****

**ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ**

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

Παράλληλα Συστήματα

«Παράλληλη υλοποίηση φίλτρου εικόνας δισδιάστατης συνέλιξης»

Καμπυλαυκάς Ιωάννης

**ΕΠΙΒΛΕΠΩΝ: ΚΟΤΡΩΝΗΣ ΙΩΑΝΝΗΣ**

**Αθήνα - 2015**

Περιεχόμενα

[Εισαγωγικά 4](#_Toc413011234)

[Συμβάσεις για τα αρχεία εισόδου 4](#_Toc413011235)

[Εκδοχές του SPMD προγράμματος 4](#_Toc413011236)

[Συνοπτική περιγραφή της λειτουργίας του SPMD προγράμματος 5](#_Toc413011237)

[Αρχικά προβλήματα στο διαμοιρασμό των δεδομένων 5](#_Toc413011238)

[Ο πηγαίος κώδικας 7](#_Toc413011239)

[Περιγραφή της δομής του SPMD προγράμματος (master - worker) 7](#_Toc413011240)

[Ανάγνωση παραμέτρων 8](#_Toc413011241)

[Η δέσμευση, χρήση και αποδέσμευση μνήμης 8](#_Toc413011242)

[Αποστολή και λήψη των τμημάτων εικόνας από τον master 9](#_Toc413011243)

[Δέσμευση μνήμης από τους workers και λήψη του τμήματος εικόνας 10](#_Toc413011244)

[Δέσμευση των δυο πινάκων κινητής υποδιαστολής για τους υπολογισμούς 11](#_Toc413011245)

[Προσδιορισμός των rank των γειτονικών worker διεργασιών 11](#_Toc413011246)

[Δημιουργία των datatypes για την επικοινωνία μεταξύ των workers 12](#_Toc413011247)

[Ο κύριος βρόχος επεξεργασίας με ασύγχρονη MPI επικοινωνία 12](#_Toc413011248)

[Ο κύριος βρόχος επεξεργασίας με σύγχρονη MPI επικοινωνία 18](#_Toc413011249)

[Υβριδική MPI/OpenMP υλοποίηση, ασύγχρονη MPI επικοινωνία 19](#_Toc413011250)

[Υβριδική MPI/OpenMP υλοποίηση, σύγχρονη MPI επικοινωνία 21](#_Toc413011251)

[Μετρήσεις 22](#_Toc413011252)

[Hardware 22](#_Toc413011253)

[Τρόπος λήψης μετρήσεων 23](#_Toc413011254)

[MPI υλοποίηση 24](#_Toc413011255)

[Περιγραφή μετρήσεων 24](#_Toc413011256)

[Διαγράμματα - πίνακες μετρήσεων 25](#_Toc413011257)

[Συγκεντρωτικοί πίνακες Speedup / Efficiency 30](#_Toc413011258)

[Μικροδιαφορές: σύγκριση persistent / non-persistent ασύγχρονης MPI επικοινωνίας 31](#_Toc413011259)

[Υβριδική MPI/OpenMP υλοποίηση 32](#_Toc413011260)

[Περιγραφή μετρήσεων 32](#_Toc413011261)

[Διαγράμματα - πίνακες μετρήσεων 33](#_Toc413011262)

[Συγκεντρωτικοί πίνακες Speedup / Efficiency 38](#_Toc413011263)

[Μικροδιαφορές: σύγκριση μεταξύ της «σύνθετης» και της «απλής» MPI/OpenMP υλοποίησης 39](#_Toc413011264)

[CUDA 40](#_Toc413011265)

[Πηγαίος κώδικας 40](#_Toc413011266)

[Η δέσμευση και αποδέσμευση μνήμης 40](#_Toc413011267)

[Οι παράμετροι της συσκευής 41](#_Toc413011268)

[Ο kernel για το εσωτερικό μέρος του φίλτρου 42](#_Toc413011269)

[Hardware 43](#_Toc413011270)

[Επεξεργαστής 43](#_Toc413011271)

[Κάρτα γραφικών 44](#_Toc413011272)

[Μετρήσεις 44](#_Toc413011273)

# Εισαγωγικά

## Συμβάσεις για τα αρχεία εισόδου

Τα αρχεία εικόνας που δόθηκαν με την εκφώνηση της εργασίας έχουν διαστάσεις 1920 x 2520, ανάλυση την οποία χαρακτηρίσαμε “1.00x”. Από τις αρχικές εικόνες δημιουργήσαμε εικόνες μεγαλύτερων και μικρότερων διαστάσεων. Στον παρακάτω πίνακα φαίνεται η αντιστοίχιση αναλύσεων και ονομάτων τα οποία χρησιμοποιούνται στο παρόν κείμενο, τους πίνακες και τα διαγράμματα.

Πίνακας 1: Οι αναλύσεις εικόνων που χρησιμοποιήθηκαν στις μετρήσεις

|  |  |  |
| --- | --- | --- |
| ονομασία | ανάλυση | προσανατολισμός |
| 0.25x | 960x1260 | portrait |
| 0.50x | 1920x1260 | landscape |
| 1.00x | 1920x2520 | portrait |
| 2.00x | 3840x2520 | landscape |
| 4.00x | 3840x5040 | portrait |

Κάποια από τα πλέγματα υπολογισμού που χρησιμοποιήθηκαν για την υποδιαίρεση της εικόνας είχαν διαφορετικό αριθμό γραμμών και στηλών. Ανάλογα με τον «προσανατολισμό» της εικόνας χρησιμοποιήθηκε και το κατάλληλο πλέγμα, για παράδειγμα σε υποδιαίρεση 12 διεργασιών χρησιμοποιήθηκε 4x3 πλέγμα για τις portrait και 3x4 για τις landscape εικόνες. Στόχος ήταν, τουλάχιστον θεωρητικά, υποδιαιρέσεις της εικόνας όσο γίνεται πιο κοντά στο τετράγωνο ώστε να έχουμε πιο ισορροπημένη επικοινωνία με μεγιστοποίηση του μεγέθους των μηνυμάτων μεταξύ των διεργασιών.

## Εκδοχές του SPMD προγράμματος

Ακολουθεί μια σύνοψη των εκδοχών του προγράμματος που αναπτύχθηκαν με τα χαρακτηριστικά που υποστηρίζει η κάθε μια.

1. Σειραϊκό (φάκελος spmd/serial)

* serial (single thread)
* serial\_omp (OpenMP)

1. Παράλληλο με ασύγχρονη επικοινωνία (φάκελος spmd/async)

* async\_nonper (MPI, non-persistent επικοινωνία)
* async (MPI, persistent επικοινωνία)
* async\_omp\_simple (MPI/OpenMP, δημιουργία νημάτων σε κάθε επανάληψη του βρόχου)
* async\_omp (MPI/OpenMP, δημιουργία νημάτων μόνο μια φορά, έξω από τον κύριο βρόχο)

1. Παράλληλο με σύγχρονη επικοινωνία (φάκελος spmd/sync)

* sync (MPI)
* sync\_omp\_simple (MPI/OpenMP, δημιουργία νημάτων σε κάθε επανάληψη του βρόχου)
* sync\_omp (MPI/OpenMP, δημιουργία νημάτων μόνο μια φορά, έξω από τον κύριο βρόχο)

Το σειραϊκό πρόγραμμα αναπτύχθηκε ως πρόγραμμα «αναφοράς» την έξοδο του οποίου ακολούθως επιδιώξαμε να διατηρήσουμε χωρίς αλλαγές και στις παράλληλες εκδόσεις. Όλα τα παράλληλα προγράμματα έχουν την ίδια έξοδο, μάλιστα το φίλτρο είναι απόλυτα σταθερό για συγκεκριμένη είσοδο και αριθμό επαναλήψεων, ανεξάρτητα από την υποδιαίρεση που θα γίνει σε συγκεκριμένο πλέγμα υπολογισμού. Αυτό απαίτησε κατάλληλο χειρισμό των διαγώνιων οριακών δεδομένων στα παράλληλα προγράμματα MPI/OpenMP, παίρνοντας δεδομένα που λείπουν από τυχόν διαθέσιμους οριζόντιους ή κάθετους γείτονες. Η διατήρηση της σταθερότητας του φίλτρου καθ’ όλη τη διάρκεια της ανάπτυξης βοήθησε πολύ στον έλεγχο ορθότητας των διαφόρων εκδοχών του προγράμματος και στον έγκαιρο εντοπισμό προβλημάτων με την αποστολή μηνυμάτων ή το διαμοιρασμό της επεξεργασίας σε νήματα.

## Συνοπτική περιγραφή της λειτουργίας του SPMD προγράμματος

Το SPMD πρόγραμμα είναι φτιαγμένο με τη λογική μιας master διεργασίας που αναλαμβάνει να ανοίξει το αρχείο τοπικά, να το μοιράσει στις worker διεργασίες του πλέγματος υπολογισμού, και κατόπιν να συλλέξει και αποθηκεύσει τα αποτελέσματα. Χρησιμοποιούνται δυο communicators. Ο προκαθορισμένος global communicator για την επικοινωνία της master διεργασίας με τους workers, και ένας επιπλέον communicator που περιέχει μόνο τις worker διεργασίες και υποστηρίζει την επικοινωνία μεταξύ τους κατά το στάδιο των υπολογισμών. Ο δεύτερος αυτός communicator υποστηρίζει συντεταγμένες καρτεσιανής τοπολογίας και δημιουργείται βασισμένος σε έναν προσωρινό communicator που μόνη χρησιμότητα έχει τον αποκλεισμό της master διεργασίας. Το πρωτόκολλο μεταξύ του master και των workers έχει ως εξής: Αρχικά ο master περιμένει από κάθε worker να του στείλει τις καρτεσιανές του συντεταγμένες, ακολούθως ο master στέλνει βάσει αυτών των συντεταγμένων το μέρος της εικόνας που αντιστοιχεί στον worker, και τέλος περιμένει από τον worker να στείλει τα αποτελέσματα της επεξεργασίας πάνω στο κομμάτι αυτό.

## Αρχικά προβλήματα στο διαμοιρασμό των δεδομένων

Γρήγορα ανακαλύψαμε κάποιους περιορισμούς στην αποστολή/λήψη δεδομένων με τη χρήση master διεργασίας. Στην αρχική υλοποίηση ο master αναλάμβανε και τη μετατροπή των δεδομένων από byte σε αριθμούς κινητής υποδιαστολής, και τα δεδομένα αποστέλλονταν στους workers ως αριθμοί κινητής υποδιαστολής. Αυτό επέτρεπε στον κάθε worker να έχει μόνο δυο πίνακες κινητής υποδιαστολής με την τρέχουσα και προηγούμενη έκδοση της εικόνας. Οι αριθμοί κινητής υποδιαστολής όμως απαιτούν πολλαπλάσιο χώρο αποθήκευσης και κατά την αποστολή της εικόνας παίρναμε μηνύματα λάθους σχετικά με το MPI υποσύστημα και είχαμε τερματισμό των διεργασιών. Αυτό συνέβαινε όταν χρησιμοποιούσαμε μεγάλες εικόνες ή/και πλέγμα υπολογισμού με μικρό συνολικό πλήθος διεργασιών, ενδεχομένως λόγω κάποιου ανώτατου ορίου μεγέθους των MPI μηνυμάτων. Χρήση MPI-IO θα επέτρεπε σε κάθε διεργασία να ανοίξει το δικό της μέρος του αρχείου εικόνας άμεσα. Η λύση που επιλέξαμε τελικά ήταν απλά αποστολή των δεδομένων ως byte, με την πρόσθετη απαίτηση να υπάρχει ένας τρίτος πίνακας τύπου byte στις worker διεργασίες με τις διαστάσεις της τοπικής εικόνας για την λήψη/αποστολή της. Η χρήση byte μας επέτρεψε την αποστολή/λήψη όλων των διαστάσεων εικόνας σε RGB ακόμη και με πλέγμα 1x1 (master διεργασία και ένας worker). Ο παρακάτω πίνακας συνοψίζει τους περιορισμούς που αναφέραμε για την περίπτωση 4.00x RGB.

Πίνακας 2: Διαμοιρασμός εικόνας RGB 3840 x 5040 (4.00x) σε πλέγματα υπολογισμού μικρών διαστάσεων

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| double | | float | | byte | |
| 1x1 | Description: D:\Documents and Settings\John\Desktop\x.png | 1x1 | Description: D:\Documents and Settings\John\Desktop\x.png | 1x1 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 2x1 | Description: D:\Documents and Settings\John\Desktop\x.png | 2x1 | Description: D:\Documents and Settings\John\Desktop\x.png | 2x1 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 1x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 1x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 1x2 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 2x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 2x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 2x2 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 3x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 3x2 | Description: D:\Documents and Settings\John\Desktop\x.png | 3x2 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 2x3 | Description: D:\Documents and Settings\John\Desktop\x.png | 2x3 | Description: D:\Documents and Settings\John\Desktop\tick.png | 2x3 | Description: D:\Documents and Settings\John\Desktop\tick.png |
| 3x3 | Description: D:\Documents and Settings\John\Desktop\x.png | 3x3 | Description: D:\Documents and Settings\John\Desktop\tick.png | 3x3 | Description: D:\Documents and Settings\John\Desktop\tick.png |
|  |  |  |  |  |  |
|  |  | 1x3 | Description: D:\Documents and Settings\John\Desktop\tick.pngDescription: D:\Documents and Settings\John\Desktop\excl.png |  |  |
|  |  | 1x4 | Description: D:\Documents and Settings\John\Desktop\tick.pngDescription: D:\Documents and Settings\John\Desktop\excl.png |  |  |

Παρατηρούμε κάποιες ασυμμετρίες στην περίπτωση float, όπου σε ένα πλέγμα 2x2 δεν έχουμε επιτυχή μετάδοση, όμως σε 1x4 έχουμε επιτυχή, όπως και σε 1x3. Επίσης σε πλέγμα 3x2 έχουμε αποτυχία ενώ σε 2x3 επιτυχία. Φαίνεται ότι για κάποιο λόγο μικρός αριθμός γραμμών ευνοεί την επιτυχή μετάδοση. Σημειώνουμε ότι η αποστολή του τμήματος της εικόνας που αντιστοιχεί στον κάθε worker έγινε με απλή χρήση του MPI datatype vector, χωρίς scatter. Η λύση αυτή δεν είναι βέλτιστη, όμως βρίσκεται στο ακολουθιακό μέρος του προγράμματος και δεν επηρεάζει τους κυρίως υπολογισμούς και τις μετρήσεις.

# Ο πηγαίος κώδικας

## Περιγραφή της δομής του SPMD προγράμματος (master - worker)

|  |
| --- |
| ανάγνωση παραμέτρων  δημιουργία communicator που εξαιρεί την master διεργασία  master {  εκτύπωση παραμέτρων  ανάγνωση εικόνας από αρχείο  λήψη καρτεσιανών συντεταγμένων από τους workers  δημιουργία datatype για το τμήμα εικόνας  αποστολή τμημάτων εικόνας στους workers  λήψη επεξεργασμένων τμημάτων από τους workers  εγγραφή αποτελεσμάτων σε αρχεία  αποδέσμευση μνήμης  }  worker {  δημιουργία worker communicator (καρτεσιανές συντεταγμένες)    υπολογισμός καρτεσιανών συντεταγμένων στον worker communicator    αποστολή καρτεσιανών συντεταγμένων στον master    δημιουργία datatype για το τμήμα εικόνας    λήψη τμήματος εικόνας στον byte buffer    δημιουργία δυο buffer κινητής υποδιαστολής για το τμήμα εικόνας    αντιγραφή των δεδομένων του byte buffer στον ένα από τους buffer κινητής υποδιαστολής    προσδιορισμός όλων των rank των γειτονικών worker διεργασιών, αν υπάρχουν    δημιουργία datatypes row, column, corner για επικοινωνία μεταξύ των workers    εκκίνηση χρονομέτρησης    **κύριος βρόχος εφαρμογής του φίλτρου στο τμήμα εικόνας**    τέλος χρονομέτρησης    προσδιορισμός ελάχιστου, μεγίστου, μέσου χρόνου εκτέλεσης για τους workers    εκτύπωση στατιστικών χρονομέτρησης    αποστολή επεξεργασμένου τμήματος εικόνας πίσω στη master διεργασία    αποδέσμευση μνήμης  } |

Οι παράλληλες εκδόσεις του προγράμματος διαφοροποιούνται στο σημείο του κυρίου βρόχου εφαρμογής του φίλτρου. Για κάθε εκδοχή του προγράμματος αυτός θα παρουσιαστεί ξεχωριστά.

## Ανάγνωση παραμέτρων

Το πρόγραμμα διαβάζει από τις παραμέτρους της γραμμής εντολών τον αριθμό επαναλήψεων του φίλτρου (iterations), το κάθε πόσες επαναλήψεις θέλουμε να γίνεται έλεγχος σύγκλισης (convergence) και τον αριθμό γραμμών και στηλών του πλέγματος υπολογισμού (rows, columns). Ακολούθως υπολογίζει τις διαστάσεις του τμήματος εικόνας της κάθε worker διεργασίας (height, width) βασιζόμενο στις διαστάσεις της εικόνας εισόδου (HEIGHT, WIDTH). Οι συμβολικές σταθερές για τις διαστάσεις της εικόνας εισόδου και το πλήθος των καναλιών της (CHANNELS) καθώς και άλλες ρυθμίσεις, βρίσκονται στο αρχείο settings.h. Για εκτέλεση μέχρι τη σύγκλιση θέτουμε iterations 0. Αν iterations και convergence τεθούν και τα δυο 0, το πρόγραμμα δεν κάνει καμία επεξεργασία.

## Η δέσμευση, χρήση και αποδέσμευση μνήμης

Η αναπαράσταση των δεδομένων στη μνήμη διατηρεί τη δομή του αρχείου εισόδου, δηλαδή τριών byte στη σειρά που αντιστοιχούν στα κανάλια R,G,B αντίστοιχα. Με άλλα λόγια οι περιοχές μνήμης στις οποίες δουλεύουμε είναι δισδιάστατοι πίνακες που περιέχουν ως στοιχεία πίνακες byte με μέγεθος CHANNELS, που στην περίπτωση RGB είναι 3 και στην περίπτωση GREY 1. Όσον αφορά την πρόσβαση στα στοιχεία αυτά, θελήσαμε να διατηρήσουμε τη σύνταξη δείκτη σε δείκτη, έτσι ώστε ο τύπος να μπορεί να περαστεί εύκολα ως παράμετρος σε συναρτήσεις χωρίς να έχει πληροφορίες των διαστάσεων που χρησιμοποιούνται κάθε φορά (πλάτος γραμμών), και για να μην απαιτείται κανένας ιδιαίτερος υπολογισμός για τους δύο δείκτες i, j του δισδιάστατου πίνακα. Για να διατηρήσουμε την απαίτηση της συνεχόμενης μνήμης που επιτρέπει τη χρήση MPI datatypes χρησιμοποιήσαμε την παρακάτω συνάρτηση για τη δέσμευση μνήμης (αρχείο common/2d\_malloc.c):

|  |
| --- |
| 2 \* File: 2d\_malloc.c  ...  15 bool alloc\_uchar\_array(unsigned char \*\*\*array, int rows, int columns, int channels)  16 {  17 unsigned char \*p;  18 p = malloc(rows \* columns \* channels \* sizeof (unsigned char));  19 if (p == NULL)  20 {  21 perror("malloc");  22 return false;  23 }  24  25 (\*array) = malloc(rows \* sizeof (unsigned char \*));  26 if ((\*array) == NULL)  27 {  28 perror("malloc");  29 free(p);  30 return false;  31 }  32  33 int i;  34 for (i = 0; i < rows; i++)  35 (\*array)[i] = &(p[i \* columns \* channels]);  36  37 return true;  38 }  39  40 void dealloc\_uchar\_array(unsigned char \*\*\*array)  41 {  42 free(&((\*array)[0][0]));  43 free(\*array);  44 \*array = NULL;  45 }  ... |

Η συνάρτηση alloc\_uchar\_array() δεσμεύει έναν δισδιάστατο πίνακα με διαστάσεις rows x columns και στοιχεία πίνακες τύπου unsigned char [channels]. Πρόσβαση στο i, j στοιχείο του πίνακα αυτού μπορεί να γίνει απλά με array[i][j], παρ’ όλο που το array είναι δείκτης σε δείκτη σε unsigned char και όχι πραγματικός δισδιάστατος πίνακας (δείκτης σε γραμμή πίνακα). Αξίζει να σημειωθεί ότι ο πίνακας φαίνεται να είναι πίνακας με στοιχεία χαρακτήρες και όχι πίνακες χαρακτήρων, αλλά αυτό δεν μας εμποδίζει να χρησιμοποιήσουμε πίνακες χαρακτήρων ως στοιχεία κάνοντας τους σωστούς υπολογισμούς (πολλαπλασιασμός επί channels) κατά τη δέσμευση μνήμης. Η χρήση αυτού του σχήματος δέσμευσης μνήμης μας επιτρέπει ο κώδικας να είναι ιδιαίτερα καθαρός χωρίς υπολογισμούς για το i, κάτι που βοήθησε πολύ στην ανάπτυξη και τον έλεγχο ορθότητας του προγράμματος καθώς και τη μεγαλύτερη ευκολία για υποστήριξη φίλτρων συνέλιξης με διαστάσεις μεγαλύτερες από 3x3, όπως θα φανεί παρακάτω. Αντίστοιχη εκδοχή της συνάρτησης υπάρχει και για τον τύπο float, αφού οι δυο πίνακες πάνω στους οποίους δουλεύουν οι worker διεργασίες είναι κινητής υποδιαστολής ώστε να υποστηριχθούν σωστά οι αριθμητικές πράξεις πάνω στα τοπικά δεδομένα της εικόνας.

## Αποστολή και λήψη των τμημάτων εικόνας από τον master

|  |
| --- |
| 2 \* File: main\_async.c  ...  93 /\* Create "subarray" datatype. \*/  94  95 MPI\_Datatype local\_image\_t;  96 MPI\_Type\_vector(height, width \* CHANNELS, WIDTH \* CHANNELS,  MPI\_UNSIGNED\_CHAR, &local\_image\_t);  97 MPI\_Type\_commit(&local\_image\_t);  98  99 /\* Send each process its corresponding subarray. \*/  100  101 for (r = 0; r < rows \* columns; r++)  102 MPI\_Send(&(image\_buffer[coords[r][0] \* height][coords[r][1] \* width][0]), 1,  local\_image\_t, r + 1, 0, MPI\_COMM\_WORLD);  103  104 /\* Receive processed output from worker processes. \*/  105  106 for (r = 0; r < rows \* columns; r++)  107 MPI\_Recv(&(image\_buffer[coords[r][0] \* height][coords[r][1] \* width][0]), 1,  local\_image\_t, r + 1, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  ... |

Εδώ έχουμε τη δημιουργία ενός MPI datatype τύπου vector με τις εξής παραμέτρους:

* count: height
* blocklength: width \* CHANNELS
* stride: WIDTH \* CHANNELS

Αυτός ο datatype χρησιμοποιείται στις κλήσεις MPI\_Send() και MPI\_Recv(), προσδιορίζοντας το σημείο του πίνακα της αρχικής εικόνας από το οποίο θέλουμε να ξεκινά, ανάλογα με τις καρτεσιανές συντεταγμένες της κάθε worker διεργασίας.

## Δέσμευση μνήμης από τους workers και λήψη του τμήματος εικόνας

|  |
| --- |
| 2 \* File: main\_async.c  ...  145 /\* Allocate memory for local buffer. \*/  146  147 unsigned char (\*\*local\_buffer)[CHANNELS];  148  149 alloc\_uchar\_array((unsigned char \*\*\*) &local\_buffer,  B + height + B, B + width + B, CHANNELS);  150  151 /\* Create local buffer datatype for master receive/send. \*/  152  153 MPI\_Datatype local\_buffer\_t;  154 MPI\_Type\_vector(height, width \* CHANNELS, (B + width + B) \* CHANNELS,  MPI\_UNSIGNED\_CHAR, &local\_buffer\_t);  155 MPI\_Type\_commit(&local\_buffer\_t);  156  157 /\* Receive local image data from master. \*/  158  159 MPI\_Recv(&(local\_buffer[B][B][0]), 1, local\_buffer\_t, master, 0,  MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);  ... |

Εδώ έχουμε τη δημιουργία του byte buffer για τη λήψη/αποστολή του τμήματος της εικόνας που αντιστοιχεί στη worker διεργασία, χρησιμοποιώντας τη συνάρτηση alloc\_uchar\_array(). Οι διαστάσεις του buffer είναι αυτές του τμήματος εικόνας προσαυξημένες κατά Β + Β ούτως ώστε να φιλοξενηθούν τα δεδομένα των γειτονικών οριακών περιοχών. Ο MPI datatype που δημιουργείται είναι και πάλι vector με τις εξής παραμέτρους:

* count: height
* blocklength: width \* CHANNELS
* stride: (B + width + B) \* CHANNELS

Αυτός o datatype ακολούθως χρησιμοποιείται στην κλήση MPI\_Recv() για να τοποθετήσει τα εισερχόμενα δεδομένα στη σωστή θέση του buffer (εντός των οριακών περιοχών) με διεύθυνση &(local\_buffer[B][B][0]).

## Δέσμευση των δυο πινάκων κινητής υποδιαστολής για τους υπολογισμούς

|  |
| --- |
| 2 \* File: main\_async.c  ...  164 /\* Allocate two 2d float arrays for image processing. \*/  165  166 float (\*\*image\_a)[CHANNELS];  167 float (\*\*image\_b)[CHANNELS];  168  169 alloc\_float\_array((float \*\*\*) &image\_a, B + height + B, B + width + B, CHANNELS);  170 alloc\_float\_array((float \*\*\*) &image\_b, B + height + B, B + width + B, CHANNELS);  171  172 /\* Declare current and previous image pointers, used for switching buffers. \*/  173  174 float (\*\*curr\_image)[CHANNELS] = image\_a;  175 float (\*\*prev\_image)[CHANNELS] = image\_b;  176  177 /\* Copy uchar buffer data to current image, converting to float for arithmetic operations. \*/  178  179 for (i = 0; i < B + height + B; i++)  180 for (j = 0; j < B + width + B; j++)  181 for (c = 0; c < CHANNELS; c++)  182 curr\_image[i][j][c] = (float) local\_buffer[i][j][c];  ... |

Αμέσως μετά τη λήψη των δεδομένων έχουμε τη δέσμευση δύο πινάκων κινητής υποδιαστολής για την επεξεργασία του τμήματος εικόνας που αντιστοιχεί στη worker διεργασία. Χρησιμοποιείται η συνάρτηση alloc\_float\_array() η οποία είναι εντελώς ανάλογη της alloc\_uchar\_array() που είδαμε νωρίτερα. Η συγκεκριμένη υλοποίηση (async) χρησιμοποιεί δυο σταθερούς δείκτες (image\_a και image\_b) και δυο βοηθητικούς δείκτες (curr\_image και prev\_image) των οποίων οι τιμές εναλλάσσονται μεταξύ των δύο σταθερών δεικτών. Η χρήση των βοηθητικών δεικτών διευκολύνει το να συμπεράνουμε σε κάθε επανάληψη της εφαρμογής του φίλτρου ποιος πίνακας έχει την τρέχουσα έκδοση της τοπικής εικόνας. Αμέσως μετά τη δέσμευση της μνήμης αντιγράφονται στον τρέχοντα πίνακα κινητής υποδιαστολής τα δεδομένα που έχουν μόλις ληφθεί και βρίσκονται στον byte πίνακα.

## Προσδιορισμός των rank των γειτονικών worker διεργασιών

|  |
| --- |
| 2 \* File: main\_async.c  ...  184 /\* Get neighboring process ranks. \*/  185  186 int r\_n, r\_s, r\_e, r\_w;  187 int r\_ne, r\_nw, r\_se, r\_sw;  188  189 get\_neighbors(comm\_workers, &r\_n, &r\_s, &r\_e, &r\_w, &r\_nw, &r\_se, &r\_ne, &r\_sw);  ... |

Για το σκοπό αυτό χρησιμοποιείται η συνάρτηση get\_neighbors() (αρχείο common/topology.c). Ο communicator που περνιέται ως πρώτη παράμετρος έχει πληροφορίες καρτεσιανής τοπολογίας και με τη βοήθεια της συνάρτησης MPI\_Cart\_shift() προσδιορίζονται τα ranks των γειτονικών workers. Αν δεν υπάρχει κάποια γειτονική διεργασία, το αντίστοιχο rank παίρνει την τιμή MPI\_PROC\_NULL.

## Δημιουργία των datatypes για την επικοινωνία μεταξύ των workers

|  |
| --- |
| 2 \* File: main\_async.c  ...  191 /\* Create border datatypes for communication between workers. \*/  192  193 MPI\_Datatype row\_t, column\_t, corner\_t;  194  195 MPI\_Type\_vector(B, width \* CHANNELS, (B + width + B) \* CHANNELS, MPI\_FLOAT, &row\_t);  196 MPI\_Type\_commit(&row\_t);  197  198 MPI\_Type\_vector(height, B \* CHANNELS, (B + width + B) \* CHANNELS, MPI\_FLOAT, &column\_t);  199 MPI\_Type\_commit(&column\_t);  200  201 MPI\_Type\_vector(B, B \* CHANNELS, (B + width + B) \* CHANNELS, MPI\_FLOAT, &corner\_t);  202 MPI\_Type\_commit(&corner\_t);  ... |

Εδώ έχουμε τη δημιουργία τριών datatypes, για γραμμή, στήλη και γωνιακή οριακή περιοχή της εικόνας. Χαρακτηριστική είναι η παράμετρος Β με την οποία μπορούμε να προσαρμόσουμε το πρόγραμμα για διαφορετικές διαστάσεις φίλτρου. Χρησιμοποιούμε B = 1 για φίλτρο 3x3, Β = 2 για φίλτρο 5x5, και ούτω καθεξής. Σαν αποτέλεσμα, οι διαστάσεις των οριακών περιοχών είναι οι εξής:

* γραμμή: (width x B)
* στήλη: (B x height)
* γωνιακή περιοχή: (B x B)

Τα δεδομένα ανταλλάσσονται μεταξύ των workers ως αριθμοί κινητής υποδιαστολής (MPI\_FLOAT), πράγμα που εδώ δε δημιουργεί πρόβλημα λόγω του σχετικά μικρού μεγέθους των μηνυμάτων.

## Ο κύριος βρόχος επεξεργασίας με ασύγχρονη MPI επικοινωνία

Μια συνοπτική περιγραφή του έχει ως εξής:

|  |
| --- |
| ανταλλαγή οριακών δεδομένων με υπάρχοντες γείτονες (non-blocking επικοινωνία, MPI\_Isend - MPI\_Irecv)  εφαρμογή εσωτερικού φίλτρου (δεν απαιτούνται οριακά δεδομένα)  συμπλήρωση οριακών δεδομένων για γείτονες που δεν υπάρχουν  αναμονή για τη λήψη μηνυμάτων (MPI\_Waitall)  συμπλήρωση οριακών δεδομένων για διαγώνιες περιπτώσεις που απαιτούν τις λήψεις να έχουν ολοκληρωθεί  εφαρμογή εξωτερικού φίλτρου (απαιτεί όλα τα οριακά δεδομένα να έχουν συμπληρωθεί)  αναμονή για την αποστολή μηνυμάτων (MPI\_Waitall)  έλεγχος σύγκλισης |

Αρχικά έχουμε την ανταλλαγή οριακών δεδομένων μεταξύ των worker διεργασιών:

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  225 /\* Send / receive vertical data. \*/  226  227 if (r\_s != MPI\_PROC\_NULL) // sendrecv south  228 {  229 MPI\_Isend(&(curr\_image[height][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &sends[p++]);  230 MPI\_Irecv(&(curr\_image[height + B][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &recvs[q++]);  231 }  232  233 if (r\_n != MPI\_PROC\_NULL) // sendrecv north  234 {  235 MPI\_Isend(&(curr\_image[B][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &sends[p++]);  236 MPI\_Irecv(&(curr\_image[0][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &recvs[q++]);  237 }  ... |

Δεν έχουμε κάποια ανάγκη συντονισμού των διεργασιών αφού η επικοινωνία είναι ασύγχρονη. Αμέσως μετά έχουμε την επικάλυψη της επικοινωνίας που μόλις ξεκίνησε με όσο το δυνατόν περισσότερους υπολογισμούς που μπορούν να γίνουν με τοπικά δεδομένα. Αρχικά έχουμε την εφαρμογή του εσωτερικού φίλτρου:

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  279 /\* Apply inner filter, does not require having border data available. \*/  280  281 apply\_inner\_filter(prev\_image, curr\_image, B + height + B, B + width + B);  ... |

Ακολούθως, στην περίπτωση που κάποιος γείτονας δεν υπάρχει, συμπληρώνουμε τα οριακά δεδομένα με τις ακραίες τιμές της τοπικής εικόνας. Εξαίρεση αποτελούν κάποιες διαγώνιες περιπτώσεις στις οποίες πρέπει πρώτα να περιμένουμε τη λήψη από έναν οριζόντιο η κατακόρυφο γείτονα ώστε να συμπληρώσουμε τη γωνιακή περιοχή με τα δεδομένα που μας έχει στείλει. Ο συγκεκριμένος χειρισμός των διαγώνιων οριακών περιοχών κάνει την εφαρμογή του φίλτρου απόλυτα σταθερή για συγκεκριμένο φίλτρο και αριθμό επαναλήψεων, ανεξάρτητα από τις διαστάσεις του πλέγματος υπολογισμού που καθορίζουν την υποδιαίρεση της εικόνας σε τμήματα.

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  283 /\* If a neighbor is null, fill border buffer with edge image data. \*/  284  285 if (r\_s == MPI\_PROC\_NULL)  286 for (i = height + B; i < height + 2 \* B; i++)  287 for (j = B; j < B + width; j++)  288 for (c = 0; c < CHANNELS; c++)  289 curr\_image[i][j][c] = curr\_image[B + height - 1][j][c];  290  291 if (r\_n == MPI\_PROC\_NULL)  292 for (i = 0; i < B; i++)  293 for (j = B; j < B + width; j++)  294 for (c = 0; c < CHANNELS; c++)  295 curr\_image[i][j][c] = curr\_image[B][j][c];  ...  323 if (r\_sw == MPI\_PROC\_NULL)  324 if (r\_s == MPI\_PROC\_NULL && r\_w == MPI\_PROC\_NULL) // use corner data  325 for (i = height + B; i < height + 2 \* B; i++)  326 for (j = 0; j < B; j++)  327 for (c = 0; c < CHANNELS; c++)  328 curr\_image[i][j][c] = curr\_image[B + height - 1][B][c];  329  330 if (r\_ne == MPI\_PROC\_NULL)  331 if (r\_n == MPI\_PROC\_NULL && r\_e == MPI\_PROC\_NULL) // use corner data  332 for (i = 0; i < B; i++)  333 for (j = width + B; j < width + 2 \* B; j++)  334 for (c = 0; c < CHANNELS; c++)  335 curr\_image[i][j][c] = curr\_image[B][B + width - 1][c];  ... |

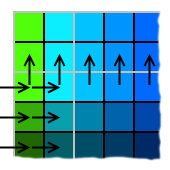
Οι δύο τελευταίες περιπτώσεις είναι διαγώνια δεδομένα στα οποία γνωρίζουμε ότι δεν υπάρχει κανένας από τους δυο σχετικούς γείτονες (οριζόντια - κάθετα), οπότε τα δεδομένα πρέπει να συμπληρωθούν από την τοπική εικόνα. Ουσιαστικά πρόκειται για τις γωνιακές περιοχές της συνολικής εικόνας. Μετά τη συμπλήρωση των οριακών δεδομένων έχουμε αναμονή για τις λήψεις μηνυμάτων.

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  337 /\* Wait for recvs. \*/  338  339 MPI\_Waitall(q, recvs, recv\_status);  ... |

Ακολουθεί η συμπλήρωση των γωνιακών δεδομένων που απαιτούν να έχουν ολοκληρωθεί οι λήψεις:

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  341 /\* Handle diagonal border data cases that require recvs to have completed. \*/  342  343 if (r\_se == MPI\_PROC\_NULL) // southeast  344 {  345 if (r\_s != MPI\_PROC\_NULL) // get data from south (received)  346 for (i = height + B; i < height + 2 \* B; i++)  347 for (j = width + B; j < width + 2 \* B; j++)  348 for (c = 0; c < CHANNELS; c++)  349 curr\_image[i][j][c] = curr\_image[i][B + width - 1][c];  350 else if (r\_e != MPI\_PROC\_NULL) // get data from east (received)  351 for (i = height + B; i < height + 2 \* B; i++)  352 for (j = width + B; j < width + 2 \* B; j++)  353 for (c = 0; c < CHANNELS; c++)  354 curr\_image[i][j][c] = curr\_image[B + height - 1][j][c];  355 }  ... |

Ακολούθως παραθέτουμε μια εικόνα που αναπαριστά ένα παράδειγμα μιας τέτοιας διαγώνιας περίπτωσης, συμπλήρωση βορειοδυτικής γωνιακής περιοχής όταν έχουμε δυτική αλλά όχι βόρεια γειτονική διεργασία. Το παράδειγμα είναι με B = 2, για φίλτρο 5x5.



Σχήμα 1: Συμπλήρωση βορειοδυτικής γωνιακής περιοχής όταν υπάρχει δυτικός αλλά όχι βόρειος γείτονας. (Β = 2)

Η συμπλήρωση και των περιπτώσεων αυτών ολοκληρώνει τη δημιουργία όλων των απαιτούμενων οριακών δεδομένων για την εφαρμογή και του εξωτερικού φίλτρου το οποίο ακολουθεί αμέσως μετά:

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  399 /\* Apply outer filter, requires having all border data available. \*/  400  401 apply\_outer\_filter(prev\_image, curr\_image, B + height + B, B + width + B);  ... |

Η εφαρμογή του φίλτρου σηματοδοτεί το τέλος της επεξεργασίας για την τρέχουσα επανάληψη. Πριν την εναλλαγή τρέχουσας - προηγούμενης εικόνας πρέπει να είμαστε βέβαιοι ότι έχουν ολοκληρωθεί και οι αποστολές μηνυμάτων αφού χρησιμοποιούν δεδομένα της τρέχουσας εικόνας.

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  403 /\* Wait for sends before we switch buffers. \*/  404  405 MPI\_Waitall(p, sends, send\_status);  406  407 /\* Switch current / previous image buffers. \*/  408  409 float (\*\*temp)[CHANNELS];  410 temp = curr\_image;  411 curr\_image = prev\_image;  412 prev\_image = temp;  ... |

Τέλος, για την ολοκλήρωση μιας επανάληψης της εφαρμογής του φίλτρου, έχουμε τον έλεγχο σύγκλισης. Χρησιμοποιείται ο τελεστής MPI\_LAND με τη συνάρτηση MPI\_Allreduce() αφού στην περίπτωση σύγκλισης πρέπει να ενημερωθούν όλες οι διεργασίες για να σταματήσουν την επεξεργασία:

|  |
| --- |
| 2 \* File: main\_async\_nonper.c  ...  416 if (convergence > 0 && n % convergence == 0)  417 {  418 int identical = images\_identical(curr\_image, prev\_image,  B + height + B, B + width + B) ? 1 : 0;  419 int all\_identical = 0;  420  421 MPI\_Allreduce(&identical, &all\_identical, 1, MPI\_INT, MPI\_LAND, comm\_workers);  422  423 if (all\_identical)  424 {  425 if (worker\_rank == 0)  426 printf("Filter has converged after %d iterations.\n", n);  427  428 break;  429 }  430 }  ... |

Πέρα από την υλοποίηση που είδαμε η οποία είναι με non-persistent επικοινωνία (βρίσκεται στο αρχείο async/main\_async\_nonper.c), έχει υλοποιηθεί και μια ασύγχρονη εκδοχή με persistent επικοινωνία (async/main\_async.c). Πέρα από το στάδιο της προετοιμασίας των requests πριν τον κύριο βρόχο, βασική διαφοροποίηση σε σχέση με την προηγούμενη υλοποίηση είναι και η ανάγκη να έχουμε δυο σύνολα requests, ένα για τον κάθε buffer που έχουμε δεσμεύσει για το τμήμα εικόνας:

|  |
| --- |
| 2 \* File: main\_async.c  ...  206 MPI\_Request sends\_a[8];  207 MPI\_Request recvs\_a[8];  208 MPI\_Request sends\_b[8];  209 MPI\_Request recvs\_b[8];  210 MPI\_Status send\_status[8];  211 MPI\_Status recv\_status[8];  212  213 unsigned int p = 0, q = 0;  214  217 if (r\_s != MPI\_PROC\_NULL) // sendrecv south  218 {  219 MPI\_Send\_init(&(image\_a[height][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &sends\_a[p]);  220 MPI\_Send\_init(&(image\_b[height][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &sends\_b[p]);  221 MPI\_Recv\_init(&(image\_a[height + B][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &recvs\_a[q]);  222 MPI\_Recv\_init(&(image\_b[height + B][B][0]), 1, row\_t, r\_s, 0, comm\_workers, &recvs\_b[q]);  223 p++;  224 q++;  225 }  226  227 if (r\_n != MPI\_PROC\_NULL) // sendrecv north  228 {  229 MPI\_Send\_init(&(image\_a[B][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &sends\_a[p]);  230 MPI\_Send\_init(&(image\_b[B][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &sends\_b[p]);  231 MPI\_Recv\_init(&(image\_a[0][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &recvs\_a[q]);  232 MPI\_Recv\_init(&(image\_b[0][B][0]), 1, row\_t, r\_n, 0, comm\_workers, &recvs\_b[q]);  233 p++;  234 q++;  235 }  ... |

Τη στιγμή που γίνεται η εκκίνηση μιας αποστολής ή λήψης μέσα στο βρόχο, τρέχων buffer μπορεί να είναι οποιοσδήποτε από τους δύο που έχουν δεσμευθεί. Στην αρχή του βρόχου αποφασίζεται ποια σύνολα requests θα χρησιμοποιηθούν σύμφωνα με τον ενεργό buffer, ουσιαστικά ανάλογα με το αν είμαστε σε άρτιο ή περιττό αριθμό επανάληψης:

|  |
| --- |
| 2 \* File: main\_async.c  ...  310 /\* Select appropriate sends/recvs depending on active image buffer. \*/  311  312 MPI\_Request \*sends = (curr\_image == image\_a) ? (sends\_a) : (curr\_image == image\_b ? sends\_b :  NULL);  313 MPI\_Request \*recvs = (curr\_image == image\_a) ? (recvs\_a) : (curr\_image == image\_b ? recvs\_b :  NULL);  314  315 /\* Reset send/recv indexes. \*/  316  317 p = 0;  318 q = 0;  319  320 /\* Send / receive vertical data. \*/  321  322 if (r\_s != MPI\_PROC\_NULL) // sendrecv south  323 {  324 MPI\_Start(&sends[p++]);  325 MPI\_Start(&recvs[q++]);  326 }  327  328 if (r\_n != MPI\_PROC\_NULL) // sendrecv north  329 {  330 MPI\_Start(&sends[p++]);  331 MPI\_Start(&recvs[q++]);  332 }  ... |

Πέρα από την αντικατάσταση των MPI\_Isend(), MPI\_Irecv() με MPI\_Start(), η εκδοχή αυτή δε διαφοροποιείται δομικά σε σχέση με αυτή χωρίς persistent επικοινωνία.

## Ο κύριος βρόχος επεξεργασίας με σύγχρονη MPI επικοινωνία

Μια συνοπτική περιγραφή του έχει ως εξής:

|  |
| --- |
| ανταλλαγή οριακών δεδομένων με υπάρχοντες γείτονες (blocking επικοινωνία, MPI\_Send - MPI\_Recv)  συμπλήρωση οριακών δεδομένων για γείτονες που δεν υπάρχουν  εφαρμογή φίλτρου  έλεγχος σύγκλισης |

Το κύριο χαρακτηριστικό του σύγχρονου βρόχου είναι η ανάγκη συντονισμού των worker διεργασιών ώστε να γίνει δυνατή η σωστή ανταλλαγή των μηνυμάτων. Στο αρχείο common/topology.c υπάρχουν δυο συναρτήσεις που χρησιμοποιούνται γι αυτό το σκοπό και απλά αποφασίζουν αν μια διεργασία βρίσκεται σε άρτιο ή περιττό αριθμό γραμμής ή στήλης σύμφωνα με τις πληροφορίες τοπολογίας του worker communicator. Η κλήση τους γίνεται μια φορά, πριν την εκτέλεση του κυρίου βρόχου του φίλτρου. Η ανταλλαγή μηνυμάτων στον κύριο βρόχο έχει ως εξής:

|  |
| --- |
| 2 \* File: main\_sync.c  ...  218 /\* Send / receive vertical data. \*/  219  220 if (even\_row)  221 {  222 if (r\_s != MPI\_PROC\_NULL) // sendrecv south  223 {  224 MPI\_Send(&(curr\_image[height][B][0]), 1, row\_t, r\_s, 0, comm\_workers);  225 MPI\_Recv(&(curr\_image[height + B][B][0]), 1, row\_t, r\_s, 0, comm\_workers,  MPI\_STATUS\_IGNORE);  226 }  227  228 if (r\_n != MPI\_PROC\_NULL) // sendrecv north  229 {  230 MPI\_Send(&(curr\_image[B][B][0]), 1, row\_t, r\_n, 0, comm\_workers);  231 MPI\_Recv(&(curr\_image[0][B][0]), 1, row\_t, r\_n, 0, comm\_workers, MPI\_STATUS\_IGNORE);  232 }  233 } else // odd row  234 {  235 if (r\_n != MPI\_PROC\_NULL) // sendrecv north  236 {  237 MPI\_Recv(&(curr\_image[0][B][0]), 1, row\_t, r\_n, 0, comm\_workers, MPI\_STATUS\_IGNORE);  238 MPI\_Send(&(curr\_image[B][B][0]), 1, row\_t, r\_n, 0, comm\_workers);  239 }  240  241 if (r\_s != MPI\_PROC\_NULL) // sendrecv south  242 {  243 MPI\_Recv(&(curr\_image[height + B][B][0]), 1, row\_t, r\_s, 0, comm\_workers,  MPI\_STATUS\_IGNORE);  244 MPI\_Send(&(curr\_image[height][B][0]), 1, row\_t, r\_s, 0, comm\_workers);  245 }  246 }  ... |

Αναλόγως γίνεται και η ανταλλαγή των οριζοντίων και διαγωνίων οριακών δεδομένων. Μετά την ανταλλαγή των οριακών δεδομένων απλά εφαρμόζεται το εσωτερικό και εξωτερικό μέρος του φίλτρου αφού όλα τα δεδομένα είναι διαθέσιμα μετά το τέλος της σύγχρονης ανταλλαγή μηνυμάτων. Στο τέλος του βρόχου γίνεται έλεγχος σύγκλισης μεταξύ της τρέχουσας και προηγούμενης έκδοσης της εικόνας, όπως και στην ασύγχρονη έκδοση.

## Υβριδική MPI/OpenMP υλοποίηση, ασύγχρονη MPI επικοινωνία

Έχουν υλοποιηθεί δύο εκδοχές του ασύγχρονου υβριδικού προγράμματος, μια με τη δημιουργία - καταστροφή νημάτων σε κάθε επανάληψη του φίλτρου (async/main\_async\_omp\_simple.c), και μια με τη δημιουργία τους μια φορά, πριν την εκτέλεση του κυρίου βρόχου (async/main\_async\_omp.c). Και οι δυο εκδοχές χρησιμοποιούν persistent MPI επικοινωνία. Στην πρώτη περίπτωση απλά έχουμε την εξής αλλαγή στο SPMD πρόγραμμα κατά την εφαρμογή του εσωτερικού μέρους του φίλτρου:

|  |
| --- |
| 2 \* File: main\_async\_omp\_simple.c  ...  374 #pragma omp parallel  375 {  380 /\* Apply inner filter using omp for, does not require having border data available. \*/  381  382 apply\_inner\_filter\_openmp(prev\_image, curr\_image, B + height + B, B + width + B);  383 }  ... |

Το εσωτερικό φίλτρο έχει μόνη προσθήκη την οδηγία για διαμοιρασμό του for στα διαθέσιμα νήματα:

|  |
| --- |
| 2 \* File: filter.c  ...  120 void apply\_inner\_filter\_openmp(float (\*\*output\_image)[CHANNELS], float (\*\*input\_image)[CHANNELS],  int height, int width)  121 {  122 unsigned int i, j, c;  123 int p, q;  124  125 #pragma omp for  126 for (i = 2 \* B; i < height - 2 \* B; i++)  127 {  128 for (j = 2 \* B; j < width - 2 \* B; j++)  129 {  130 for (c = 0; c < CHANNELS; c++)  131 {  132 float value = 0.0f;  133  134 for (p = -B; p <= B; p++)  135 for (q = -B; q <= B; q++)  136 value += input\_image[i - p][j - q][c] \* filter[p + B][q + B];  137  138 output\_image[i][j][c] = value;  139 }  140 }  141 }  142 }  ... |

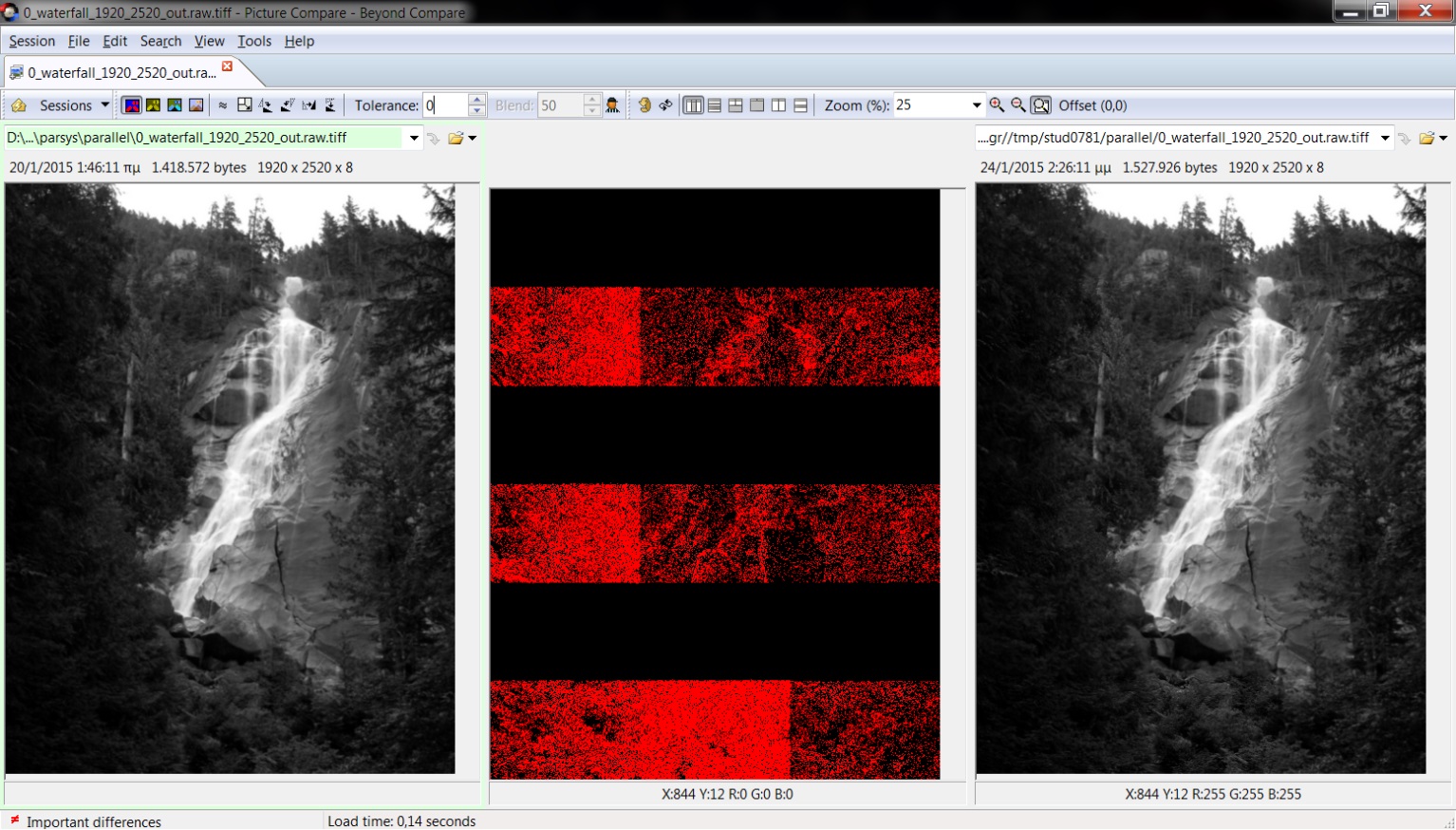
Στην εκδοχή με τη δημιουργία των νημάτων μία μόνο φορά, η αρχή του βρόχου έχει ως εξής:

|  |
| --- |
| 2 \* File: main\_async\_omp.c  ...  308 #pragma omp parallel private (i,j,c)  309 {  310 #pragma omp single  311 if (worker\_rank == 0)  312 printf("Threads: %d\n", omp\_get\_num\_threads());  313  314 unsigned int n;  315  316 for (n = 0; !converged &&  (iterations == 0 || n < iterations) && (iterations != 0 || convergence != 0); n++)  317 {  ... |

Στο εσωτερικό του βρόχου, επειδή o χειρισμός της MPI επικοινωνίας γίνεται από το master νήμα, είχαμε σαν συνέπεια αυτό να πρέπει να αναλάβει και άλλες εργασίες που εξαρτώνται από την ολοκλήρωση αποστολών ή λήψεων MPI μηνυμάτων. Ουσιαστικά η μόνη εργασία που διαμοιράζεται είναι η εφαρμογή του εσωτερικού μέρους του φίλτρου, όπως και στην απλή εκδοχή. Μια εναλλακτική, η οποία δε χρησιμοποιήθηκε τελικά, ήταν τα νήματα να μοιράζονται κάποιες εργασίες όπως η συμπλήρωση οριακών περιοχών με τοπικά δεδομένα αλλά αυτό έκανε αναγκαίο το συγχρονισμό των νημάτων μέσα στο βρόχο. Στην τελική υλοποίηση, έχουμε συγχρονισμό των νημάτων μόνο μία φορά, στο τέλος του βρόχου:

|  |
| --- |
| 2 \* File: main\_async\_omp.c  ...  541 /\* Threads should not move to next iteration until master thread has switched buffers. \*/  542 #pragma omp barrier  543  544 }  545 }  ... |

Χωρίς τη χρήση του barrier στο τέλος του βρόχου, κατά την εκτέλεση με δύο νήματα είχαμε σαν αποτέλεσμα το δεύτερο νήμα να προχωρά στις επόμενες επαναλήψεις χωρίς να περιμένει το master νήμα να ολοκληρώσει σημαντικές εργασίες όπως η αποστολή / λήψη μηνυμάτων ή η εναλλαγή των buffers, με ενδιαφέροντα αποτελέσματα στην έξοδο του προγράμματος:



Σχήμα 2: Σύγκριση αποτελεσμάτων υβριδικού προγράμματος χωρίς barrier με τα  
σωστά αποτελέσματα. (3x3 πλέγμα υπολογισμού, δύο νήματα ανά διεργασία)

Οι διαφορές δεν είναι εύκολα ορατές με γυμνό μάτι. Η απόκλιση από τα αναμενόμενα αποτελέσματα φαίνεται με κόκκινο χρώμα. Το πλέγμα υπολογισμού είναι 3x3. Το master νήμα κάθε διεργασίας, το οποίο ολοκληρώνει σωστά την MPI επικοινωνία και εναλλαγή των buffers πριν προχωρήσει στην επόμενη επανάληψη παράγει σωστά αποτελέσματα (πάνω μισό κάθε διεργασίας), σε αντίθεση με το δεύτερο νήμα που προχωράει ανεξέλεγκτα δουλεύοντας πάνω σε μη έγκυρα δεδομένα (κάτω μισό κάθε διεργασίας). Και εδώ η υλοποίηση με βάση τη σταθερότητα του φίλτρου όσον αφορά τα παραγόμενα αποτελέσματα βοήθησε στον εντοπισμό ενός αρχικά μη αναμενόμενου προβλήματος.

## Υβριδική MPI/OpenMP υλοποίηση, σύγχρονη MPI επικοινωνία

Όπως και στην ασύγχρονη περίπτωση, έτσι και στη σύγχρονη έχουν υλοποιηθεί δυο εκδοχές του υβριδικού προγράμματος, μια με τη δημιουργία και καταστροφή των νημάτων σε κάθε επανάληψη εφαρμογής του φίλτρου (sync/main\_sync\_omp\_simple.c), και μια με τη δημιουργία νημάτων μια φορά μόνο, πριν την εκτέλεση του κυρίου βρόχου (sync/main\_sync\_omp.c). Και οι δύο βασίζονται στη μονονηματική MPI εκδοχή με σύγχρονη επικοινωνία, και οι αλλαγές που έχουν γίνει σε σχέση με αυτή ακολουθούν ίδια λογική με αυτή της ασύγχρονης υλοποίησης.

# Μετρήσεις

## Hardware

Οι μετρήσεις MPI/OpenMP έγιναν στο εργαστήριο Linux του τμήματος, χρησιμοποιώντας έως και 16 από τις μηχανές με αριθμό μέχρι και 22. Οι μηχανές με αριθμό 23 και πάνω είναι πιο σύγχρονες και όταν συμπεριλαμβάνονταν στο πλέγμα υπολογισμού μας έδιναν εσφαλμένα γρηγορότερα αποτελέσματα (υπέρ-γραμμική επιτάχυνση την οποία αρχικά είχαμε αποδώσει σε άλλους πιθανούς παράγοντες). Χαρακτηριστική είναι η παρατήρηση ότι οι νεότερες μηχανές αν και πιο γρήγορες, έχουν μικρότερη συχνότητα λειτουργίας από τις παλαιότερες Pentium D, πράγμα που δείχνει τη στροφή του υλικού προς περισσότερους και πιο αποδοτικούς πυρήνες και την ανακοπή της συνεχούς αύξησης των συχνοτήτων χρονισμού των επεξεργαστών.

linux01: product: Intel(R) Pentium(R) D CPU 2.80GHz

linux02: product: Intel(R) Pentium(R) D CPU 2.80GHz

...

linux20: product: Intel(R) Pentium(R) D CPU 2.80GHz

linux22: product: Intel(R) Pentium(R) D CPU 2.80GHz

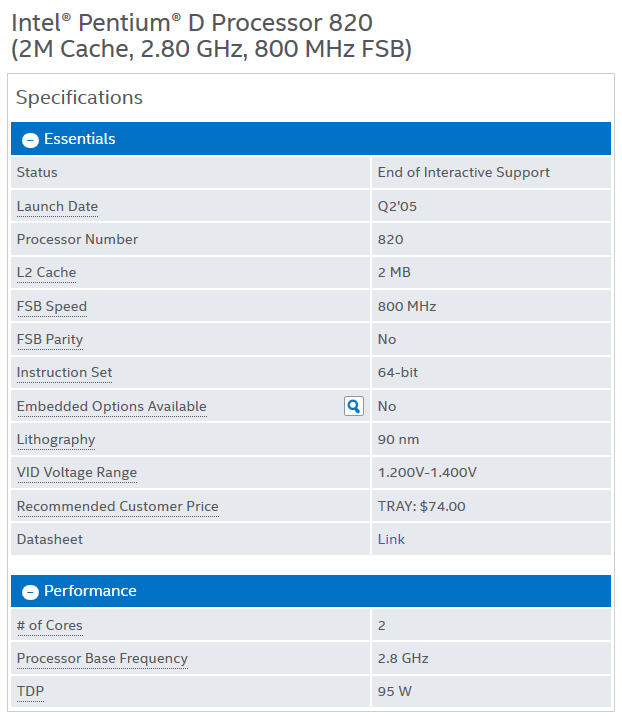
linux23: product: Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz

linux25: product: Intel(R) Core(TM)2 Duo CPU E6550 @ 2.33GHz

linux26: product: Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz

linux27: product: Intel(R) Core(TM)2 Duo CPU E6750 @ 2.66GHz

...



Σχήμα 3: Μέρος των προδιαγραφών του επεξεργαστή Intel Pentium D 820

Ο επεξεργαστής Pentium D 820 δεν υποστηρίζει τεχνολογία Hyper-Threading, οπότε είχαμε διαθέσιμα δυο λογικά νήματα, όσοι και οι πυρήνες του επεξεργαστή. Στις μετρήσεις με MPI επιλέξαμε την εκτέλεση μιας διεργασίας ανά μηχανή, για να αποφύγουμε ασυμμετρίες που έχουν να κάνουν με το αν η διεργασία επικοινωνεί με διεργασία που τρέχει στην ίδια η διαφορετική μηχανή. Έτσι είχαμε προβλέψιμο latency και throughput στην επικοινωνία, όσο προσφέρει το δίκτυο που συνδέει τους υπολογιστές. Η εκτέλεση μιας διεργασίας ανά επεξεργαστή μας επέτρεψε επίσης στις μετρήσεις των υβριδικών υλοποιήσεων MPI/OpenMP να έχουμε δύο διαθέσιμους πυρήνες χωρίς φόρτο που μπορούν να αξιοποιηθούν από ισάριθμα νήματα.

## Τρόπος λήψης μετρήσεων

Στις μετρήσεις MPI/OpenMP, κάθε υλοποίηση εκτελέστηκε 10 φορές για όλες τις διαστάσεις πλέγματος και κάθε μέγεθος προβλήματος τόσο στην RGB όσο και στη GREY περίπτωση. Σε κάθε εκτέλεση το πρόγραμμα εμφανίζει στατιστικά ελάχιστου, μέγιστου και μέσου χρόνου εκτέλεσης για κάθε πυρήνα. Αντί για τον ελάχιστο χρησιμοποιήσαμε τον μέσο χρόνο εκτέλεσης και ελήφθη υπ’ όψιν η ελάχιστη τιμή του. Στους χρόνους οι οποίοι εμφανίζονταν σε δευτερόλεπτα έγινε στρογγυλοποίηση σε τρία δεκαδικά ψηφία. Στις περιπτώσεις που χρειάστηκε στρογγύλευση προς τα κάτω, αυτή έγινε χωρίς δισταγμό αν κάποια μηχανή είχε καλύτερες επιδόσεις από τη στρογγυλοποιημένη τιμή και με μεγαλύτερη προσοχή σε αντίθετη περίπτωση. Μια μηχανή θα έπρεπε να έχει φτάσει εξαιρετικά κοντά στη στρογγυλοποιημένη τιμή, αλλιώς γινόταν στρογγύλευση προς τα επάνω.

## MPI υλοποίηση

### Περιγραφή μετρήσεων

Στην περίπτωση RGB πήραμε μετρήσεις για 100 επαναλήψεις εφαρμογής του φίλτρου ενώ στην περίπτωση GREY επιλέξαμε 200 επαναλήψεις. Οι αύξουσες καμπύλες είναι αυτές του Speedup και οι φθίνουσες του Efficiency. Το Efficiency έχει κανονικοποιηθεί στο 10 για να είναι εύκολα ορατό στα διαγράμματα. Χρόνος αναφοράς είναι η μέτρηση για πλέγμα υπολογισμού 1x1 η οποία πρακτικά συμπίπτει με αυτή του καθαρά ακολουθιακού προγράμματος. Οι μετρήσεις για την ασύγχρονη περίπτωση είναι αυτές της υλοποίησης με persistent επικοινωνία (main\_async). Ακολουθούν τα διαγράμματα μετρήσεων για κάθε μέγεθος εικόνας σε RGB και GREY καθώς και οι αντίστοιχοι πίνακες. Παρουσιάζονται σχόλια όπου κρίνεται απαραίτητο - βοηθητικό. Στο τέλος υπάρχουν συγκεντρωτικοί πίνακες Speedup / Efficiency και αμέσως μετά μια σύγκριση της ασύγχρονης υλοποίησης με persistent επικοινωνία με αυτή χωρίς persistent επικοινωνία.

### Διαγράμματα - πίνακες μετρήσεων

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 9,493s | 9,474s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 5,011s | 4,781s | 1,89 | 1,98 | 0,95 | 0,99 |
| 4 | 2,754s | 2,421s | 3,45 | 3,91 | 0,86 | 0,98 |
| 6 | 1,953s | 1,632s | 4,86 | 5,81 | 0,81 | 0,97 |
| 9 | 1,402s | 1,113s | 6,77 | 8,51 | 0,75 | 0,95 |
| 12 | 1,227s | 0,852s | 7,74 | 11,12 | 0,64 | 0,93 |
| 16 | 1,108s | 0,654s | 8,57 | 14,49 | 0,54 | 0,91 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 5,957s | 5,934s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 3,213s | 3,007s | 1,85 | 1,97 | 0,93 | 0,99 |
| 4 | 1,891s | 1,548s | 3,15 | 3,83 | 0,79 | 0,96 |
| 6 | 1,366s | 1,066s | 4,36 | 5,57 | 0,73 | 0,93 |
| 9 | 1,047s | 0,734s | 5,69 | 8,08 | 0,63 | 0,90 |
| 12 | 0,999s | 0,565s | 5,96 | 10,50 | 0,50 | 0,88 |
| 16 | 1,023s | 0,439s | 5,82 | 13,52 | 0,36 | 0,84 |

Η σύγχρονη υλοποίηση υποφέρει σημαντικά λόγω του μεγέθους προβλήματος και «καταρρέει» για πλέγμα 4x4 αφού ακόμη και η επιτάχυνση είναι μικρότερη από αυτή του πλέγματος 4x3. Αντίθετα η ασύγχρονη υλοποίηση καταφέρνει να διατηρήσει ικανοποιητικές επιδόσεις ακόμη και στα πλέγματα μεγαλύτερων διαστάσεων.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 18,868s | 18,857s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 9,785s | 9,508s | 1,93 | 1,98 | 0,96 | 0,99 |
| 4 | 5,219s | 4,793s | 3,62 | 3,93 | 0,90 | 0,98 |
| 6 | 3,539s | 3,224s | 5,33 | 5,85 | 0,89 | 0,97 |
| 9 | 2,605s | 2,183s | 7,24 | 8,64 | 0,80 | 0,96 |
| 12 | 2,122s | 1,661s | 8,89 | 11,35 | 0,74 | 0,95 |
| 16 | 1,813s | 1,262s | 10,41 | 14,94 | 0,65 | 0,93 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 12,155s | 12,189s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 6,326s | 6,014s | 1,92 | 2,03 | 0,96 | 1,01 |
| 4 | 3,474s | 3,051s | 3,50 | 4,00 | 0,87 | 1,00 |
| 6 | 2,437s | 2,060s | 4,99 | 5,92 | 0,83 | 0,99 |
| 9 | 1,806s | 1,433s | 6,73 | 8,51 | 0,75 | 0,95 |
| 12 | 1,573s | 1,095s | 7,73 | 11,13 | 0,64 | 0,93 |
| 16 | 1,502s | 0,844s | 8,09 | 14,44 | 0,51 | 0,90 |

Γραμμική επιτάχυνση για πλέγμα 2x2 στην ασύγχρονη υλοποίηση, υπέρ-γραμμική επιτάχυνση για πλέγμα 2x1. Αν και ενδιαφέρουσες παρατηρήσεις εμφανίζονται σε πλέγματα μικρών διαστάσεων.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 38,022s | 38,032s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 19,488s | 19,018s | 1,95 | 2,00 | 0,98 | 1,00 |
| 4 | 10,169s | 9,577s | 3,74 | 3,97 | 0,93 | 0,99 |
| 6 | 7,002s | 6,417s | 5,43 | 5,93 | 0,91 | 0,99 |
| 9 | 4,904s | 4,321s | 7,75 | 8,80 | 0,86 | 0,98 |
| 12 | 3,910s | 3,253s | 9,72 | 11,69 | 0,81 | 0,97 |
| 16 | 3,153s | 2,467s | 12,06 | 15,42 | 0,75 | 0,96 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 23,780s | 23,871s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 12,292s | 12,033s | 1,93 | 1,98 | 0,97 | 0,99 |
| 4 | 6,511s | 6,035s | 3,65 | 3,96 | 0,91 | 0,99 |
| 6 | 4,492s | 4,073s | 5,29 | 5,86 | 0,88 | 0,98 |
| 9 | 3,585s | 3,083s | 6,63 | 7,74 | 0,74 | 0,86 |
| 12 | 2,655s | 2,099s | 8,96 | 11,37 | 0,75 | 0,95 |
| 16 | 2,360s | 1,600s | 10,08 | 14,92 | 0,63 | 0,93 |

Παρατηρείται ασυμμετρία στις μετρήσεις για το πλέγμα 3x3, τόσο στη σύγχρονη όσο και στην ασύγχρονη υλοποίηση. Οι επιδόσεις φαίνεται να «ανακάμπτουν» στα πλέγματα με μεγαλύτερες διαστάσεις. Η συμπεριφορά αυτή επιβεβαιώθηκε με δεύτερη λήψη μετρήσεων και παρατηρήθηκε και στην υβριδική MPI/OpenMP υλοποίηση της οποίας οι μετρήσεις παρουσιάζονται αργότερα. Αποδίδεται μάλλον σε κάποια εσωτερική παράμετρο του MPI σε σχέση με το μέγεθος μηνύματος, στα όρια της οποίας το MPI υποσύστημα αλλάζει συμπεριφορά.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 76,489s | 75,427s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 38,885s | 37,822s | 1,97 | 1,99 | 0,98 | 1,00 |
| 4 | 20,058s | 18,989s | 3,81 | 3,97 | 0,95 | 0,99 |
| 6 | 13,683s | 12,722s | 5,59 | 5,93 | 0,93 | 0,99 |
| 9 | 9,419s | 8,527s | 8,12 | 8,85 | 0,90 | 0,98 |
| 12 | 7,319s | 6,425s | 10,45 | 11,74 | 0,87 | 0,98 |
| 16 | 5,750s | 4,841s | 13,30 | 15,58 | 0,83 | 0,97 |

Πολύ κοντά στη γραμμική επιτάχυνση για την ασύγχρονη υλοποίηση σε όλες τις διαστάσεις πλέγματος.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 48,137s | 48,100s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 24,600s | 24,147s | 1,96 | 1,99 | 0,98 | 1,00 |
| 4 | 12,841s | 12,123s | 3,75 | 3,97 | 0,94 | 0,99 |
| 6 | 8,833s | 8,128s | 5,45 | 5,92 | 0,91 | 0,99 |
| 9 | 6,192s | 5,480s | 7,77 | 8,78 | 0,86 | 0,98 |
| 12 | 4,851s | 4,096s | 9,92 | 11,74 | 0,83 | 0,98 |
| 16 | 4,012s | 3,107s | 12,00 | 15,48 | 0,75 | 0,97 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 151,639s | 151,756s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 76,653s | 76,002s | 1,98 | 2,00 | 0,99 | 1,00 |
| 4 | 39,244s | 38,078s | 3,86 | 3,99 | 0,97 | 1,00 |
| 6 | 26,535s | 25,395s | 5,71 | 5,98 | 0,95 | 1,00 |
| 9 | 18,052s | 16,978s | 8,40 | 8,94 | 0,93 | 0,99 |
| 12 | 13,892s | 12,755s | 10,92 | 11,90 | 0,91 | 0,99 |
| 16 | 10,737s | 9,598s | 14,12 | 15,81 | 0,88 | 0,99 |

Πρακτικά γραμμική επιτάχυνση για την ασύγχρονη υλοποίηση σε όλες τις διαστάσεις πλέγματος. Ακόμη και η σύγχρονη υλοποίηση έχει πολύ ικανοποιητικές επιδόσεις λόγω του μεγάλου μεγέθους προβλήματος. Καθώς αυτό αυξάνεται, ο χρόνος αναμονής για την ολοκλήρωση αποστολής / λήψης μηνυμάτων αρχίζει να γίνεται σχετικά μικρός σε σχέση με το χρόνο της ωφέλιμης επεξεργασίας δεδομένων.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 1 | 96,332s | 96,281s | 1,00 | 1,00 | 1,00 | 1,00 |
| 2 | 48,946s | 48,146s | 1,97 | 2,00 | 0,98 | 1,00 |
| 4 | 25,293s | 24,198s | 3,81 | 3,98 | 0,95 | 0,99 |
| 6 | 17,234s | 16,176s | 5,59 | 5,95 | 0,93 | 0,99 |
| 9 | 11,902s | 10,866s | 8,09 | 8,86 | 0,90 | 0,98 |
| 12 | 9,273s | 8,154s | 10,39 | 11,81 | 0,87 | 0,98 |
| 16 | 7,232s | 6,090s | 13,32 | 15,81 | 0,83 | 0,99 |

### Συγκεντρωτικοί πίνακες Speedup / Efficiency

**RGB 100 Sync. MPI**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 1 | 2 | 4 | 6 | 9 | 12 | 16 |
| 0.25x | Sp. | 1,00 | 1,89 | 3,45 | 4,86 | 6,77 | 7,74 | 8,57 |
| Eff. | 1,00 | 0,95 | 0,86 | 0,81 | 0,75 | 0,64 | 0,54 |
| 0.50x | Sp. | 1,00 | 1,93 | 3,62 | 5,33 | 7,24 | 8,89 | 10,41 |
| Eff. | 1,00 | 0,96 | 0,90 | 0,89 | 0,80 | 0,74 | 0,65 |
| 1.00x | Sp. | 1,00 | 1,95 | 3,74 | 5,43 | 7,75 | 9,72 | 12,06 |
| Eff. | 1,00 | 0,98 | 0,93 | 0,91 | 0,86 | 0,81 | 0,75 |
| 2.00x | Sp. | 1,00 | 1,97 | 3,81 | 5,59 | 8,12 | 10,45 | 13,30 |
| Eff. | 1,00 | 0,98 | 0,95 | 0,93 | 0,90 | 0,87 | 0,83 |
| 4.00x | Sp. | 1,00 | 1,98 | 3,86 | 5,71 | 8,40 | 10,92 | 14,12 |
| Eff. | 1,00 | 0,99 | 0,97 | 0,95 | 0,93 | 0,91 | 0,88 |

**RGB 100 Async. MPI**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 1 | 2 | 4 | 6 | 9 | 12 | 16 |
| 0.25x | Sp. | 1,00 | 1,98 | 3,91 | 5,81 | 8,51 | 11,12 | 14,49 |
| Eff. | 1,00 | 0,99 | 0,98 | 0,97 | 0,95 | 0,93 | 0,91 |
| 0.50x | Sp. | 1,00 | 1,98 | 3,93 | 5,85 | 8,64 | 11,35 | 14,94 |
| Eff. | 1,00 | 0,99 | 0,98 | 0,97 | 0,96 | 0,95 | 0,93 |
| 1.00x | Sp. | 1,00 | 2,00 | 3,97 | 5,93 | 8,80 | 11,69 | 15,42 |
| Eff. | 1,00 | 1,00 | 0,99 | 0,99 | 0,98 | 0,97 | 0,96 |
| 2.00x | Sp. | 1,00 | 1,99 | 3,97 | 5,93 | 8,85 | 11,74 | 15,58 |
| Eff. | 1,00 | 1,00 | 0,99 | 0,99 | 0,98 | 0,98 | 0,97 |
| 4.00x | Sp. | 1,00 | 2,00 | 3,99 | 5,98 | 8,94 | 11,90 | 15,81 |
| Eff. | 1,00 | 1,00 | 1,00 | 1,00 | 0,99 | 0,99 | 0,99 |

**GREY 200 Sync. MPI**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 1 | 2 | 4 | 6 | 9 | 12 | 16 |
| 0.25x | Sp. | 1,00 | 1,85 | 3,15 | 4,36 | 5,69 | 5,96 | 5,82 |
| Eff. | 1,00 | 0,93 | 0,79 | 0,73 | 0,63 | 0,50 | 0,36 |
| 0.50x | Sp. | 1,00 | 1,92 | 3,50 | 4,99 | 6,73 | 7,73 | 8,09 |
| Eff. | 1,00 | 0,96 | 0,87 | 0,83 | 0,75 | 0,64 | 0,51 |
| 1.00x | Sp. | 1,00 | 1,93 | 3,65 | 5,29 | 6,63 | 8,96 | 10,08 |
| Eff. | 1,00 | 0,97 | 0,91 | 0,88 | 0,74 | 0,75 | 0,63 |
| 2.00x | Sp. | 1,00 | 1,96 | 3,75 | 5,45 | 7,77 | 9,92 | 12,00 |
| Eff. | 1,00 | 0,98 | 0,94 | 0,91 | 0,86 | 0,83 | 0,75 |
| 4.00x | Sp. | 1,00 | 1,97 | 3,81 | 5,59 | 8,09 | 10,39 | 13,32 |
| Eff. | 1,00 | 0,98 | 0,95 | 0,93 | 0,90 | 0,87 | 0,83 |

**GREY 200 Async. MPI**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 1 | 2 | 4 | 6 | 9 | 12 | 16 |
| 0.25x | Sp. | 1,00 | 1,97 | 3,83 | 5,57 | 8,08 | 10,50 | 13,52 |
| Eff. | 1,00 | 0,99 | 0,96 | 0,93 | 0,90 | 0,88 | 0,84 |
| 0.50x | Sp. | 1,00 | 2,03 | 4,00 | 5,92 | 8,51 | 11,13 | 14,44 |
| Eff. | 1,00 | 1,01 | 1,00 | 0,99 | 0,95 | 0,93 | 0,90 |
| 1.00x | Sp. | 1,00 | 1,98 | 3,96 | 5,86 | 7,74 | 11,37 | 14,92 |
| Eff. | 1,00 | 0,99 | 0,99 | 0,98 | 0,86 | 0,95 | 0,93 |
| 2.00x | Sp. | 1,00 | 1,99 | 3,97 | 5,92 | 8,78 | 11,74 | 15,48 |
| Eff. | 1,00 | 1,00 | 0,99 | 0,99 | 0,98 | 0,98 | 0,97 |
| 4.00x | Sp. | 1,00 | 2,00 | 3,98 | 5,95 | 8,86 | 11,81 | 15,81 |
| Eff. | 1,00 | 1,00 | 0,99 | 0,99 | 0,98 | 0,98 | 0,99 |

### Μικροδιαφορές: σύγκριση persistent / non-persistent ασύγχρονης MPI επικοινωνίας

**RGB 100**

|  |  |  |
| --- | --- | --- |
| 0.25x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 9,474s | 9,479s |
| 2 | 4,781s | 4,771s |
| 4 | 2,421s | 2,421s |
| 6 | 1,632s | 1,637s |
| 9 | 1,113s | 1,111s |
| 12 | 0,852s | 0,850s |
| 16 | 0,654s | 0,655s |

|  |  |  |
| --- | --- | --- |
| 0.50x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 18,857s | 18,860s |
| 2 | 9,508s | 9,505s |
| 4 | 4,793s | 4,801s |
| 6 | 3,224s | 3,222s |
| 9 | 2,183s | 2,191s |
| 12 | 1,661s | 1,660s |
| 16 | 1,262s | 1,262s |

|  |  |  |
| --- | --- | --- |
| 1.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 38,032s | 37,946s |
| 2 | 19,018s | 19,021s |
| 4 | 9,577s | 9,589s |
| 6 | 6,417s | 6,419s |
| 9 | 4,321s | 4,317s |
| 12 | 3,253s | 3,258s |
| 16 | 2,467s | 2,461s |

|  |  |  |
| --- | --- | --- |
| 2.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 75,427s | 75,421s |
| 2 | 37,822s | 37,832s |
| 4 | 18,989s | 18,944s |
| 6 | 12,722s | 12,705s |
| 9 | 8,527s | 8,530s |
| 12 | 6,425s | 6,416s |
| 16 | 4,841s | 4,838s |

|  |  |  |
| --- | --- | --- |
| 4.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 151,756s | 151,710s |
| 2 | 76,002s | 75,988s |
| 4 | 38,078s | 38,046s |
| 6 | 25,395s | 25,367s |
| 9 | 16,978s | 16,978s |
| 12 | 12,755s | 12,762s |
| 16 | 9,598s | 9,604s |

**GREY 200**

|  |  |  |
| --- | --- | --- |
| 0.25x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 5,934s | 5,945s |
| 2 | 3,007s | 3,016s |
| 4 | 1,548s | 1,549s |
| 6 | 1,066s | 1,064s |
| 9 | 0,734s | 0,733s |
| 12 | 0,565s | 0,565s |
| 16 | 0,439s | 0,437s |

|  |  |  |
| --- | --- | --- |
| 0.50x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 12,189s | 12,117s |
| 2 | 6,014s | 6,006s |
| 4 | 3,051s | 3,050s |
| 6 | 2,060s | 2,064s |
| 9 | 1,433s | 1,431s |
| 12 | 1,095s | 1,094s |
| 16 | 0,844s | 0,837s |

|  |  |  |
| --- | --- | --- |
| 1.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 23,871s | 23,795s |
| 2 | 12,033s | 11,990s |
| 4 | 6,035s | 6,012s |
| 6 | 4,073s | 4,059s |
| 9 | 3,083s | 3,099s |
| 12 | 2,099s | 2,095s |
| 16 | 1,600s | 1,599s |

|  |  |  |
| --- | --- | --- |
| 2.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 48,100s | 48,066s |
| 2 | 24,147s | 24,156s |
| 4 | 12,123s | 12,133s |
| 6 | 8,128s | 8,139s |
| 9 | 5,480s | 5,476s |
| 12 | 4,096s | 4,087s |
| 16 | 3,107s | 3,092s |

|  |  |  |
| --- | --- | --- |
| 4.00x | Async. | |
| Cores | Pers. | Nonp. |
| 1 | 96,281s | 96,373s |
| 2 | 48,146s | 48,186s |
| 4 | 24,198s | 24,193s |
| 6 | 16,176s | 16,164s |
| 9 | 10,866s | 10,865s |
| 12 | 8,154s | 8,168s |
| 16 | 6,090s | 6,101s |

## Υβριδική MPI/OpenMP υλοποίηση

### Περιγραφή μετρήσεων

Στις μετρήσεις υβριδικού προγράμματος MPI/OpenMP χρησιμοποιήσαμε ως χρόνους αναφοράς τους χρόνους ολοκλήρωσης της εργασίας για πλέγμα 1x1 από την MPI υλοποίηση, δηλαδή ουσιαστικά το χρόνο ολοκλήρωσης της εργασίας από έναν πυρήνα. Αυτός είναι ο λόγος που για πλέγμα 1x1 (1 μηχανή - 2 πυρήνες) έχουμε αποδοτικότητα λίγο μικρότερη του 1,00. Θέλαμε με αυτόν τον τρόπο να εισάγουμε στις μετρήσεις και ένα μέτρο της αποδοτικότητας του OpenMP σε σχέση με το διαμοιρασμό της εργασίας στους δύο πυρήνες μιας μηχανής. Στα διαγράμματα και τους πίνακες αναφερόμαστε πάντα σε χρησιμοποιούμενους πυρήνες, για παράδειγμα για πλέγμα υπολογισμού 3x3 έχουμε 18 χρησιμοποιούμενους πυρήνες. Σαν αποτέλεσμα έχουμε μια ένδειξη της κλιμάκωσης του υβριδικού προγράμματος με μέγιστο 32 συνολικά πυρήνες, χρησιμοποιώντας 16 μηχανές σε πλέγμα 4x4. Όπως είδαμε στην παρουσίαση του πηγαίου κώδικα, η υβριδική υλοποίηση έχει δύο εκδοχές τόσο στη σύγχρονη όσο και στην ασύγχρονη περίπτωση. Στα διαγράμματα παρουσιάζεται η «σύνθετη» εκδοχή (δημιουργία των νημάτων μια μόνο φορά εκτός του κυρίου βρόχου εφαρμογής του φίλτρου) που είχε συνολικά ελαφρώς καλύτερες επιδόσεις. Μια σύγκριση επιδόσεων της «σύνθετης» και της «απλής» υλοποίησης παρουσιάζεται μετά τους συγκεντρωτικούς πίνακες Speedup / Efficiency.

### Διαγράμματα - πίνακες μετρήσεων

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 4,835s | 4,855s | 1,96 | 1,95 | 0,98 | 0,98 |
| 4 | 2,646s | 2,451s | 3,59 | 3,87 | 0,90 | 0,97 |
| 8 | 1,595s | 1,260s | 5,95 | 7,52 | 0,74 | 0,94 |
| 12 | 1,177s | 0,866s | 8,07 | 10,94 | 0,67 | 0,91 |
| 18 | 0,891s | 0,595s | 10,65 | 15,92 | 0,59 | 0,88 |
| 24 | 0,823s | 0,470s | 11,53 | 20,16 | 0,48 | 0,84 |
| 32 | 0,816s | 0,369s | 11,63 | 25,67 | 0,36 | 0,80 |

Με τη σύγχρονη υλοποίηση στο μικρότερο μέγεθος προβλήματος έχουμε μικρότερη επιτάχυνση στο 4x4 πλέγμα (32) από ότι στο 4x3 (24), ακόμα και στην RGB περίπτωση.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 3,034s | 3,043s | 1,96 | 1,95 | 0,98 | 0,98 |
| 4 | 1,785s | 1,567s | 3,34 | 3,79 | 0,83 | 0,95 |
| 8 | 1,186s | 0,830s | 5,02 | 7,15 | 0,63 | 0,89 |
| 12 | 0,881s | 0,582s | 6,76 | 10,20 | 0,56 | 0,85 |
| 18 | 0,720s | 0,408s | 8,27 | 14,54 | 0,46 | 0,81 |
| 24 | 0,752s | 0,326s | 7,92 | 18,20 | 0,33 | 0,76 |
| 32 | 0,847s | 0,263s | 7,03 | 22,56 | 0,22 | 0,71 |

Στη GREY περίπτωση με τη σύγχρονη υλοποίηση έχουμε την απολύτως χειρότερη επίδοση σε αποδοτικότητα με το 4x4 πλέγμα και μέγιστο αριθμό πυρήνων (32). Την καλύτερη της επιτάχυνση, η σύγχρονη υλοποίηση την επιτυγχάνει με 18 πυρήνες (3x3 πλέγμα) και αποδοτικότητα μικρότερη του 0,50.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 9,712s | 9,737s | 1,94 | 1,94 | 0,97 | 0,97 |
| 4 | 5,182s | 4,923s | 3,64 | 3,83 | 0,91 | 0,96 |
| 8 | 2,887s | 2,495s | 6,54 | 7,56 | 0,82 | 0,94 |
| 12 | 2,077s | 1,690s | 9,08 | 11,16 | 0,76 | 0,93 |
| 18 | 1,556s | 1,167s | 12,13 | 16,16 | 0,67 | 0,90 |
| 24 | 1,343s | 0,890s | 14,05 | 21,19 | 0,59 | 0,88 |
| 32 | 1,226s | 0,688s | 15,39 | 27,41 | 0,48 | 0,86 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 6,123s | 6,101s | 1,99 | 2,00 | 0,99 | 1,00 |
| 4 | 3,363s | 3,111s | 3,61 | 3,92 | 0,90 | 0,98 |
| 8 | 1,991s | 1,611s | 6,10 | 7,57 | 0,76 | 0,95 |
| 12 | 1,470s | 1,111s | 8,27 | 10,97 | 0,69 | 0,91 |
| 18 | 1,173s | 0,791s | 10,36 | 15,41 | 0,58 | 0,86 |
| 24 | 1,097s | 0,616s | 11,08 | 19,79 | 0,46 | 0,82 |
| 32 | 1,141s | 0,474s | 10,65 | 25,72 | 0,33 | 0,80 |

«Κατάρρευση» για τη σύγχρονη υλοποίηση ακόμη και για το 0.50x μέγεθος προβλήματος στη GREY περίπτωση. Μέγιστη επιτάχυνση επιτυγχάνει με το πλέγμα 3x4 και 24 πυρήνες και την αποδοτικότητα κάτω του 0,50. Απεναντίας, η ασύγχρονη υλοποίηση καταφέρνει να διατηρήσει σχετικά καλές επιδόσεις με 0,80 αποδοτικότητα στο 4x4 πλέγμα.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 19,332s | 19,414s | 1,97 | 1,96 | 0,98 | 0,98 |
| 4 | 10,032s | 9,695s | 3,79 | 3,92 | 0,95 | 0,98 |
| 8 | 5,473s | 4,912s | 6,95 | 7,74 | 0,87 | 0,97 |
| 12 | 3,870s | 3,296s | 9,82 | 11,54 | 0,82 | 0,96 |
| 18 | 2,779s | 2,262s | 13,68 | 16,81 | 0,76 | 0,93 |
| 24 | 2,344s | 1,711s | 16,22 | 22,23 | 0,68 | 0,93 |
| 32 | 1,983s | 1,304s | 19,17 | 29,17 | 0,60 | 0,91 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 12,274s | 12,295s | 1,94 | 1,94 | 0,97 | 0,97 |
| 4 | 6,454s | 6,171s | 3,68 | 3,87 | 0,92 | 0,97 |
| 8 | 3,593s | 3,131s | 6,62 | 7,62 | 0,83 | 0,95 |
| 12 | 2,546s | 2,132s | 9,34 | 11,20 | 0,78 | 0,93 |
| 18 | 2,112s | 2,130s | 11,26 | 11,21 | 0,63 | 0,62 |
| 24 | 1,712s | 1,680s | 13,89 | 14,21 | 0,58 | 0,59 |
| 32 | 1,660s | 1,134s | 14,33 | 21,05 | 0,45 | 0,66 |

Η «ανωμαλία» του πλέγματος 3x3 (18 πυρήνες) επηρεάζει και το υβριδικό πρόγραμμα όπως προηγουμένως το MPI, με τη διαφορά ότι εδώ δεν έχουμε το ίδιο καλή ανάκαμψη για μεγαλύτερες διαστάσεις πλέγματος. Θυμίζουμε ότι η υλοποίηση που παρουσιάζεται είναι η «σύνθετη» με τη δημιουργία νημάτων μια φορά εκτός του κυρίου βρόχου εφαρμογής του φίλτρου. Ενδιαφέρον είναι ότι η «απλή» υλοποίηση (που δεν εμφανίζεται εδώ) παρουσίασε και αυτή μεν την ασυμμετρία στο πλέγμα διάστασης 3x3, είχε όμως δε πολύ καλύτερη «ανάκαμψη», παρουσιάζοντας μια εικόνα πολύ παρόμοια με αυτή της καθαρής MPI υλοποίησης που είδαμε προηγουμένως. (Η διαφορά αυτή της «σύνθετης» με την «απλή» υβριδική υλοποίηση στο μέγεθος προβλήματος 1.00x σημειώνεται στο τέλος της παρούσας ενότητας, μετά τους συγκεντρωτικούς πίνακες Speedup / Efficiency.)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 38,885s | 38,761s | 1,97 | 1,95 | 0,98 | 0,97 |
| 4 | 20,014s | 19,555s | 3,82 | 3,86 | 0,96 | 0,96 |
| 8 | 10,639s | 9,828s | 7,19 | 7,67 | 0,90 | 0,96 |
| 12 | 7,387s | 6,616s | 10,35 | 11,40 | 0,86 | 0,95 |
| 18 | 5,379s | 4,474s | 14,22 | 16,86 | 0,79 | 0,94 |
| 24 | 4,154s | 3,349s | 18,41 | 22,52 | 0,77 | 0,94 |
| 32 | 3,432s | 2,537s | 22,29 | 29,73 | 0,70 | 0,93 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 24,366s | 24,450s | 1,98 | 1,97 | 0,99 | 0,98 |
| 4 | 12,892s | 12,392s | 3,73 | 3,88 | 0,93 | 0,97 |
| 8 | 6,989s | 6,242s | 6,89 | 7,71 | 0,86 | 0,96 |
| 12 | 4,799s | 4,263s | 10,03 | 11,28 | 0,84 | 0,94 |
| 18 | 3,428s | 2,873s | 14,04 | 16,74 | 0,78 | 0,93 |
| 24 | 2,869s | 2,171s | 16,78 | 22,16 | 0,70 | 0,92 |
| 32 | 2,547s | 1,665s | 18,90 | 28,89 | 0,59 | 0,90 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 77,177s | 77,041s | 1,96 | 1,97 | 0,98 | 0,98 |
| 4 | 39,208s | 38,634s | 3,87 | 3,93 | 0,97 | 0,98 |
| 8 | 20,653s | 19,398s | 7,34 | 7,82 | 0,92 | 0,98 |
| 12 | 14,055s | 12,964s | 10,79 | 11,71 | 0,90 | 0,98 |
| 18 | 9,826s | 8,709s | 15,43 | 17,43 | 0,86 | 0,97 |
| 24 | 7,693s | 6,567s | 19,71 | 23,11 | 0,82 | 0,96 |
| 32 | 6,115s | 4,952s | 24,80 | 30,65 | 0,77 | 0,96 |

Οι απολύτως καλύτερες επιδόσεις για το υβριδικό πρόγραμμα είναι επιτάχυνση 30,65 με την ασύγχρονη υλοποίηση στο 4x4 (32) πλέγμα. Η αποδοτικότητα διατηρείται σε εξαιρετικά επίπεδα (0,96) ενώ έχει «ξεκινήσει» από το 0,98 με το 1x1 (2) πλέγμα.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x | Time | | Speedup | | Efficiency | |
| Cores | Sync | Async | Sync | Async | Sync | Async |
| 2 | 49,087s | 49,206s | 1,96 | 1,96 | 0,98 | 0,98 |
| 4 | 25,095s | 24,513s | 3,84 | 3,93 | 0,96 | 0,98 |
| 8 | 13,208s | 12,419s | 7,29 | 7,75 | 0,91 | 0,97 |
| 12 | 9,112s | 8,299s | 10,57 | 11,60 | 0,88 | 0,97 |
| 18 | 6,458s | 5,650s | 14,92 | 17,04 | 0,83 | 0,95 |
| 24 | 5,257s | 4,279s | 18,32 | 22,50 | 0,76 | 0,94 |
| 32 | 4,271s | 3,248s | 22,55 | 29,64 | 0,70 | 0,93 |

### Συγκεντρωτικοί πίνακες Speedup / Efficiency

**RGB 100 Sync. MPI/OMP**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 2 | 4 | 8 | 12 | 18 | 24 | 32 |
| 0.25x | Sp. | 1,96 | 3,59 | 5,95 | 8,07 | 10,65 | 11,53 | 11,63 |
| Eff. | 0,98 | 0,90 | 0,74 | 0,67 | 0,59 | 0,48 | 0,36 |
| 0.50x | Sp. | 1,94 | 3,64 | 6,54 | 9,08 | 12,13 | 14,05 | 15,39 |
| Eff. | 0,97 | 0,91 | 0,82 | 0,76 | 0,67 | 0,59 | 0,48 |
| 1.00x | Sp. | 1,97 | 3,79 | 6,95 | 9,82 | 13,68 | 16,22 | 19,17 |
| Eff. | 0,98 | 0,95 | 0,87 | 0,82 | 0,76 | 0,68 | 0,60 |
| 2.00x | Sp. | 1,97 | 3,82 | 7,19 | 10,35 | 14,22 | 18,41 | 22,29 |
| Eff. | 0,98 | 0,96 | 0,90 | 0,86 | 0,79 | 0,77 | 0,70 |
| 4.00x | Sp. | 1,96 | 3,87 | 7,34 | 10,79 | 15,43 | 19,71 | 24,80 |
| Eff. | 0,98 | 0,97 | 0,92 | 0,90 | 0,86 | 0,82 | 0,77 |

**RGB 100 Async. MPI/OMP**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 2 | 4 | 8 | 12 | 18 | 24 | 32 |
| 0.25x | Sp. | 1,95 | 3,87 | 7,52 | 10,94 | 15,92 | 20,16 | 25,67 |
| Eff. | 0,98 | 0,97 | 0,94 | 0,91 | 0,88 | 0,84 | 0,80 |
| 0.50x | Sp. | 1,94 | 3,83 | 7,56 | 11,16 | 16,16 | 21,19 | 27,41 |
| Eff. | 0,97 | 0,96 | 0,94 | 0,93 | 0,90 | 0,88 | 0,86 |
| 1.00x | Sp. | 1,96 | 3,92 | 7,74 | 11,54 | 16,81 | 22,23 | 29,17 |
| Eff. | 0,98 | 0,98 | 0,97 | 0,96 | 0,93 | 0,93 | 0,91 |
| 2.00x | Sp. | 1,95 | 3,86 | 7,67 | 11,40 | 16,86 | 22,52 | 29,73 |
| Eff. | 0,97 | 0,96 | 0,96 | 0,95 | 0,94 | 0,94 | 0,93 |
| 4.00x | Sp. | 1,97 | 3,93 | 7,82 | 11,71 | 17,43 | 23,11 | 30,65 |
| Eff. | 0,98 | 0,98 | 0,98 | 0,98 | 0,97 | 0,96 | 0,96 |

**GREY 200 Sync. MPI/OMP**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 2 | 4 | 8 | 12 | 18 | 24 | 32 |
| 0.25x | Sp. | 1,96 | 3,34 | 5,02 | 6,76 | 8,27 | 7,92 | 7,03 |
| Eff. | 0,98 | 0,83 | 0,63 | 0,56 | 0,46 | 0,33 | 0,22 |
| 0.50x | Sp. | 1,99 | 3,61 | 6,10 | 8,27 | 10,36 | 11,08 | 10,65 |
| Eff. | 0,99 | 0,90 | 0,76 | 0,69 | 0,58 | 0,46 | 0,33 |
| 1.00x | Sp. | 1,94 | 3,68 | 6,62 | 9,34 | 11,26 | 13,89 | 14,33 |
| Eff. | 0,97 | 0,92 | 0,83 | 0,78 | 0,63 | 0,58 | 0,45 |
| 2.00x | Sp. | 1,98 | 3,73 | 6,89 | 10,03 | 14,04 | 16,78 | 18,90 |
| Eff. | 0,99 | 0,93 | 0,86 | 0,84 | 0,78 | 0,70 | 0,59 |
| 4.00x | Sp. | 1,96 | 3,84 | 7,29 | 10,57 | 14,92 | 18,32 | 22,55 |
| Eff. | 0,98 | 0,96 | 0,91 | 0,88 | 0,83 | 0,76 | 0,70 |

**GREY 200 Async. MPI/OMP**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Cores | | 2 | 4 | 8 | 12 | 18 | 24 | 32 |
| 0.25x | Sp. | 1,95 | 3,79 | 7,15 | 10,20 | 14,54 | 18,20 | 22,56 |
| Eff. | 0,98 | 0,95 | 0,89 | 0,85 | 0,81 | 0,76 | 0,71 |
| 0.50x | Sp. | 2,00 | 3,92 | 7,57 | 10,97 | 15,41 | 19,79 | 25,72 |
| Eff. | 1,00 | 0,98 | 0,95 | 0,91 | 0,86 | 0,82 | 0,80 |
| 1.00x | Sp. | 1,94 | 3,87 | 7,62 | 11,20 | 11,21 | 14,21 | 21,05 |
| Eff. | 0,97 | 0,97 | 0,95 | 0,93 | 0,62 | 0,59 | 0,66 |
| 2.00x | Sp. | 1,97 | 3,88 | 7,71 | 11,28 | 16,74 | 22,16 | 28,89 |
| Eff. | 0,98 | 0,97 | 0,96 | 0,94 | 0,93 | 0,92 | 0,90 |
| 4.00x | Sp. | 1,96 | 3,93 | 7,75 | 11,60 | 17,04 | 22,50 | 29,64 |
| Eff. | 0,98 | 0,98 | 0,97 | 0,97 | 0,95 | 0,94 | 0,93 |

### Μικροδιαφορές: σύγκριση μεταξύ της «σύνθετης» και της «απλής» MPI/OpenMP υλοποίησης

**RGB 100**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 4,835s | 4,866s |  | 4,855s | 4,850s |
| 4 |  | 2,646s | 2,688s |  | 2,451s | 2,451s |
| 8 |  | 1,595s | 1,590s |  | 1,260s | 1,268s |
| 12 |  | 1,177s | 1,183s |  | 0,866s | 0,868s |
| 18 |  | 0,891s | 0,910s |  | 0,595s | 0,606s |
| 24 |  | 0,823s | 0,844s |  | 0,470s | 0,471s |
| 32 |  | 0,816s | 0,817s |  | 0,369s | 0,371s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 9,712s | 9,738s |  | 9,737s | 9,761s |
| 4 |  | 5,182s | 5,242s |  | 4,923s | 4,943s |
| 8 |  | 2,887s | 2,928s |  | 2,495s | 2,491s |
| 12 |  | 2,077s | 2,126s |  | 1,690s | 1,695s |
| 18 |  | 1,556s | 1,582s |  | 1,167s | 1,169s |
| 24 |  | 1,343s | 1,356s |  | 0,890s | 0,888s |
| 32 |  | 1,226s | 1,244s |  | 0,688s | 0,685s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 19,332s | 19,359s |  | 19,414s | 19,430s |
| 4 |  | 10,032s | 10,303s |  | 9,695s | 9,716s |
| 8 |  | 5,473s | 5,562s |  | 4,912s | 4,929s |
| 12 |  | 3,870s | 3,934s |  | 3,296s | 3,297s |
| 18 |  | 2,779s | 2,903s |  | 2,262s | 2,252s |
| 24 |  | 2,344s | 2,451s |  | 1,711s | 1,714s |
| 32 |  | 1,983s | 2,121s |  | 1,304s | 1,308s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 38,885s | 38,757s |  | 38,761s | 38,915s |
| 4 |  | 20,014s | 20,117s |  | 19,555s | 19,602s |
| 8 |  | 10,639s | 10,663s |  | 9,828s | 9,829s |
| 12 |  | 7,387s | 7,625s |  | 6,616s | 6,611s |
| 18 |  | 5,379s | 5,266s |  | 4,474s | 4,449s |
| 24 |  | 4,154s | 4,182s |  | 3,349s | 3,351s |
| 32 |  | 3,432s | 3,439s |  | 2,537s | 2,537s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 77,177s | 77,139s |  | 77,041s | 77,372s |
| 4 |  | 39,208s | 39,691s |  | 38,634s | 38,617s |
| 8 |  | 20,653s | 20,733s |  | 19,398s | 19,449s |
| 12 |  | 14,055s | 14,191s |  | 12,964s | 12,995s |
| 18 |  | 9,826s | 9,950s |  | 8,709s | 8,733s |
| 24 |  | 7,693s | 7,806s |  | 6,567s | 6,565s |
| 32 |  | 6,115s | 6,196s |  | 4,952s | 4,958s |

**GREY 200**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.25x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 3,034s | 3,038s |  | 3,043s | 3,043s |
| 4 |  | 1,785s | 1,773s |  | 1,567s | 1,568s |
| 8 |  | 1,186s | 1,186s |  | 0,830s | 0,829s |
| 12 |  | 0,881s | 0,889s |  | 0,582s | 0,583s |
| 18 |  | 0,720s | 0,734s |  | 0,408s | 0,425s |
| 24 |  | 0,752s | 0,761s |  | 0,326s | 0,329s |
| 32 |  | 0,847s | 0,847s |  | 0,263s | 0,265s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0.50x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 6,123s | 6,106s |  | 6,101s | 6,130s |
| 4 |  | 3,363s | 3,344s |  | 3,111s | 3,107s |
| 8 |  | 1,991s | 1,991s |  | 1,611s | 1,603s |
| 12 |  | 1,470s | 1,475s |  | 1,111s | 1,110s |
| 18 |  | 1,173s | 1,182s |  | 0,791s | 0,794s |
| 24 |  | 1,097s | 1,094s |  | 0,616s | 0,613s |
| 32 |  | 1,141s | 1,147s |  | 0,474s | 0,473s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 12,274s | 12,245s |  | 12,295s | 12,278s |
| 4 |  | 6,454s | 6,492s |  | 6,171s | 6,182s |
| 8 |  | 3,593s | 3,626s |  | 3,131s | 3,128s |
| 12 |  | 2,546s | 2,598s |  | 2,132s | 2,129s |
| 18 |  | 2,112s | 2,181s |  | **2,130s** | **1,690s** |
| 24 |  | 1,712s | 1,727s |  | **1,680s** | **1,136s** |
| 32 |  | 1,660s | 1,647s |  | **1,134s** | **0,882s** |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 24,366s | 24,437s |  | 24,450s | 24,398s |
| 4 |  | 12,892s | 12,903s |  | 12,392s | 12,352s |
| 8 |  | 6,989s | 6,954s |  | 6,242s | 6,252s |
| 12 |  | 4,799s | 4,869s |  | 4,263s | 4,231s |
| 18 |  | 3,428s | 3,517s |  | 2,873s | 2,882s |
| 24 |  | 2,869s | 2,880s |  | 2,171s | 2,175s |
| 32 |  | 2,547s | 2,562s |  | 1,665s | 1,668s |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4.00x |  | Sync. | |  | Async. | |
|  |  | Comp. | Simp. |  | Comp. | Simp. |
| 2 |  | 49,087s | 49,162s |  | 49,206s | 49,056s |
| 4 |  | 25,095s | 25,368s |  | 24,513s | 24,580s |
| 8 |  | 13,208s | 13,414s |  | 12,419s | 12,480s |
| 12 |  | 9,112s | 9,215s |  | 8,299s | 8,339s |
| 18 |  | 6,458s | 6,546s |  | 5,650s | 5,659s |
| 24 |  | 5,257s | 5,347s |  | 4,279s | 4,273s |
| 32 |  | 4,271s | 4,344s |  | 3,248s | 3,218s |

# CUDA

## Πηγαίος κώδικας

### Η δέσμευση και αποδέσμευση μνήμης

Έγινε προσπάθεια να διατηρηθεί το σχήμα δέσμευσης μνήμης που παρουσιάστηκε στο SPMD πρόγραμμα, έτσι ώστε να χρησιμοποιηθούν ήδη έτοιμα και δοκιμασμένα τμήματα κώδικα επεξεργασίας δεδομένων με ελάχιστες αλλαγές. Παρακάτω παρουσιάζονται οι συναρτήσεις δέσμευσης και αποδέσμευσης μνήμης:

|  |
| --- |
| 2 \* File: 2d\_malloc\_cuda.cpp  ...  32 extern "C" bool alloc\_uchar\_array\_cuda(unsigned char \*\*\*array\_d, unsigned char \*\*p\_d,  int rows, int columns, int channels)  33 {  34 unsigned char \*p;  35 cudaMalloc((void \*\*) &p, rows \* columns \* channels \* sizeof (unsigned char));  36 if (p == NULL)  37 {  38 fprintf(stderr, "cudaMalloc(): could not allocate device memory\n");  39 return false;  40 }  41  42 \*p\_d = p;  43  44 unsigned char \*\*array\_p;  45 cudaMalloc((void \*\*) &array\_p, rows \* sizeof (unsigned char \*));  46 if (array\_p == NULL)  47 {  48 fprintf(stderr, "cudaMalloc(): could not allocate device memory\n");  49 cudaFree(p);  50 return false;  51 }  52  53 dim3 dimBl(1);  54 dim3 dimGr(1);  55 k\_assign\_uchar\_ptrs<<<dimGr, dimBl>>>(array\_p, p, rows, columns, channels);  56  57 \*array\_d = array\_p;  58  59 return true;  60 }  ...  92 extern "C" void dealloc\_uchar\_array\_cuda(unsigned char \*\*\*array\_d, unsigned char \*\*p\_d)  93 {  94 cudaFree(p\_d);  95 p\_d = NULL;  96 cudaFree(\*array\_d);  97 \*array\_d = NULL;  98 }  ... |

Η συνάρτηση alloc\_uchar\_array\_cuda() κάνει χρήση της κλήσης cudaMalloc() για δέσμευση της συνεχόμενης περιοχής μνήμης στο χώρο διευθύνσεων της κάρτας γραφικών. Για να αρχικοποιηθούν σωστά οι δείκτες στον πίνακα δεικτών \*array\_d είναι αναγκαία η κλήση ενός kernel ο οποίος έχει τη δυνατότητα πρόσβασης στη μνήμη της κάρτας γραφικών:

|  |
| --- |
| 2 \* File: 2d\_malloc\_cuda.cpp  ...  18 \_\_global\_\_ void k\_assign\_uchar\_ptrs(unsigned char \*\*array, unsigned char \*p,  int rows, int columns, int channels)  19 {  20 int i;  21 for (i = 0; i < rows; i++)  22 array[i] = &(p[i \* columns \* channels]);  23 }  ... |

Πέρα από τον πίνακα \*array\_d ο οποίος βρίσκεται στην περιοχή μνήμης της κάρτας γραφικών και περιέχει διευθύνσεις στην ίδια περιοχή, η συνάρτηση alloc\_uchar\_array\_cuda() επιστρέφει και ένα δείκτη στην αρχή της συνεχόμενης περιοχής μνήμης έτσι ώστε το κυρίως πρόγραμμα να μπορεί να αντιγράψει τα δεδομένα της εικόνας στη συσκευή, να κάνει την εναλλαγή των δυο buffers που χρησιμοποιούνται κατά την εκτέλεση του βρόχου εφαρμογής του φίλτρου και να αποδεσμεύσει τη μνήμη στο τέλος της επεξεργασίας. Μια προσπάθεια να προσδιοριστεί η διεύθυνση της συνεχόμενης περιοχής μνήμης με μια παράσταση της μορφής &(array[0][0]) από το κυρίως πρόγραμμα αποτυγχάνει αφού αυτό δεν έχει πρόσβαση στην περιοχή μνήμης της κάρτας γραφικών.

### Οι παράμετροι της συσκευής

Οι παράμετροι της κάρτας γραφικών που χρησιμοποιήθηκε υπολογίζονται μια φορά από το κυρίως πρόγραμμα και περνιούνται στη συνάρτηση εφαρμογής του εσωτερικού μέρους του φίλτρου σε κάθε επανάληψη του βρόχου. Χρησιμοποιούμε block των 64 threads και ένα τετραγωνικό grid από blocks με επαρκείς διαστάσεις για να έχουμε τα απαιτούμενα νήματα για τις εκάστοτε διαστάσεις της εικόνας. Μεγαλύτερα μεγέθη block παρουσίαζαν αυξανόμενα χειρότερες επιδόσεις, αν και όχι κατά πολύ. Για το grid, στην περίπτωση που επιλέγαμε να χρησιμοποιήσουμε μονοδιάστατο, είχαμε υπέρβαση του ορίου 65535 για τις διαστάσεις του στα μεγάλα μεγέθη εικόνας, περιορισμός της κάρτας γραφικών που χρησιμοποιήθηκε η οποία υποστηρίζει (1.1 compute capability).

|  |
| --- |
| 2 \* File: main\_cuda.c  ...  98 /\* nVidia G94 supports 8 resident blocks per SMP, 768 resident threads per SMP. \*/  99  100 unsigned int block\_size = 64; // maximum 512 threads per block for nVidia G94  101 printf("Block size: %u\n", block\_size);  102  103 /\* nVidia G94 supports 2-dimensional grids with a maximum of 65535 for x,y dimension. \*/  104  105 unsigned int grid\_dim = HEIGHT \* WIDTH / block\_size;  106 double sqr = sqrt(grid\_dim);  107 grid\_dim = sqr;  108 grid\_dim++;  109 printf("Grid: %ux%u\n", grid\_dim, grid\_dim);  ... |

### Ο kernel για το εσωτερικό μέρος του φίλτρου

Το κύριο μέρος της εργασίας είναι η εφαρμογή του εσωτερικού μέρους του φίλτρου. Κάθε νήμα αναλαμβάνει να διαβάσει τις αναγκαίες περιοχές μνήμης βάσει του φίλτρου που χρησιμοποιείται και να ενημερώσει την πληροφορία για όλα τα κανάλια ενός pixel της εικόνας. Περαιτέρω διαμοιρασμός ενός νήματος ανά κανάλι και όχι ανά pixel δεν έδωσε καλύτερη απόδοση. Ο kernel του εσωτερικού μέρους του φίλτρου φαίνεται παρακάτω.

|  |
| --- |
| 2 \* File: filter\_cuda.cu  ...  107 \_\_global\_\_ void k\_apply\_filter\_cuda(float (\*\*output\_image\_d)[CHANNELS],  float (\*\*input\_image\_d)[CHANNELS], float (\*\*filter\_d)[1])  108 {  109 unsigned int threadsPerBlock = blockDim.x \* blockDim.y;  110 unsigned int blockId = blockIdx.y \* gridDim.x + blockIdx.x;  111 unsigned int threadId = threadIdx.y \* blockDim.x + threadIdx.x;  112 unsigned int globalId = blockId \* threadsPerBlock + threadId;  113  114 unsigned int i, j, c;  115 int p, q;  116  117 i = globalId / WIDTH;  118 j = globalId % WIDTH;  119  120 if (i > HEIGHT - 1)  121 return;  122  123 i += B;  124 j += B;  125  126 for (c = 0; c < CHANNELS; c++)  127 {  128 float value = 0.0f;  129  130 for (p = -B; p <= B; p++)  131 for (q = -B; q <= B; q++)  132 value += input\_image\_d[i - p][j - q][c] \* filter\_d[p + B][q + B][0];  133  134 output\_image\_d[i][j][c] = value;  135 }  136 }  ... |

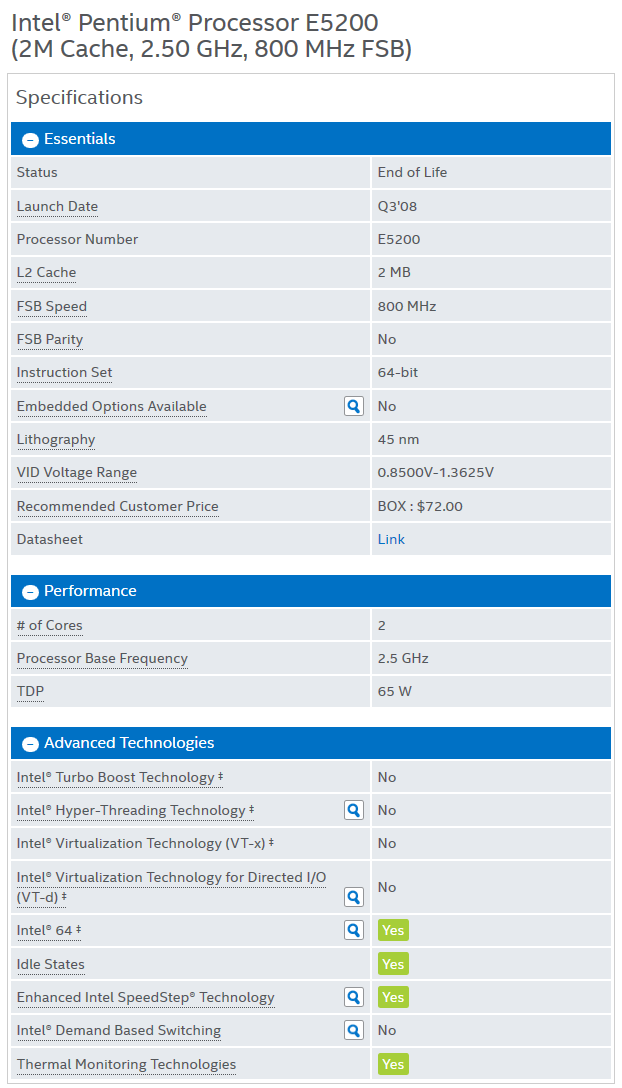
Μετά τον υπολογισμό ενός global id για το νήμα που εκτελείται, έχουμε βάσει αυτού προσδιορισμό του pixel που θα ενημερωθεί. Αφού γίνει έλεγχος για να τερματίσουν τα νήματα που δε θα χρειαστούν τελικά, ακολουθεί ο κυρίως βρόχος ο οποίος είναι ίδιος με αυτόν του MPI/OpenMP προγράμματος, περιορισμένος σε ένα μόνο pixel. Το φίλτρο περνιέται ως παράμετρος στη συνάρτηση και βρίσκεται εντός της περιοχής διευθύνσεων μνήμης της κάρτας γραφικών. Η αντιγραφή του φίλτρου από την κύρια μνήμη στη μνήμη της συσκευής έχει γίνει από το κυρίως πρόγραμμα πριν την είσοδο στο βρόχο εφαρμογής του φίλτρου.

## Hardware

Η υλοποίηση σε CUDA και οι μετρήσεις έγιναν σε σύστημα του οποίου οι προδιαγραφές από πλευράς επεξεργαστή και κάρτας γραφικών παρουσιάζονται παρακάτω:

### Επεξεργαστής





### Κάρτα γραφικών

|  |  |
| --- | --- |
| **D:\Documents and Settings\John\Desktop\NVidia_GeForce_9_Series.jpg** | **D:\Documents and Settings\John\My Documents\Data\University\Μαθήματα\Τρέχοντα\Παράλληλα Συστήματα\Εργασία\τεκμηρίωση\33_gainward_9600gt_pcx_hdmi.jpg** |
| **NVIDIA® GeForce® 9600 GT** |

* Launch date: February 21, 2008
* 65 nm **G94** GPU.
* 64 stream processors.
* 16 raster operation (ROP) units, 32 texture address (TA)/texture filter (TF) units.
* 20.8 billion texels/s fill rate.
* 650 MHz core clock, with a 1625 MHz unified shader clock.
* 1008 MHz memory (2016 MHz datarate), 256-bit interface for 64.5GB/s of bandwidth. (57.6 GB/s for 1800MHz configuration).
* 512 MB of GDDR3 or DDR2 memory.
* 505M transistor count
* DirectX 10.0, Shader Model 4.0, OpenGL 3.3, and PCI-Express 2.0.[[8]](http://en.wikipedia.org/wiki/GeForce_9_series#cite_note-8)
* Supports second-generation [PureVideo HD](http://en.wikipedia.org/wiki/Nvidia_PureVideo) technology with partial VC1 decoding.
* Is compatible with [HDCP](http://en.wikipedia.org/wiki/High-bandwidth_Digital_Content_Protection), but the implementation will depend on the manufacturer.
* Supports [CUDA](http://en.wikipedia.org/wiki/CUDA) (1.1 compute capability) and the Quantum Effects physics processing engine.

Το έτος κυκλοφορίας και για τις δύο συσκευές είναι το 2008 οπότε η σύγκριση των επιδόσεων προγραμμάτων που τις αξιοποιούν έχει κάποιο ενδιαφέρον.

## Μετρήσεις

Έγινε σύγκριση των επιδόσεων της υλοποίησης CUDA με το πρόγραμμα αναφοράς με χρήση OpenMP (serial\_omp) για αξιοποίηση και των δυο πυρήνων του κεντρικού επεξεργαστή. Οι μετρήσεις αντικατοπτρίζουν τις επιδόσεις των συγκεκριμένων υλοποιήσεων και μόνο εν μέρει την πραγματική υπολογιστική ισχύ των συσκευών. Χρησιμοποιήθηκαν όλες οι διαστάσεις προβλήματος, ενώ για την RGB περίπτωση χρησιμοποιήθηκαν 100 επαναλήψεις και για την GREY 200, όπως και στις μετρήσεις του SPMD προγράμματος. Για την CUDA υλοποίηση καταγράφηκαν και οι χρόνοι αντιγραφής από την κεντρική μνήμη στη μνήμη της συσκευής και αντίστροφα.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RGB |  | 9600GT | |  | E5200 |  |  |
|  |  | Filter | Memcpy |  | Filter |  | Ratio |
| 0.25x |  | 14,424s | 0,017s |  | 3,530s |  | 4,09 |
| 0.50x |  | 23,056s | 0,034s |  | 7,000s |  | 3,29 |
| 1.00x |  | 46,027s | 0,068s |  | 14,000s |  | 3,29 |
| 2.00x |  | 89,589s | 0,135s |  | 27,850s |  | 3,22 |
| 4.00x |  | - | - |  | 55,990s |  | - |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| GREY |  | 9600GT | |  | E5200 |  |  |
|  |  | Filter | Memcpy |  | Filter |  | Ratio |
| 0.25x |  | 7,859s | 0,006s |  | 2,030s |  | 3,87 |
| 0.50x |  | 15,274s | 0,012s |  | 4,010s |  | 3,81 |
| 1.00x |  | 29,911s | 0,023s |  | 8,010s |  | 3,73 |
| 2.00x |  | 61,957s | 0,045s |  | 16,010s |  | 3,87 |
| 4.00x |  | 120,925s | 0,090s |  | 32,430s |  | 3,73 |

Στο μέγεθος προβλήματος 4.00x για την RGB περίπτωση δεν ήταν δυνατή η λήψη μετρήσεων επειδή είχαμε αποτυχία της κλήσης cudaMalloc() για τη δέσμευση της απαραίτητης μνήμης. Συνολικά, η OpenMP υλοποίηση είναι τρεις με τέσσερις φορές γρηγορότερη από την υλοποίηση με CUDA. Παρ’ όλα αυτά, ο τρόπος πρόσβασης στη μνήμη στην CUDA υλοποίηση δεν είναι βέλτιστος. Για κάθε pixel της εικόνας εξόδου έχουμε ανάγνωση από όσα γειτονικά του pixels απαιτούνται από το φίλτρο που χρησιμοποιείται. Για παράδειγμα κάθε pixel της εικόνας εισόδου διαβάζεται 9 φορές συνολικά για φίλτρο 3x3. Μια εναλλακτική θα ήταν κάθε pixel να διαβάζεται μια μόνο φορά, και να αθροίζεται η συνεισφορά του στα pixel της εικόνας εξόδου τα οποία επηρεάζει. Με αυτό τον τρόπο θα είχαμε καλύτερες επιδόσεις, όμως για να γίνει δυνατό να γράψουν πολλαπλά νήματα σε διαμοιραζόμενες περιοχές μνήμης απαιτούνται ατομικές πράξεις πρόσθεσης, οι οποίες γενικά δεν υποστηρίζονται σε συσκευές με 1.1 compute capability.