμμές στους γείτονες με τις non-blocking συναρτήσεις **MPI\_ISend()** και **MPI\_IRecv()**.

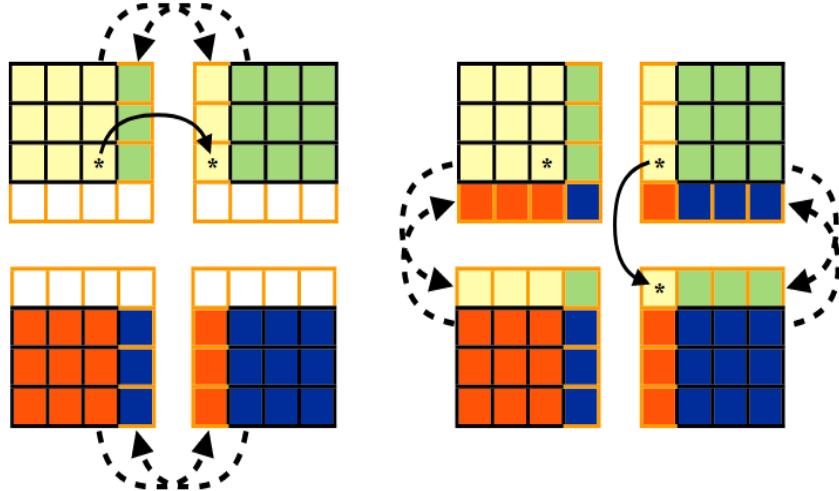


Figure 9: Border Exchange in two waves to copy corner cells across diagonals. One cell is marked to show its journey to the diagonal neighbor.

Όσο εκτελείται αυτή η διαδικασία(2η φάση), το μπλοκ προχωράει στον υπόλογισμό των εσωτερικών του στοιχείων. Επίσης ανάλογα με το ποια γραμμή του έχει έρθει πρώτα προχωράει στον υπολογισμό της. Αυτό επιτυγχάνεται μέσω των συναρτήσεων **MPI\_Test()**.

## 2.4 Υλοποίηση υβριδικού MPI+OpenMp

Η υλοποίηση είναι ένα hybrid παράλληλο σύστημα που περιέχει τόσο MPI για τον διαμοιρασμό του προβλήματος σε πολλαπλά processes και την μεταξύ τους επικοινωνία, όσο και openMP για την δημιουργία και την εκτέλεση παράλληλων threads μέσα στο κάθε process.

Το openMP χρησιμοποιείται για τον υπολογισμό των εσωτερικών κελιών των blocks από κάθε διεργασία και σε κάθε επανάληψη/στιγμιότυπο καθώς επίσης και για τη σύγχριση ομοιότητας πινάκων πριν και μετά.

## 2.5 Έλεγχος τερματισμού

Για να ελέγξουμε αν το πρόγραμμα πρέπει να τερματιστεί σε κάθε επανάληψη ελέγχουμε αν το φίλτρο είχε ως απότελεσμα την αλλαγή της εικόνας σε μια νέα διαφορετική ή όχι.

Αυτό το επιτυγχάνουμε με τη συνάρτηση **ImageChanged()**. Παρολόγιτά παρατηρήσαμε ότι οι χρόνοι τρέχοντας το πρόγραμμα με τη συνάρτηση αυτή είναι ελαφρά χειρότερη.

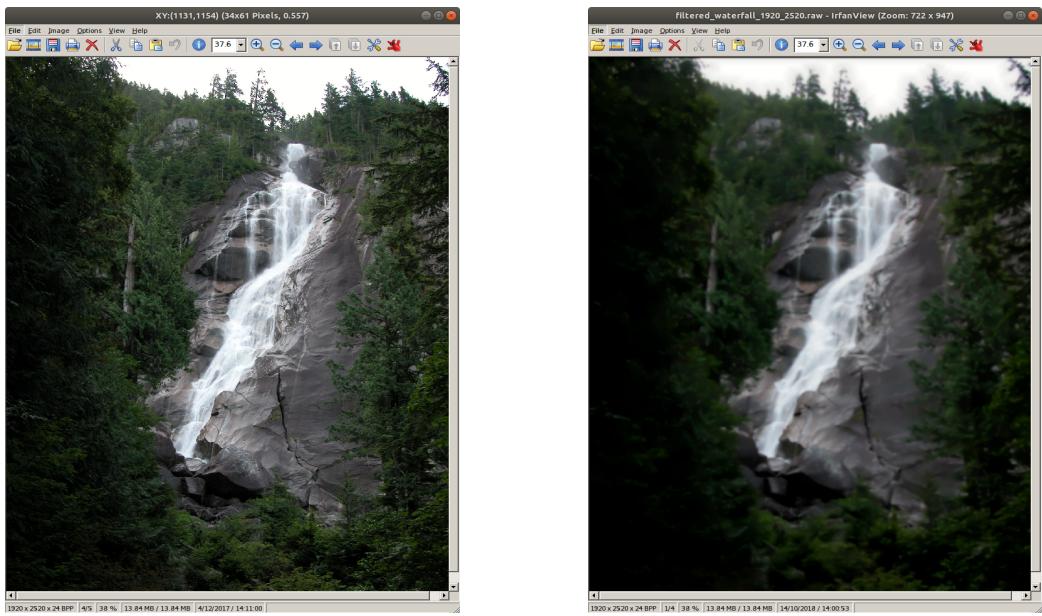
### 3 Μετρήσεις και Συμπεράσματα

Οι μετρήσεις και τα συμπεράσματα που απορρέουν βρίσκονται στο αρχείο metrics.pdf .

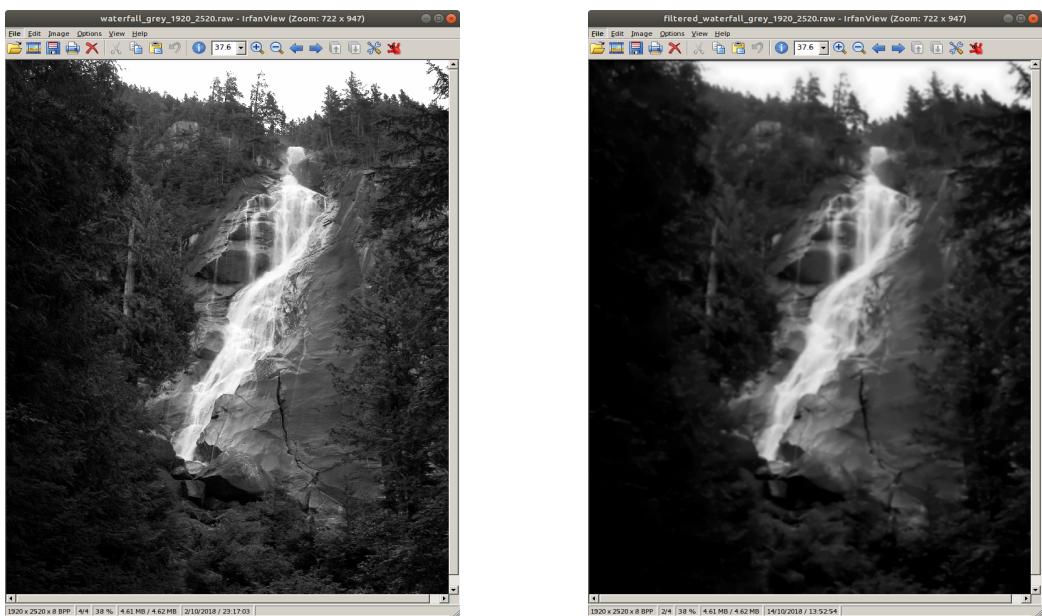
### 4 Εικόνες

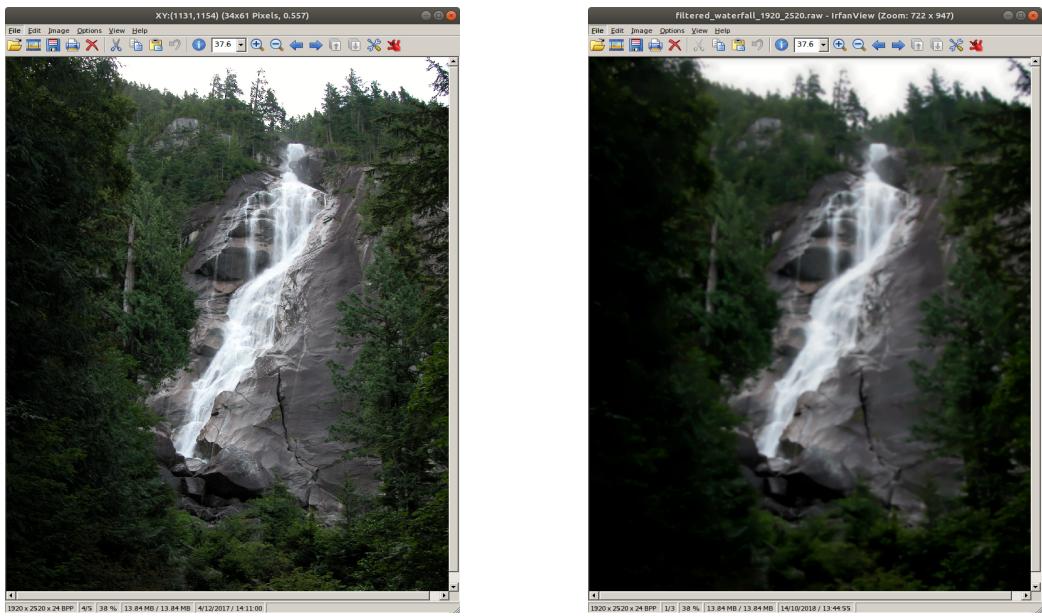
#### 4.1 Πριν και Μετά με 20 επαναλήψεις:





## 4.2 Πριν και Μετά με 40 επαναλήψεις:





## 5 Βιβλιογραφία

ghost cell pattern paper