Image Convolution Kernel with Mpi-OpenMP

Author Name

Panagiotis Drakatos

Department of Informatics and Telecommunications
University of Athens

Brief Summary

Συνέλιξη μιας δισδιάστατης εικόνας με φίλτρο συνέλιξης.
Χρησιμοποιείται για την εφαρμογή
εφέ,για αντιταύτιση και εξομάλυνση εικόνων

02/05/2018
Date

Περιεχόμενα

Κατάλογος πινάκων

Κατάλογος σχημάτων

1. Εισαγωγή

Σκοπός αυτής της διδακτικής ενότητας είναι να αναλύσουμε την έννοια της συνέλιξης, να δούμε πως προκύπτει η μαθηματική ακολουθία, καθώς και να κατανοήσουμε πως μπορεί να εφαρμοστεί με την χρήση φίλτρου για την εξομάλυνση εικόνων. Επιπροσθέτως στις επόμενες υποενότητες αναλύεται η μεθοδολογία για το πως πραγματοποιείται ο πολλαπλασιασμός των δισδιάστατων πινάκων για τον υπολογισμό της συνέλιξης ενός σειριακού προγράμματος, έτσι ώστε να κατανοήσουμε τα βήματα του βασικού αλγορίθμου πριν φτάσουμε στην παραλληλοποίησή του.

A. Μαθηματική Αναπαράσταση

Μια πολύ χρήσιμη πράξη στα Γραφικά και στην Επεξεργασία Εικόνας είναι η συνέλιξη μιας δισδιάστατης εικόνας με ένα φίλτρο συνέλιξης, που εφαρμόζεται σε δύο συναρτήσεις. Η συνέλιξη είναι μια απλή μαθηματική λειτουργία που χρησιμοποιείται από πολλούς κοινούς διαχειριστές εικόνων και χρησιμοποιείται για την αντιταύτιση και εξομάλυνση εικόνων.

Το θεώρημα της συνέλιξης καθορίζει ότι η εν λόγω μέθοδος μπορεί να εφαρμοστεί με τον ίδιο τρόπο με τον πολλαπλασιασμό ανά συχνότητα στη περιοχή συχνοτήτων. Γεγονός που υποδηλώνει ότι μπορεί να εφαρμοστεί ο μετασχηματισμός Fourier και πιο συγκεκριμένα ο διακριτός μετασχηματισμός Fourier (Για τον διακριτό μετασχηματισμό Fourier συνήθως χρησιμοποιούμε την συντομογραφία DFT (Discrete Fourier Transform)).

Σύμφωνα με τον διακριτό μετασχηματισμό Fourier μας παρέχεται η δυνατότητα μετάβασης από το πεδίο χώρου μίας εικόνας (spatial domain) στο αντίστοιχο πεδίο συχνοτήτων της (frequency domain) αναλύοντας την εικόνα ως άθροισμα μιγαδικών εκθετικών εικόνων. Δηλαδή μπορούμε να θεωρήσουμε ότι κάθε συνεχές μονοδιάστατο σήμα μπορεί να αναπαρασταθεί ως άθροισμα ημιτονικών σημάτων. Ως εκ τούτου και η εικόνα μπορεί να αναπαρασταθεί ως άθροισμα ημιτονικών εικόνων. Αυτή η δυνατότητα είναι πολύ σημαντική γιατί η επέμβαση στο πεδίο συχνοτήτων μίας εικόνας είναι ένας από τους σημαντικότερους τρόπους τροποποίησης και επεξεργασίας της.

Για να κατανοήσουμε καλυτέρα το θεωρητικό υπόβαθρο που αναφέραμε και προηγουμένως θεωρούμε ότι βρισκόμαστε στο πεδίο των συχνοτήτων όπου θα εφαρμοστεί και το βαθυπερατό φίλτρο συνέλιξης (βλέπε παρακάτω blurring) στην εικόνα προκυμμένου να αποκόψει ψηλές συχνότητες. Για να πραγματοποιηθεί η συνέλιξη χρησιμοποιούμε τον μετασχηματισμό Laplace έτσι ώστε να ανάγουμε την επίλυση της διαφορικής εξίσωσης στην επίλυση μιας αλγεβρικής εξίσωσης.
Παρακάτω ακολουθεί η απόδειξη με βάση την μαθηματική ακολουθία:

Για οποιαδήποτε

$$x, y \in C(x \times y) \iff X, Y \tag{1}$$

Η συνέλιξη των δυο συναρτήσεων $f(t)$ και $g(t)$ υποδηλώνεται με:

$$(f \times g)(t)^2 \tag{2}$$

Επίσης, το θεώρημα συνέλιξης δίνει τον αντίστροφο μετασχηματισμό Laplace του γινομένου δύο μετασχηματισμένων συναρτήσεων:

$$L^{-1}\{F(S) \times G(s)\} = (f \times g)(t) \tag{3}$$

Έστω τα $f(t)$ και $g(t)$ είναι δυο συνεχείς συναρτήσεις του t. Αυτό σημαίνει ότι και η συνέλιξη $f(t)$ και $g(t)$ προκύπτει να είναι επίσης μια συνεχείς συνάρτηση του t που υποδηλώνεται από το $(f \times g)(t)$. Και ορίζεται από την ακόλουθη σχέση:

$$(f \times g)(t) = \int_{-\infty}^{+\infty} f(t-x)g(x)dx \tag{4}$$

Παρόλα αυτά, εάν η f και g είναι δυο αιτιατά σήματα τότε το $f(t)$ και $g(t)$ γράφεται $f(t)u(t)$ και $g(t)\ u(t)$ ως αντίστοιχα έτσι ώστε να ισχύει:

$$(f \times g) = \int_{-\infty}^{+\infty} f(t-x)u(t-x)g(x)u(x)dx = \int_{-\infty}^{+\infty} f(t-x)g(x)dx \tag{5}$$

Επίσης με βάση τα παραπάνω βήματα της ακολουθίας ισχύει ότι [1]

- **Αν x>t** τότε $u(t-x) = 0$
- **Αν x<0** τότε $u(x) = 0$

Β. Φίλτρα Συνέλιξης

Τα βαθυπερατά φίλτρα, φιλτράρουν τις υψηλές συχνότητες που βασικά είναι ανεπιθύμητα σήματα όπως για παράδειγμα ο θόρυβος. Εκτός όμως από τον θόρυβο "λειαίνουν" απότομες μεταβολές στην ένταση. Η διαδικασία αυτή συνεπάγεται στην θόλωση της εικόνας. Γενικότερα τρεις είναι οι βασικές κατηγορίες βαθυπερατών φίλτρων που διακρίνουμε:

- Φίλτρα μέσης τιμής (mean filter)
- Φίλτρα μορφής Gaussian (Gaussian filter)
- Φίλτρα διάμεσης τιμής (median filter)

Το φίλτρο συνέλιξης χρησιμοποιείται για να συνδυάσει τα pixel της εικόνας που βρίσκονται σε μορφή αριθμών από ακέραιους με τιμές που κυμαίνονται από 0-255 (στην περίπτωση RGB 3 bytes).

Κάθε ένα του οποίου χαρακτηρίζει την φωτεινότητα της εικόνας που αναλύεται, προσδίδοντας έτσι διαφορετικά εφέ στην εικόνα αναλόγως με το αποτέλεσμα που επιθυμούμε να επιτύχουμε. Τα σημαντικότερα εφέ είναι τα ακόλουθα:

[1] Σημείωση: Πρέπει να τονιστεί ότι τα (x, y) αποτελούν συντεταγμένες χώρου (καθορίζουν τις συντεταγμένες των pixels) ενώ τα (u, v) αποτελούν συντεταγμένες συχνοτήτων οι οποίες εκφράζονται σε κύκλους ανά εικόνα.

1. Identity
2. Edge detection
3. Sharpen
4. Box blur
5. Gaussian blur
6. Unsharp masking

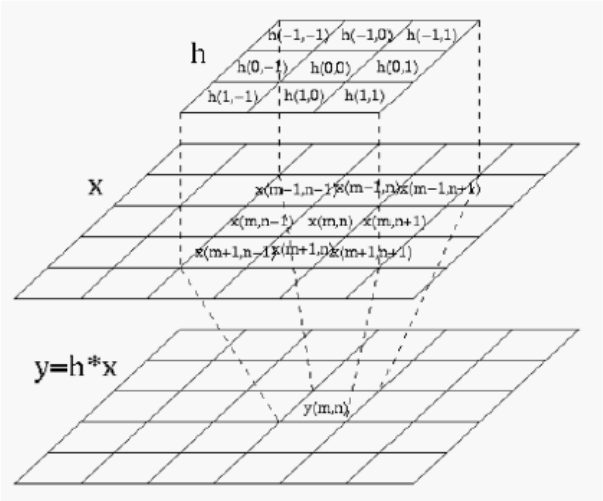
Στα πλαίσια της παρούσας εργασίας με βάση την δοθείσα εκφώνηση θα μας απασχολήσει μόνο το φίλτρο της θόλωσης μορφοποιημένο κατά Γκαούς (Gaussian blur 3 × 3). Το φίλτρο αυτό ανήκει στην κατηγορία των βαθυπερατών φίλτρων όπως αναφέραμε και προηγουμένως.

Η επεξεργασία της εικόνας επιτυγχάνεται πραγματοποιώντας την συνέλιξη ανάμεσα στην δισδιάστατη αναπαράσταση με τους ακέραιους της εικόνας και του φίλτρου συνέλιξης. Ο αριθμός των σειρών και των στηλών εξαρτάται από τον τύπο του προβλήματος και απο το επιθυμητό αποτέλεσμα προς επίτευξη. Για παράδειγμα σε ένα φίλτρο συνέλιξης διαστάσεων 3 × 3 το κέντρο βρίσκετε στην θέση (1,1). Τοποθετούμε το κέντρο αυτό του φίλτρου (μπορούμε να το δούμε σαν ένα παράθυρο) έτσι ώστε, να αντιστοιχίζεται κάθε φορά σε κάθε ακέραιο αριθμό (byte) του πίνακα της εικόνας. Ο ακέραιος αυτός αριθμός που γίνεται η αντιστοίχιση υποδηλώνει και το τρέχον pixel που θα διαμορφωθεί. Επαναλαμβάνουμε αυτή την διαδικασία της αντιστοίχισης με όλους τους ακέραιους αριθμούς του πίνακα της εικόνας προκειμένου να παραχθεί η τελική διαμόρφωση. Η μέθοδος υπολογισμού της διαμόρφωσης πραγματοποιείται ακολουθώντας τον εξής τύπο πάντα μιλώντας για φίλτρο διαστάσεων 3 × 3:

$$dst(x,y) = (src(x-1,y-1) \times a0 + src(x,y-1) \times a1)$$
$$src(x,y+1) \times a7 + \frac{src(x+1,y+1) \times a8)}{divisor}) + bias$$

Πιο συγκεκριμένα, το φίλτρο συνέλιξης με μήτρα 3 x 3 παίρνει το pixel (x-1, y-1) για το pixel που βρίσκεται στα (x, y) και το πολλαπλασιάζει με το (0, 0) και στη συνέχεια προσθέτει το pixel (x, y-1) πολλαπλασιασμένο με την τιμή της μήτρας στο (0, 1), και κ.λπ. μέχρι να πολλαπλασιαστούν όλες οι τιμές του πίνακα με την αντίστοιχη τιμή pixel. (Αυτό γίνεται για κάθε ακέραιο αριθμό που αναπαριστά την φωτεινότητα της εικόνας.) Τέλος, διαπιστώνει το άθροισμα, το διαιρεί με την αξία του διαιρέτη (στην περίπτωση μας με το 16 αφού εφαρμόζεται το Gaussian blur 3 × 3). Συνεπώς, στην συνέλιξη δισδιάστατου πίνακα με μήτρα $M \times N$, απαιτούνται $M \times N$ πολλαπλασιασμοί **Εικόνα 1.1**.

Η παραπάνω φόρμουλα όπως θα δούμε και από τον τύπο που θα προκύψει είναι διακριτή. Επίσης, είναι αυτή που ακολουθείτε στα πλαίσια της άσκησης και αποτελεί τον κανόνα για τον σχεδιαστικό αλγόριθμο υπολογισμού του σειριακού, άρα και του παράλληλου τμήματος. Σύμφωνα με την εξήγηση που δώσαμε προηγουμένως και σύμφωνα με τον τύπο που έχει δοθεί και στην εκφώνηση της άσκησης (ο οποίος παρεμπιπτόντως επιτυγχάνει γραμμικό χρόνο σύμφωνα με ασυμπτωτική ανάλυση) για την συνέλιξη μιας

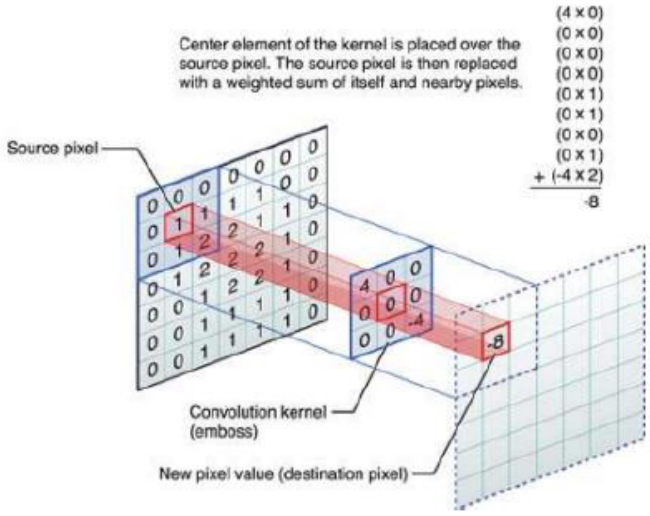


Εικόνα 1.1: Συνέλιξη δισδιάστατου πίνακα με φίλτρο h

εικόνας εισόδου I_I με το φίλτρο h ώστε να προκύψει η παραγωγή μιας εικόνας εξόδου I_0 ο μαθηματικός τύπος ορίζεται ως εξής:

$$\sum_{p=-s}^{s}\sum_{q=-s}^{s} I_1 \times (i-p, j-q) \times h(p,q) \tag{6}$$

Έστω ότι επιθυμούμε να υπολογίσουμε την συνέλιξη μια εικόνας στο σημείο $x[1,1]$ έτσι ώστε να προκύψει ένα $y[1,1]$ που είναι το άθροισμα των γινομένων των γειτονικών pixel με το φίλτρο h. Το αποτέλεσμα που ανάγεται από τον μαθηματικό τύπο της (6), σύμφωνα πάντα και με την εξήγηση υπολογισμού που δώσαμε προηγουμένως παρουσιάζεται στην παρακάτω εικόνα **Εικόνα 1.2**



Εικόνα 1.2: Συνέλιξη δισδιάστατου πίνακα μήτρας $M \times N$

Κάθε pixel είναι η τιμή χρώματος του τρέχοντος εικονοστοιχείου ή ένας γείτονας του, με την αντίστοιχη αξία της μήτρας φίλτρου. **Το κέντρο** της μήτρας φίλτρου πολλαπλασιάζεται με το τρέχον εικονοστοιχείο και τα άλλα στοιχεία της μήτρας φίλτρου με τα αντίστοιχα εικονοστοιχεία των γειτόνων. Μια λεπτότητα αυτής της διαδικασίας είναι τι γίνεται στην περίπτωση κατά μήκος των άκρων της εικόνας που βρίσκονται (βόρεια, Νότια, Ανατολικά, Δυτικά) της εικόνας όπου δεν μπορεί να τοποθετηθεί σωστά το παράθυρο του φίλτρου ώστε να εφαρμοστεί. Ένας εύκολος τρόπος για να διορθώσουμε αυτό το πρόβλημα είναι να αγνοήσουμε αυτές τις τιμές.

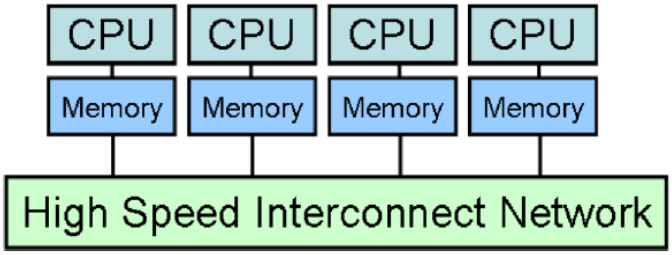
2. Ανάλυση Παραλληλοποίησης και Παραμετροποίησης

Πολλά παράλληλα προγράμματα χρησιμοποιούν την προσέγγιση (ένα πρόγραμμα-πολλά δεδομένα) ή SPMD (single program-multiple data) το οποίο είναι το πιο δημοφιλές της ταξινομίας Flynn που βρίσκουμε σε υπολογιστές σήμερα, όπου η εκτέλεση ενός προγράμματος ισοδυναμεί με την εκτέλεση πολλών προγραμμάτων, χάρη στην διακλάδωση του κώδικα με βάση δεδομένα όπως ο αριθμός κατάταξης κάθε διεργασίας.

Με παράλληλη επεξεργασία επιτυγχάνεται με ταυτόχρονη εκτέλεση εντολών (μιας εντολής κάθε φορά) στους διάφορους επεξεργαστές / πυρήνες. Κάθε διεργασία εκτελείται στο δικό της "αφιερωμένο" χώρο εκτέλεσης έχοντας το δικό της αποκλειστικό χώρο μνήμης. Για να επικοινωνήσουν οι διεργασίες πρέπει να μεταφερθούν τα δεδομένα από τη μνήμη της μιας στη μνήμη της άλλης, μέσω ενός διαύλου επικοινωνίας (bus interconnect). Αυτή η παράλληλη αρχιτεκτονική έχει γίνει γνωστή ως αρχιτεκτονική κατανεμημένης μνήμης (distributed memory architecture), όπου η μνήμη είναι κατανεμημένη στους επεξεργαστές, δηλαδή κάθε επεξεργαστής διαθέτει το δικό του χώρο.

Στην MPI, ο επικοινωνητής είναι μια συλλογή διεργασιών που μπορούν να στέλνονται μηνύματα από την μια στην άλλη. Μετά το ξεκίνημα ενός προγράμματος MPI, η διασύνδεση δημιουργεί έναν επικοινωνητή ο οποίος περιλαμβάνει όλες τις διεργασίες. Το όνομα του είναι MPI_COMM_WORLD.

Υλοποιήσεις του MPI αποτελούν το MPICH (η πιο γνωστή και αυτή που χρησιμοποιείται εδώ), το OpenMPI (μαζί με OpenMP), το MSMPI (Microsoft MPI το οποίο χρησιμοποιήθηκε με επιτυχία στην εκτέλεση στο Windows περιβάλλον) και το Intel MPI

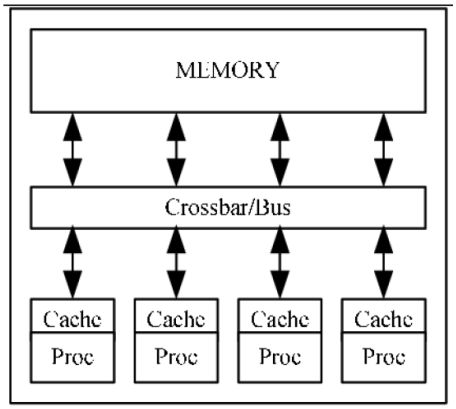


Εικόνα 2.1: System Bus

Η δεύτερη αρχιτεκτονική παράλληλης επεξεργασίας είναι αυτή της διαμοιραζόμε-

νης μνήμη (shared memory). Σύμφωνα με αυτή οι διεργασίες (που εδώ συνηθίζονται να αποκαλούνται νήματα) μοιράζονται ένα κοινό χώρο μνήμης. Τα νήματα μπορούν να επικοινωνούν μεταξύ τους χωρίς να στέλνουν "μηνύματα". Αυτό εφόσον μπορούν όλα να αναφέρονται σε κοινό χώρο διευθύνσεων. Δηλαδή, τη κύρια μνήμη (RAM) του εκάστοτε επεξεργαστή.

Για να επιτευχθεί η παραλληλοποίηση δεν αρκεί μονάχα το υλικό να την υποστηρίζει, αλλά πρέπει και να προγραμματίσουμε τα προγράμματα μας με κατάλληλο τρόπο ώστε να επωφεληθούμε από τις επιδόσεις της αρχιτεκτονικής. Με άλλα λόγια πρέπει και το υλικό και το λογισμικό να συνεργάζονται αρμονικά για να παραχθεί το επιθυμητό αποτέλεσμα σε γρηγορότερο χρόνο εν σύγκριση με το σειριακό.



Εικόνα 2.2: Shared Memory Model

Το ερώτημα όμως που τίθεται είναι γιατί να θέλουμε να χτίσουμε παράλληλα προγράμματα; Ο βασικός στόχος μας στην γραφή παράλληλων προγραμμάτων είναι συνήθως η βελτίωση των επιδόσεων. Οπότε τα επόμενα ερώτηματα που τίθενται είναι τι μπορούμε λοιπόν να περιμένουμε? Και πως μπορούμε να αξιολογήσουμε τα προγράμματα μας? Η απάντηση των ερωτημάτων αυτών πραγματοποιείται με την ανάλυση της θεωρητικής απόδοσης και παρουσιάζεται στις ακόλουθες υποενότητες.

- **Ανάλυση Επιτάχυνσης και Αποδοτικότητας(Επιδόσεις)**
- **Νόμος του Amdahl**
- **Επεκτασιμότητα**

Α. Ανάλυση Επιτάχυνσης και Αποδοτικότητας (Επιδόσεις)

Συνήθως, το καλύτερο που μπορούμε να ελπίζουμε ότι θα καταφέρουμε είναι να διαιρέσουμε εξίσου το φορτίο εργασίας μεταξύ των πυρήνων, χωρίς παράλληλα να επιβαρύνουμε με πρόσθετες εργασίες τους πυρήνες. Αν το επιτύχουμε και εκτελέσουμε το πρόγραμμα μας με p πυρήνες, χρησιμοποιώντας ένα νήμα ή διεργασία σε κάθε πυρήνα, τότε το παράλληλο πρόγραμμα μας θα εκτελείται p φορές ταχύτερα από το αντίστοιχο

σειριακό. Αν ονομάσουμε τον χρόνο εκτέλεσης του σειριακού προγράμματος T_{serial} και του παράλληλου $T_{parallel}$ τότε το καλύτερο που μπορούμε να πέτυχουμε είναι

$$T_{parallel} = \frac{T_{serial}}{p} \tag{7}$$

Όταν συμβαίνει αυτό λέμε ότι το παράλληλο πρόγραμμα μας έχει γραμμική επιτάχυνση **(linear speedup)**. Στην πράξη, είναι απίθανο να πέτυχουμε γραμμική επιτάχυνση επειδή η χρήση πολλών διεργασιών νημάτων οπωσδήποτε επιφέρει κάποιες επιβαρύνσεις. Επιπλέον, είναι πιθανό οι επιβαρύνσεις να αυξάνονται όσο αυξάνουμε το πλήθος των διεργασιών ή των νημάτων, αφού όσο περισσότερα είναι τα νήματα εκτέλεσης τόσο πιο πολλά νήματα θα χρειαστεί να προσπελάσουν κάποιο κρίσιμο τμήμα. Περισσότερες διεργασίες, επίσης σημαίνει ότι μάλλον περισσότερα δεδομένα θα χρειαστεί να μεταδίδονται μέσω του δικτύου. Συνεπώς, αν ορίσουμε την επιτάχυνση (speedup) ενός παράλληλου προγράμματος ως

$$S = \frac{T_{serial}}{T_{parallel}} \tag{8}$$

Τότε για την γραμμική επιτάχυνση είναι S=p πράγμα ασυνήθιστο. Επιπλέον όσο αυξάνει το p περιμένουμε ότι το S θα γίνει όλο και μικρότερο ποσοστό της ιδανικής γραμμικής επιτάχυνσης p. Ένας ακόμα τρόπος να εκφράσουμε αυτή την διαπίστωση είναι ότι το κλάσμα $\frac{S}{P}$ μάλλον θα ελαττώνεται όσο το p αυξάνεται. Η τιμή αυτή $\frac{S}{P}$ ονομάζεται αποδοτικότητα του παράλληλου προγράμματος. Αν χρησιμοποιήσουμε την προηγούμενη σχέση (8) για το S βρίσκουμε ότι η αποδοτικότητα είναι:

$$E = \frac{S}{P} = \frac{\frac{T_{serial}}{T_{parallel}}}{P} = \frac{T_{serial}}{P \times T_{parallel}} \tag{9}$$

Για την γραμμική επιτάχυνση είναι S=p και E=1. Στην πράξη, θα είναι πάντα S<p και E<1. Αν το μέγεθος του προβλήματος διατηρείται σταθερό, το E συνήθως μειώνεται όσο αυξάνουμε το p, ενώ αν διατηρούμε σταθερό το πλήθος των διεργασιών/νημάτων τα S και E συνήθως αυξάνονται όσο αυξάνουμε το μέγεθος του προβλήματος

Β. Νόμος του Amdahl

Ο νόμος του Amdahl αναφέρει ότι η επιτάχυνση ενός παράλληλου προγράμματος περιορίζεται από τις σειριακές του περιοχές. Πιο συγκεκριμένα, παρέχει ένα άνω όριο για την επιτάχυνση που μπορεί να πετύχει ένα παράλληλο πρόγραμμα: αν ποσοστό r του αρχικού σειριακού προβλήματος δεν παραλληλοποιηθεί, τότε δεν γίνεται να πετύχουμε επιτάχυνση καλύτερη από $\frac{1}{r}$, ανεξάρτητα από το πλήθος των διεργασιών/νημάτων που χρησιμοποιούμε. Στην πράξη, πολλά παράλληλα προγράμματα επιτυγχάνουν εξαιρετικές επιταχύνσεις. Ένας πιθανός λόγος για αυτή την προφανή αντίφαση είναι ότι ο νόμος του Amdahl δεν λαμβάνει υπόψη του ότι, καθώς το μέγεθος του προβλήματος αυξάνεται, το μη παραλληλοποιημένο τμήμα του προγράμματος συχνά μικραίνει σε σχέση με το παραλληλοποιημένο τμήμα.

Ο νόμος του Amdahl μπορεί να διατυπωθεί σύμφωνα με τον ακόλουθο τρόπο:

$$S_{latency(s)} = \frac{1}{(1-P) + \frac{P}{S}} \tag{10}$$

Όπου

- $S_{latency}$: Είναι η θεωρητική επιτάχυνση της εκτέλεσης ολόκληρης της διεργασίας.
- **S**: Είναι η επιτάχυνση του μέρους του έργου που επωφελείται από τους βελτιωμένους πόρους του συστήματος.
- **P**: Είναι η αναλογία του χρόνου εκτέλεσης του τμήματος που επωφελείται από τους βελτιωμένους πόρους οι οποίοι έχουν δεσμευτεί αρχικά.

Γ. Επεκτασιμότητα Weak Scaling – Νόμος Gustafson.

Η επεκτασιμότητα **(scalability)** είναι ένας όρος με ποικιλία σημασιών. Γενικά, μια τεχνολογία Θεωρείται επεκτάσιμη όταν είναι σε θέση να χειρίζεται προβλήματα διαρκώς αυξανόμενου μεγέθους. Τυπικά, ένα παράλληλο πρόγραμμα είναι επεκτάσιμο αν υπάρχει ένα ποσοστό αύξησης του μεγέθους του προβλήματος, τέτοιο ώστε, καθώς αυξάνεται το πλήθος των διεργασιών/νημάτων, η απόδοση του να παραμένει σταθερή. Ένα πρόγραμμα λέμε ότι είναι **ισχυρά επεκτάσιμο** (strongly scalable) αν το μέγεθος τον προβλήματος μπορεί να διατηρείται σταθερό, και **ασθενώς επεκτάσιμο** (weakly scalable) αν το μέγεθος του προβλήματος πρέπει να αυξάνεται κατά το ίδιο ποσοστό πού αυξάνεται και το πλήθος των διεργασιών/νημάτων.

Εδώ η επιτάχυνση υπολογίζεται θεωρητικά με την παρακάτω εξίσωση:

$$Speedup = S + N \times p \tag{11}$$

Η αποδοτικότητα υπολογίζεται με τον ακόλουθο τύπο:

$$Efficiency = \frac{T_1}{T_N} \times 100\% \tag{12}$$

Σημαντική σημείωση που συχνά περνά απαρατήρητη: Ας μην ξεχνάμε ότι όσο μεγαλώνει ο αριθμός των επεξεργαστών έχουμε και μεγαλύτερη αύξηση του κόστους. Όταν N επεξεργαστές εκτελούν μια εργασία σε χρόνο T, τότε χρειαζόμαστε N φορές περισσότερη ισχύ για να τους λειτουργήσουμε για δεδομένο χρονικό διάστημα. Φυσικά ένας επεξεργαστής σίγουρα θα χρειαζόταν πολύ περισσότερο χρόνο. Όλα αυτά πρέπει να λαμβάνονται υπόψη.

3. Σχεδιαστικές Αποφάσεις Κώδικα

Η σχεδίαση των παράλληλων προγραμμάτων παίζει καταλυτικό ρόλο στον τρόπο με τον οποίο πρέπει να μοιράσουμε το έργο του προγράμματος μεταξύ των διεργασιών/νημάτων, ώστε κάθε διεργασία να έχει περίπου το ίδιο φορτίο και να ελαχιστοποιείται η επικοινωνία. Δυστυχώς δεν υπάρχει κάποια αυτοματοποιημένη διαδικασία που να μπορούμε να ακολουθήσουμε.

Α. Σχεδίαση Παράλληλων Προγραμμάτων

Ωστόσο, ο Ιαν Foster παρέχει ένα γενικό πλαίσιο με τα απαραίτητα βήματα που πρέπει να πραγματοποιηθούν για ένα σωστό σχεδιασμό παράλληλων προγραμμάτων τα οποία διατυπώνονται ως εξής:

- **Διαμέριση (partitioning)**: Χωρίζουμε του υπολογισμούς και τα δεδομένα πάνω στα οποία θα πραγματοποιηθούν αυτοί οι υπολογισμοί σε μικρές εργασίες. Η προσοχή εδώ πρέπει να είναι εστιασμένη στην αναγνώριση εργασιών που μπορούν να εκτελούνται παράλληλα

- **Επικοινωνία (communication)**: Προσδιορίζουμε τις απαραίτητες επικοινωνίες μεταξύ των εργασιών που καθορίσαμε στο προηγούμενο βήμα

- **Συσσώρευση ή Συνάθροιση (agglomeration/aggregation)**:Ανασκόπηση βημάτων 1, 2 με σκοπό το σχηματισμό κατάλληλου αλγορίθμου για εκτέλεση σε κάποιο είδος παράλληλου υπολογιστή. Πολλές αποφάσεις πρέπει να ληφθούν σε αυτό το στάδιο, οι οποίες θα καθορίσουν την ποιότητα εκτέλεσης του προγράμματος. Στόχος είναι η παραγωγή ορθού αποτελέσματος για κάθε είδος εισόδου στο πρόβλημα (soundness & completeness).

- **Αντιστοίχιση(mapping)**: Ανάθεση διεργασιών για την επίλυση των υποπροβλημάτων. Το πλήθος διαθέσιμων υπολογιστικών μονάδων/ επεξεργαστών συνήθως διαφέρει από το πλήθος των υποπροβλημάτων. Επιζητούμε τον ισοζυγισμό των επεξεργαστών. Αυτό πρέπει να γίνεται έτσι ώστε οι επικοινωνίες να ελαχιστοποιούνται και κάθε διεργασία/νήμα να έχει περίπου το ίδιο φόρτο εργασίας.

Το παραπάνω πλαίσιο έχει μείνει γνωστό και ως **μεθοδολογία Foster**.

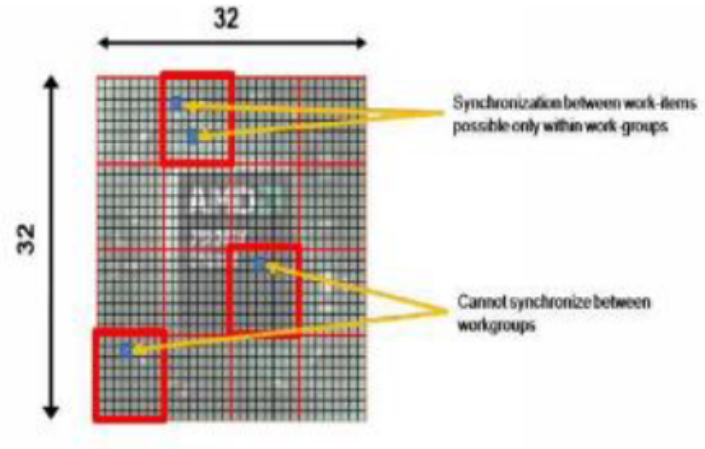
Β. Τεχνικές και Πρακτική Εφαρμογή

Σύμφωνα με την μεθοδολογία Foster που εξετάσαμε προηγουμένως, σε αυτή την υπόενότητα θα δούμε πιο αναλυτικά πως εφαρμόζεται η συγκεκριμένη μεθοδολογία στο πρόβλημα που καλούμαστε να υλοποιήσουμε. Για να είμαστε πιο σαφείς θα καλύψουμε όλες τις τεχνικές και σχεδιαστικές αποφάσεις που χρησιμοποιήσαμε για το σχεδιασμό του παράλληλου τμήματος αυτής της εργασίας.

1. Σε αυτό το στάδιο προσπαθούμε να εξάγουμε τρόπους παραλληλισμού της εκτέλεσης. Ένας καλός διαμερισμός διαιρεί σε μικρά κομμάτια τόσο τους υπολογισμούς όσο και τα δεδομένα του προβλήματος. Έτσι χωρίζουμε την εικόνα σε πιο μικρές εικόνες ανάλογα με τον αριθμό διεργασιών που δίνονται.

   Σαν πρώτη επιλογή κάθε διεργασία είναι υπεύθυνη για ένα κομμάτι με υποδιαιρεμένες διαστάσεις της αρχικής εικόνας. Δηλαδή, σαν να έχει η κάθε διεργασία μια δικιά της ξεχωριστή εικόνα. Ο λόγος που γίνεται αυτό είναι διότι επιθυμούμε την ισοκατανομή του φορτίου ανάμεσα στις διεργασίες, εφόσον η κάθε διεργασία θα έχει τη μνήμη της όπου θα περιέχεται το αντίστοιχο κομμάτι της εικόνας. Αν ο αριθμός του κατάλληλου τμήματος της αρχικής εικόνας είναι n και έχουμε

comm_sz πυρήνες ή διεργασίες μπορούμε να ορίσουμε το $local_n = \frac{n}{comm_{sz}}$. Τότε σε κάθε διεργασία μπορούμε απλώς να αναθέσουμε μπλοκ με local_n διαδοχικά υποτμήματα. Το κάθε υποτμήμα είναι αφιερωμένο σε μια διεργασία. Η κάθε διεργασία έχει τέσσερις γείτονες. Οι γειτονικές διεργασίες αναφέρονται ως North, East,South, West **(Foster Partitioning)**.



Εικόνα 3.1: Κανόνας Διαμέριση σε διεργασίες μικρότερων υποτμημάτων

Ας υποθέσουμε για παράδειγμα ότι σε μια εικόνα έχει επιλεγεί κατάλληλο τμήμα μεγέθους της, τέτοιο ώστε να δημιουργείται ένας πίνακας(32x32) από μπλοκ (/διεργασίες). Έχουμε ένα πλήθος σειρών από μπλοκ (prows) και ένα πλήθος στηλών από μπλοκ (pcols). Σύμφωνα και με την Εικόνα 3.1 έστω ότι επιλέγουμε να χρησιμοποιήσουμε 4 πυρήνες τότε θα έχουμε 8 x 8 blocks θέσεων(pixel) 32/4 που θα ανατεθούν σε κάθε πυρήνα ξεχωριστά προκειμένου να φέρει εις πέρας τον κώδικα που του έχει ανατεθεί.

2. Σαν δεύτερο στάδιο, απώτερος μας στόχος είναι να μειώσουμε όσο γίνεται περισσότερο τις ασήμαντες επικοινωνίες. Για παράδειγμα όταν αρχικοποιείται το πρόβλημα όλες οι διεργασίες θα πρέπει να πάρουν κάποια ορίσματα που θα είναι κοινά σε όλους τους επεξεργαστές. Για το σκοπό αυτό είναι ανούσιο να στέλνουμε ξεχωριστά σε κάθε διεργασία τα ίδια ορίσματα διότι αυτό θα ήταν σπατάλη χρόνου. Συνεπώς για να πραγματοποιήσουμε ένα αποδοτικό τρόπο συλλογικής επικοινωνίας όπου τα δεδομένα που ανήκουν σε μια διεργασία στέλνονται σε όλες τις διεργασίες του επικοινωνητή, χρησιμοποιήσαμε την συνάρτηση **εκπομπής** που παρέχει το MPI την **MPI Bcast**.

   Εκτός από το πρόβλημα που μόλις αναφέρθηκε είναι ολοφάνερο ότι θα υπάρχουν και κάποια ορίσματα που θα είναι τελείως διαφορετικά από επεξεργαστή σε επεξεργαστή για παράδειγμα στην Εικόνα 3.1 όπως είδαμε στο προηγούμενο παράδειγμα που έγινε η διαμέριση σε block, οι διεργασίες που δεν είναι γειτονικές μεταξύ τους δεν χρειάζεται να γνωρίζουν τα ορίσματα των μη γειτονικών διεργασιών. Επομένως, για να είναι πιο αποδοτικό το πρόγραμμα μας πρέπει να γράψουμε μια συνάρτηση που να διαβάζει την απαιτούμενη πληροφορία για παράδειγμα στην διεργασία 0 αλλά να στέλνει μόνο τα απαραίτητα υπό τμήματα από

blocks σε κάθε μια από τις άλλες διεργασίες. Η συνάρτηση αυτή είναι **MPI_Scatter**. Η MPI_ Scatter διαιρεί τα δεδομένα στα οποία αναφέρεται ο δείκτης σε τμήματα, με το πρώτο τμήμα να μεταβιβάζεται στην διεργασία 0 το δεύτερο στην διεργασία 1 κ.ο.κ. Είναι προφανές ότι το παράλληλο πρόγραμμα μας θα ήταν άχρηστο αν δεν υπάρχει μια συνάρτηση η οποία να δέχεται αυτά τα ορίσματα του κάθε επεξεργαστή αναλόγως με το κατανεμημένο τμήμα που της έχει ανατεθεί **MPI_Gather**.

Τέλος, προχωρώντας τώρα σε θέματα κώδικα σαν επιλογή έχουμε αποφασίσει να χρησιμοποιήσουμε **non blocking** αποστολές και λήψεις μηνυμάτων επιδιώκοντας με αυτόν τον τρόπο να κάνουμε την συνέλιξη για τα κεντρικά κομμάτια του πίνακα (inner data) και αφήνοντας για αργότερα τον υπολογισμό των περιφερειακών κομματιών(outer and corner data). Αυτό δίνει την δυνατότητα σε μια διεργασία να ασχολείται με ένα κομμάτι που της έχει ανατεθεί όσο περιμένει να έρθουν τα γειτονικά κελιά του πίνακα. Για την λήψη και την αποστολή αυτών των μηνυμάτων χρησιμοποιούνται οι συναρτήσεις **MPI_Isend - MPI_Irecv** οι οποίες όπως προαναφέρθηκε και προηγουμένως είναι non blocking. Το πλεονεκτήματα που μας προσφέρουν και κάνουν το πρόγραμμα μας αποδοτικότερο είναι τα εξής:

- Μπορεί να αποφύγει πιθανό αδιέξοδο(η διεργασία να κρεμάσει).
- Επιτρέπει τον διαχωρισμό μεταξύ της αρχικοποίησης της επικοινωνίας μέχρι και την ολοκλήρωση.
- Μειώνει την καθυστέρηση με την έγκαιρη επιστροφή λήψης αποτελέσματος **(Foster Communication)**.

3. Ένας ακόμα τρόπος να μειώσουμε τον συνολικό αριθμό των μηνυμάτων που στέλνουμε, προκειμένου να βελτιώσουμε σημαντικά τις επιδόσεις των προγραμμάτων μας είναι να ομαδοποιήσουμε τα δεδομένα που στέλνονται περιορίζοντας έτσι, πιθανή υπερφόρτωση ανταλλαγής μηνυμάτων. Για το λόγο αυτό, για την αποστολή και την λήψη μηνυμάτων αποφασίσαμε να γίνουν με την βοήθεια Datatypes (vector, contiguous) δεδομένων που παρέχει το MPI όπου ήταν δυνατό North, East,South, West. Η βασική ιδέα είναι ότι αν μια συνάρτηση που στέλνει δεδομένα γνωρίζει τους τύπους και τις σχετικές θέσεις στην μνήμη μιας συλλογής στοιχείων δεδομένων, τότε μπορεί να συλλέξει τα στοιχεία από την μνήμη πριν την αποστολή τους.

4. Για να βεβαιωθούμε ότι η εκτέλεση του προβλήματος από τις διεργασίες που έχουμε αναθέσει θα εκτελεστεί την ίδια ακριβώς στιγμή η MPI μας παρέχει μια συνάρτηση συλλογικής επικοινωνίας MPI_Barrier η οποία διασφαλίζει ότι καμία διεργασία δεν θα επιστρέφει από την κλήση της συνάρτησης πριν ξεκινήσει την κλήση και κάθε άλλη διεργασία που ανήκει στον ίδιο επικοινωνητή. Γι' αυτό η συνάρτηση MPI_Barrier πρέπει να τοποθετείται πάντα πριν την διαδικασία έναρξης της χρονομέτρησης όπως και έγινε στο παράλληλο πρόγραμμα μας.

5. Τέλος προκειμένου να κάνουμε ακόμα αποδοτικότερη την εργασία μας, έχουμε προσθέσει την χρήση Parallel IO, όπου επιτυγχάνεται ταχύτερος χρόνος διαβάσματος ή γραψίματος των blocks κατά την μεταφορά δεδομένων από τον δίσκο στην μνήμη. Αφού η διαδικασία γίνεται παράλληλα και κάθε διεργασία διαβάζει

το κομμάτι της εικόνας που της αναλογεί. Για την εκπλήρωση αυτού του σκοπού χρησιμοποιήθηκαν οι συναρτήσεις που προσφέρει το MPI, οι οποίες περιγράφονται ονομαστικά παρακάτω:

- MPI_File_open
- MPI_File_get_size
- MPI_ File_seek
- MPI_File_read
- MPI_File_close

Γενικότερα μια άλλη τεχνική που ενσωματώνουμε παρόλο που δεν έχει να κάνει με το σχεδιασμό του παράλληλου κομματιού και γενικότερα με την χρήση του MPI είναι ότι χρησιμοποιούμε συναρτήσεις τύπου Inline για να βελτιώσουμε το πρόγραμμα μας σε επιδόσεις. Οι λόγοι αναφέρονται παρακάτω και είναι οι εξής:

- Στην κλήση των συναρτήσεων **Inline** δεν προστίθεται επιπρόσθετος χρόνος (Overhead).
- Όταν χρησιμοποιούμε αυτές τις συναρτήσεις επιτρέπουμε στον μεταγλωτιστή να πραγματοποιήσει κάποιες βελτιστοποιήσεις κατά την κλήση των συναρτήσεων οι οποίες δεν είναι εφικτές όπως με την κλήση κανονικών συναρτήσεων
- Οι Inline Functions μπορεί να είναι χρήσιμες (αν είναι μικρή) για ενσωματωμένα συστήματα, επειδή αυτές οι συναρτήσεις μπορεί να αποδώσουν λιγότερο κώδικα σε σύγκριση με την κλήση κανονικών συναρτήσεων.

Γ. Υβριδικό Σύστημα (MPI+ OpenMP)

Το OpenMP είναι μια διεπαφή προγραμματισμού εφαρμογών (API) που υποστηρίζει την κοινόχρηστη μνήμη(multi-πλατφόρμα) και χρησιμοποιεί ένα φορητό, κλιμακωτό μοντέλο που δίνει στους προγραμματιστές μια απλή και ευέλικτη διεπαφή για την ανάπτυξη παράλληλων εφαρμογών. Αποτελείται από ένα σύνολο οδηγιών μεταγλωτιστή, ρουτίνας βιβλιοθηκών και μεταβλητές περιβάλλοντος που επηρεάζουν τη συμπεριφορά του χρόνου εκτέλεσης

Στο κομμάτι του OpenMP σκοπός είναι, τα κομμάτια που εκτελούνται τοπικά να παραλληλοποιηθούν με τη χρήση thread ώστε να βελτιώνεται η ήδη υλοποιημένη εκδοχή του MPI. Μια εφαρμογή που έχει κατασκευαστεί με το υβριδικό μοντέλο παράλληλου προγραμματισμού μπορεί να εκτελεστεί σε ένα σύμπλεγμα υπολογιστών που χρησιμοποιεί και το OpenMP και τη διεπαφή μηνυμάτων (MPI), έτσι ώστε το OpenMP να χρησιμοποιείται για παραλληλισμό εντός ενός κόμβου (multi-core) ενώ το MPI να χρησιμοποιείται για παραλληλισμό μεταξύ κόμβων. Συγκεκριμένα στην άσκηση αυτή το σημείο στο οποίο κρίναμε πως αυτή η υλοποίηση θα βοηθούσε από άποψη χρόνου είναι το σημείο του βρόχου στο οποίο διατρέχουμε με επανάληψη τον τοπικό πίνακα για κάθε διεργασία προκειμένου να πραγματοποιηθεί η συνέλιξη του τμήματος που έχει διαμεριστει. Το πλαίσιο εκτέλεσης του OpenMP διαχειρίζεται στο παρασκήνιο όλες τις χαμηλού επιπέδου λεπτομέρειες για την εξυπηρέτηση των threads. Κρίσιμο και σύνηθες σημείο για

την έναρξη μιας παράλληλης περιοχής είναι οι βρόχοι. Ένας βρόχος με τα παρακάτω χαρακτηριστικά μπορεί να παραλληλοποιηθεί, διότι δεν υπάρχουν αλληλεξαρτήσεις στα δεδομένα:

- Όλες οι αναθέσεις σε κοινή μνήμη γίνονται σε πίνακες.
- Κάθε στοιχείο που βρίσκεται σε συγκεκριμένη θέση μνήμης ελέγχεται από το πολύ μια επανάληψη του βρόχου.
- Καμιά επανάληψη δεν διαβάζει ή γράφει δεδομένα από άλλη επανάληψη του βρόχου.

Με αυτές τις παραδοχές η εκτέλεση του βρόχου μπορεί να γίνει παράλληλα από πολλαπλές διεργασίες. Σκόπιμο είναι επίσης κανείς να αναρωτηθεί επιπλέον τι είναι αυτό που διαμοιράζεται μεταξύ των threads; Τα ορίσματα αυτά είναι τα εξής:

- Καθολικές μεταβλητές (file scope variables) είναι κοινές εξορισμού.
- Οτιδήποτε δηλώνεται μέσα σε shared() κατά την έναρξη της παράλληλης περιοχής είναι κοινό.

Για την αποφυγή ορισμένων δυσάρεστων σφαλμάτων στα προγράμματα προτείνεται η χρήση

```
pragma omp parallel
```

ξεχωριστά από τη δήλωση της εκάστοτε λειτουργίας (πχ. εκτέλεση βρόχου for). Για παράδειγμα προτιμάται το παρακάτω:

```
pragma omp parallel num_threads(2)
int i = 0;
pragma omp for private (prow, pcol)
```

Σε αντίθεση με:

```
pragma omp parallel num_threads (2) for private (prow, pcol)
```

Επίσης **προσοχή!** στο σημείο που δηλώνονται οι μεταβλητές σε ένα παράλληλο χωρίο. Εάν δηλωθούν έξω από αυτές τότε είναι κοινές στα νήματα. Διαφορετικά είναι ιδιωτικές στα νήματα.

4. Οδηγίες Μεταγλώττισης και Εκτέλεση Πηγαίου Κώδικα

Αφού πλέον έχουμε αναλύσει πλήρως το δοθέν πρόβλημα καθώς και τις τεχνικές και την αρχιτεκτονική που ακολουθήσαμε για τη σχεδίαση του προβλήματος τόσο για το σειριακό όσο και για το παράλληλο τμήμα, είμαστε πλέον σε θέση για την μεταγλώτιση και την εκτέλεση του παραδοτέου. Σε αυτή την ενότητα παρουσιάζεται ο αναλυτικός

οδηγός εκτέλεσης του κώδικα. Εδώ, θα επεξηγήσουμε τις λεπτομέρειες εγκατάστασης για ότι κριθούν απαραίτητα που θα χρειαστούν και για την μετέπειτα εκτέλεσή τους. Λόγο του ότι υπάρχει ένας αναλυτικός οδηγός για την εγκατάσταση του MPICH με οδηγίες εκτέλεσης στον αντίστοιχο ιστοχώρο του μαθήματος θεωρούμε ότι η διαδικασία αυτή είναι αυτονόητη οπότε παραλείπεται.[2]

Αφού βεβαιωθούμε ότι έχουν εγκατασταθεί σωστά τα αντίστοιχα μονοπάτια εκτέλεσης του MPICH στο περιβάλλον του Unix περιηγούμαστε στο αντίστοιχο μονοπάτι του φακέλου που εδρεύουν οι φάκελοι. Έπειτα αφού βεβαιωθούμε ότι έχει εγκατασταθεί το CMake είμαστε έτοιμοι για την μεταγλώτιση του κώδικα.

Το CMake είναι ένα σύστημα ανοιχτού κώδικα που διαχειρίζεται τη διαδικασία δημιουργίας σε ένα λειτουργικό σύστημα με τρόπο ανεξάρτητο από τον μεταγλωτιστή. Σε αντίθεση με πολλά συστήματα πολλαπλών πλατφορμών, το CMake έχει σχεδιαστεί για να χρησιμοποιείται σε συνδυασμό με το εγγενές περιβάλλον δημιουργίας. Τα απλά αρχεία διαμόρφωσης που είναι τοποθετημένα σε κάθε κατάλογο προέλευσης (που ονομάζονται αρχεία CMakeLists.txt) χρησιμοποιούνται για τη δημιουργία τυπικών αρχείων δημιουργίας (π.χ. makefiles σε Unix και έργων / χώρων εργασίας στα Windows MSVC).

Παρακάτω ακολουθεί ένα μέρος του αρχείου του CMake το οποίο είναι υπεύθυνο για την εύρεση του μεταγλωτιστή GCC του προτύπου C99 για την γλώσσα C καθώς και την εύρεση του μεταγλωτιστή MPI(mpicc). Επίσης παρακάτω ορίζονται οι μεταγλωτιστές "περιτύλιξης" που χρησιμοποιούνται τόσο για την μεταγλώτιση εφαρμογών του Open MPI όσο και για του MPICH. Με αυτόν το τρόπο ανιχνεύουμε για πιθανά σφάλματα στην εφαρμογή του παράλληλου κώδικα μας. Δηλαδή, αντί να χρησιμοποιήσουμε (για παράδειγμα) gcc για να μεταγλωτίσουμε το πρόγραμμά μας, χρησιμοποιούμε το mpicc.

```
project(Hybrid_ImageConvolution C)
set(CMAKE_C_STANDARD 99)
set(CMAKE_C_COMPILER / usr / bin / mpicc)
set(SOURCE_FILES Hybrid_Main.c ConvolutionProcess.c Parallel_ReadWriteIO.c)
set(SOURCE_FILES ConvolutionProcess.h MPI_Setup.h Communication.h)
add_executable(Hybrid_ImageConvolution Hybrid_Main.c)
// Generic MPI compilers
set(_MPI_C_COMPILER_NAMES "mpicc|mpcc|mpicc_r|mpcc_r")
set(_MPI_CXX_COMPILER_NAMES "mpicxx|mpiCC|mpcxx|mpCC|mpic[++]|mpc[++]")
// GNU compiler names
set(_MPI_GNU_C_COMPILER_NAMES "mpigcc|mpgcc|mpigcc_r|mpgcc_r")
set(_MPI_GNU_CXX_COMPILER_NAMES "mpig[++]|mpg[++]|mpig[++]_r|mpg[++]_r")
// Intel MPI compiler names
set(_MPI_Intel_C_COMPILER_NAMES "mpiicc")
set(_MPI_Intel_CXX_COMPILER_NAMES "mpiicpc|mpiicxx|mpiic[++]|mpiiCC")
```

[2] **Προσοχή!!** Οι οθόνες εκτέλεσης που θα αναφερθούν σε αυτή την ενότητα αφορούν την γενική διαδικασία εκτέλεσης του προγράμματος δεν αντιπροσωπεύουν τις μετρήσεις που θα δούμε στο επόμενο κεφάλαιο οι οποίες έχουν πραγματοποιηθεί στο cluster της σχολής

```
panos@ubuntu:~/CLionProjects/MPI_ImageConvolution$ cmake CMakeLists.txt
-- The C compiler identification is GNU 7.3.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Configuring done
-- Generating done
-- Build files have been written to: /home/panos/CLionProjects/MPI_ImageConvolution
```

Το CMake εντοπίζει τις απαραίτητες ρυθμίσεις για το περιβάλλον του Linux δηλαδή τα μονοπάτια που έχουν δοθεί για τους αντίστοιχους μεταγλωτιστές καθώς και άλλες πιθανές ρυθμίσεις που αφορούν την συνολική δομή του project.

```
panos@ubuntu:~/CLionProjects/MPI_ImageConvolution$ make
Scanning dependencies of target MPI_ImageConvolution
[ 50%] Building C object CMakeFiles/MPI_ImageConvolution.dir/main.c.o
[100%] Linking C executable MPI_ImageConvolution
[100%] Built target MPI_ImageConvolution
```

Μόλις δημιουργηθεί το Makefile, η εντολή make μπορεί να χρησιμοποιηθεί για την κατασκευή του έργου:

```
panos@ubuntu:~/CLionProjects/MPI_ImageConvolution$ mpicc ImageConvolution_Main.c
panos@ubuntu:~/CLionProjects/MPI_ImageConvolution$ mpirun -np 2 ./a.out waterfall_grey_1920_2520.raw 1920 2520 2 grey
Raw file was save with success at current working dir: /home/panos/CLionProjects/MPI_ImageConvolution
Time elapsed: 0.1998 ms
```

Σε αυτή την οθόνη εκτέλεσης παρουσιάζεται το τελικό αποτέλεσμα το οποίο έχει ανατεθεί σε δυο διεργασίες με δυο επαναλήψεις και επιστρέφει την καινούργια διαμορφωμένη εικόνα που έχει υποστεί την επεξεργασία του φίλτρου. Πιο συγκεκριμένα, παρακάτω αναλύονται όλα τα απαραίτητα ορίσματα που πρέπει ο χρήστης να καθορίζει, με την σωστή σειρά σαν είσοδο στο πρόγραμμα τα οποία είναι τα εξής:

1. Την έξοδο της εκτελέσιμης διαδρομής / <όνομα αρχείου>
2. Το μονοπάτι της εικόνας προς επεξεργασία / <όνομα εικόνας>
3. Το πλάτος της εικόνας
4. Το ύψος της εικόνας
5. Πόσες φορές θα εφαρμοστεί το φίλτρο δηλαδή πόσες επαναλήψεις θα πραγματοποιηθούν για τον έλεγχο για σύγκλιση.
6. Η εικόνα είναι τύπου «RGB» ή «Grey»

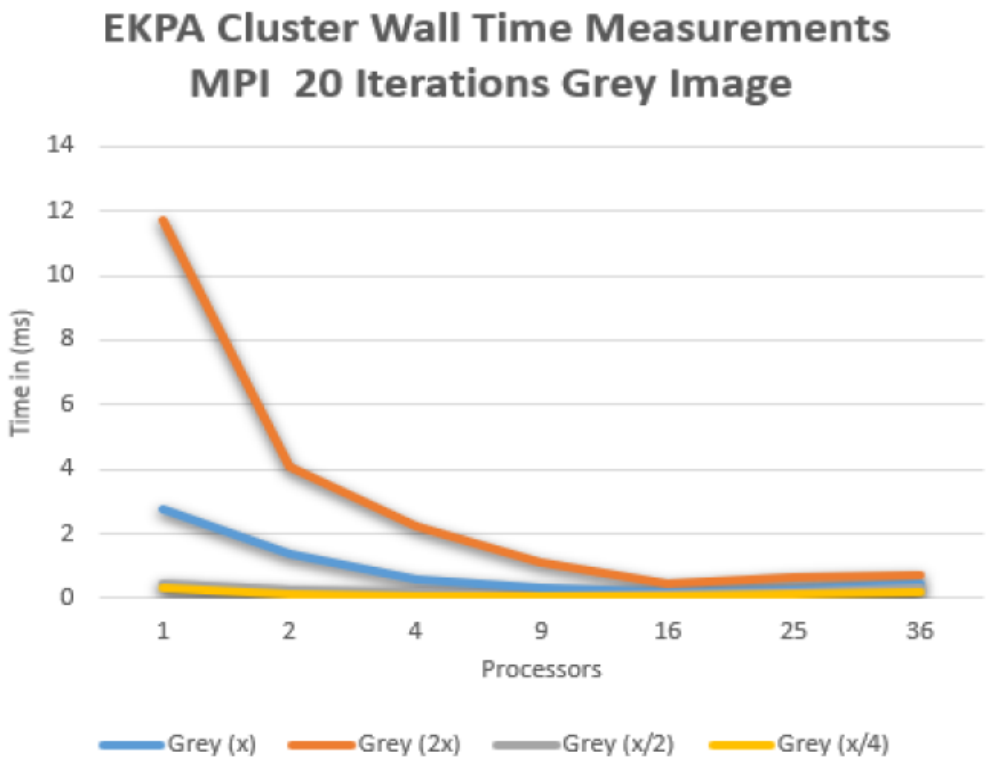
Εικόνα 4.1: Οθόνη εκτέλεσης εφαρμογής φίλτρου για 40 επαναλήψεις

Εικόνα 4.2: Οθόνη εκτέλεσης φίλτρου για 60 επαναλήψεις

Στην **Εικόνα 4.1** και στην **Εικόνα 4.2** παρουσιάζονται παραδείγματα από τις οθόνες εκτέλεση των RGB εικόνων, με τη κατάλληλη εφαρμογή του φίλτρου συνέλιξης της θόλωσης (blurring) που έχουν υποστεί. Παρατηρούμε ότι όσο ο αριθμός των επαναλήψεων αυξάνεται σταδιακά, προκαλείται και η αλλοίωση της δοθείσας εικόνας.

5. Μετρήσεις Επιδόσεις Κλιμάκωση Επεκτασιμότητα

Α. Μετρήσεις MPI Εικόνα χρώματος γκρι

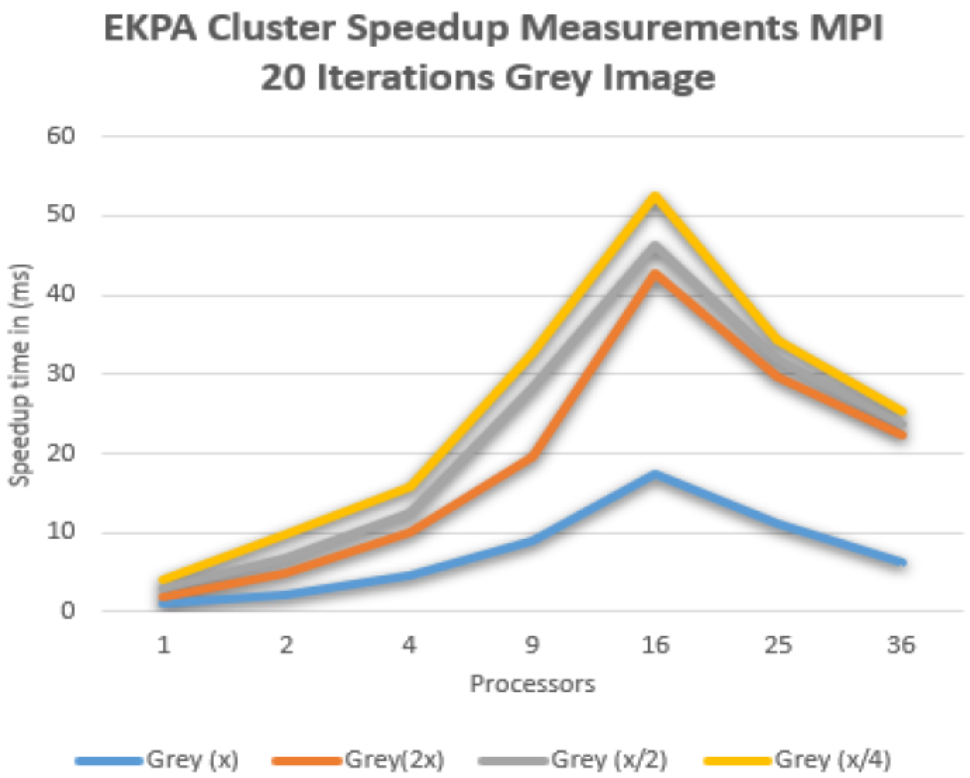


Εικόνα 5.1: Οθόνη εκτέλεσης γραφήματος που αναπαριστά τον χρόνο εκτέλεσης επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Wall Time Measurements MPI 20 Iterations Grey Image				
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	2.79	11.68	0.43	0.31
2	1.36	4.07	0.24	0.1
4	0.59	2.21	0.18	0.09
9	0.31	1.09	0.05	0.07
16	0.16	0.46	0.12	0.05
25	0.25	0.63	0.21	0.12
36	0.45	0.72	0.32	0.21

Πίνακας 5.1: Χρόνοι εκτέλεσης σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών

Σύμφωνα με το γράφημα της **Εικόνας 5.1** δεν προκαλεί έκπληξη το γεγονός ότι όταν το πεδίο «επεξεργαστές» διατηρείται σταθερό και αυξάνεται το τμήμα της αρχικής εικόνας οι χρόνοι εκτέλεσης αυξάνονται. Βέβαια καθώς αυξάνουμε το πλήθος των επεξεργαστών οι χρόνοι εκτέλεσης αρχικά μειώνονται. Όμως μετά από κάποιο σημείο οι χρόνοι εκτέλεσης παρατηρούμε ότι αρχίζουν να χειροτερεύουν αντί να βελτιώνονται σύμφωνα και από τον **Πίνακα 5.1** όπως για παράδειγμα φαίνεται με την χρήση 25, 36 επεξεργαστών αντίστοιχα όπου οι χρόνοι εκεί χειροτερεύουν αντί να καλυτερεύουν.



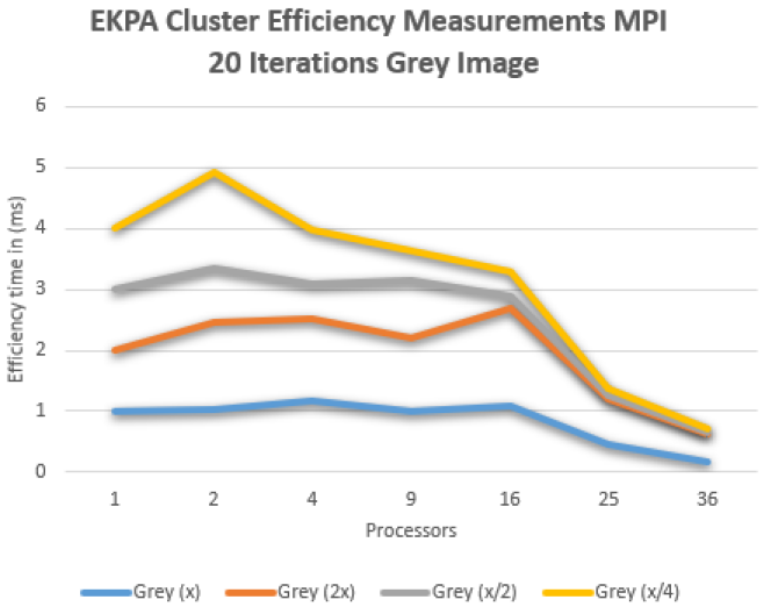
Εικόνα 5.2: Οθόνη εκτέλεσης γραφήματος που αναπαριστά την **επιτάχυνση(Speedup)** επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster SpeedUp Measurements MPI 20 Iterations Grey Image				
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	1	1	1	1
2	2.051470588	2.86977887	1.791666667	3.1
4	4.728813559	5.285067873	2.388888889	3.444444444
9	9	10.71559633	8.6	4.428571429
16	17.4375	25.39130435	3.583333333	6.2
25	11.16	18.53968254	2.047619048	2.583333333
36	6.2	16.22222222	1.34375	1.476190476

Πίνακας 5.2: Επιταχύνσεις σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών

Το πιο ευρέως χρησιμοποιούμενο μέτρο της σχέσης μεταξύ των χρόνων εκτέλεσης ενός σειριακού προγράμματος και του αντίστοιχου παράλληλου είναι η επιτάχυνση. Στην **Εικόνα 5.2** παρουσιάζεται η επιτάχυνση κατάλληλου τμήματος εικόνας συναρτήση του αριθμού των επεξεργαστών. Παρατηρούμε ότι η επιτάχυνση αυξάνεται σταδιακά έως ότου και την χρήση 16 διεργασιών, στην συνέχεια όμως με την χρήση 25,36

διεργασιών παρατηρούμε ότι η επιτάχυνση μειώνεται σταδιακά. Για μεγάλο αριθμό διεργασιών και σχετικά μικρό τμήμα διαμέρισης της εικόνας η επιτάχυνση ελαττώνεται σημαντικά. Με αποκορύφωμα την χρήση 36 διεργασιών ή πυρήνων όπου η επιτάχυνση αγγίζει την χειρότερη περίπτωση $\approx 1.3.$, **σχεδόν γραμμική επιτάχυνση**.



Εικόνα 5.3: Οθόνη εκτέλεσης γραφήματος που αναπαριστά την **αποδοτικότητα(Efficiency)** επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

	EKPA Cluster Efficiency Measurements MPI 20 Iterations Grey Image			
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	1	1	1	1
2	1.025735294	1.434889435	0.895833333	1.55
4	1.18220339	1.321266968	0.597222222	0.861111111
9	1	1.190621814	0.955555556	0.492063492
16	1.08984375	1.586956522	0.223958333	0.3875
25	0.4464	0.741587302	0.081904762	0.103333333
36	0.172222222	0.450617284	0.037326389	0.041005291

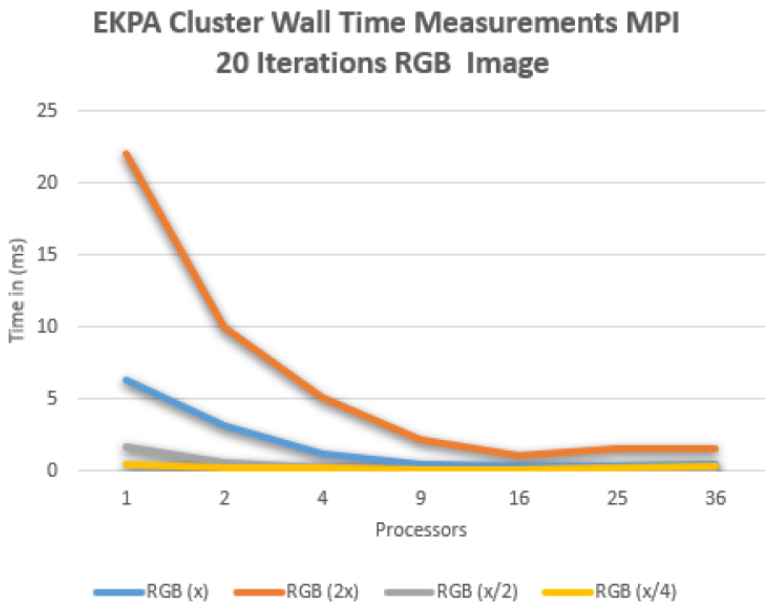
Πίνακας 5.3: Αποδοτικότητες σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών.

Όπως αναφέραμε και προηγουμένως το παράλληλο πρόγραμμα μας δεν πλησιάζει στην γραμμική επιτάχυνση για μικρά τμήματα εικόνας και μεγάλο αριθμό επεξεργαστών. Μήπως αυτό σημαίνει ότι το πρόγραμμα μας δεν είναι καλοφτιαγμένο; Η απά-

ντηση είναι ότι το γεγονός αυτό αποτελεί ένα συχνό φαινόμενο στα παράλληλα προγράμματα, καθώς πολλοί επιστήμονες των υπολογιστών απαντούν σε τέτοιες ερωτήσεις εξετάζοντας την επεκτασιμότητα του παράλληλου προγράμματος. Για να θυμηθούμε λίγο την επεκτασιμότητα η οποία αναλύθηκε θεωρητικά στα αρχικά κεφάλαια, χονδρικά ένα πρόγραμμα θεωρείται **επεκτάσιμο(scalable)** αν το μέγεθος του προβλήματος μπορεί να αυξάνεται κατά τέτοιο ποσοστό έτσι ώστε η αποδοτικότητα (Γράφημα **Εικόνας 5.3**) του προγράμματος να μην μειώνεται καθώς αυξάνεται το πλήθος των διεργασιών. Σύμφωνα με την ανάλυση του γραφήματος της **Εικόνας 5.3** παρατηρούμε ότι όταν το μέγεθος του προβλήματος αυξάνεται κατά το ίδιο ποσοστό με το πλήθος των διεργασιών ή πυρήνων η αποδοτικότητα τείνει να σταθεροποιηθεί κάτω από το 1 με πολύ μικρές διακυμάνσεις γύρω από το 0. Δηλαδή με την χρήση επεξεργαστών >25 παρατηρούμε ότι όσο πλησιάζουμε στους 36 η αποδοτικότητα τείνει να σταθεροποιηθεί ανεξάρτητα από το μέγεθος του προβλήματος(το τμήμα της εικόνας). Τέτοια προγράμματα που διατηρούν σταθερή αποδοτικότητα όταν το μέγεθος του προβλήματος αυξάνεται ταυτόχρονα με το πλήθος των διεργασιών όπως γίνεται και στην περίπτωση μας ονομάζονται **ασθενώς επεκτάσιμα(weakly scalable)**.

Α. Μετρήσεις MPI Εικόνα χρώματος RGB

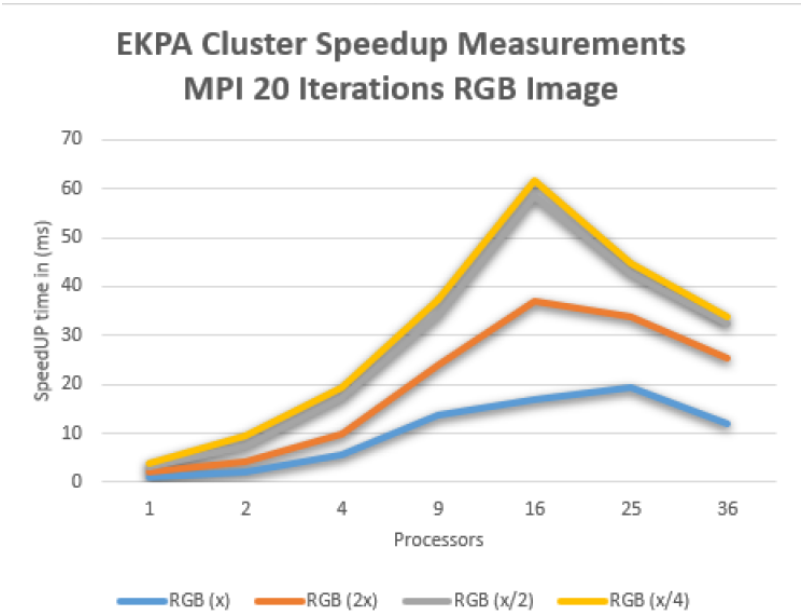
Ομοίως και για τα παρακάτω γραφήματα η αιτιολόγηση είναι ίδια με προηγουμένως καθότι το μόνο που αλλάζει είναι το χρώμα της εικόνα. Σαφώς θα υπάρχει και μια πολύ μικρή διαφορά στους χρόνους εκτέλεσης λόγο το ότι η εικόνα είναι έγχρωμη σε αντίθεση με την προηγουμένη περίπτωση που ήταν ασπρόμαυρη άλλη διαφορά αυτή είναι πολύ μικρή και θεωρείται αμελητέα. Η ουσία είναι ότι και οι δυο περιπτώσεις εξετάζονται με βάση το απλό μοντέλο (MPI), το οποίο αναλύθηκε λεπτομερώς παραπάνω για τις **επιταχύνσεις** και την **αποδοτικότητα** που πετυχαίνει.



Εικόνα 5.4: Οθόνη εκτέλεσης γραφήματος που αναπαριστά τον χρόνο εκτέλεσης επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Wall Time Measurements MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	6.35	22.03	1.71	0.44
2	3.13	10.04	0.56	0.21
4	1.15	5.15	0.23	0.2
9	0.46	2.15	0.17	0.14
16	0.38	1.08	0.08	0.13
25	0.33	1.53	0.19	0.21
36	0.53	1.63	0.25	0.31

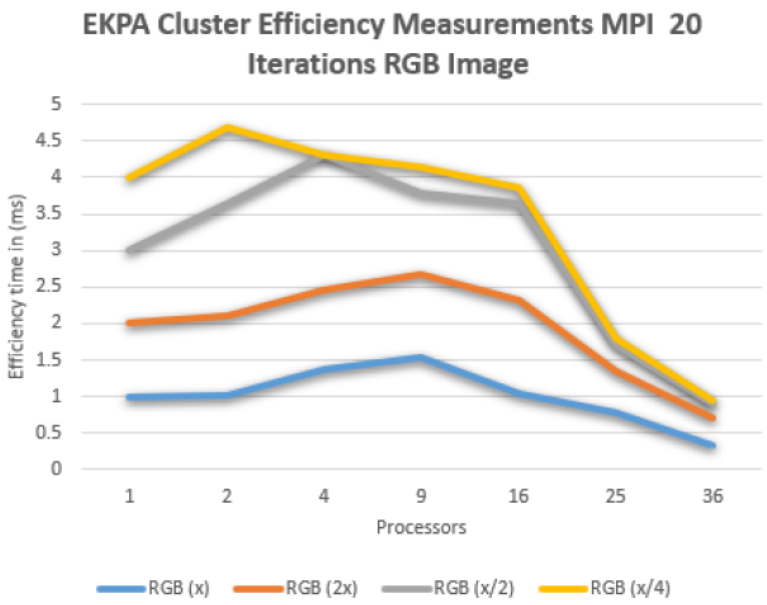
Πίνακας 5.4: Χρόνοι εκτέλεσης σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών.



Εικόνα 5.5: Οθόνη εκτέλεσης γραφήματος που αναπαριστά την επιτάχυνση(Speedup) επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster SpeedUp Measurements MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	1	1	1	1
2	2.028753994	2.194223108	3.053571429	2.095238095
4	5.52173913	4.277669903	7.434782609	2.2
9	13.80434783	10.24651163	10.05882353	3.142857143
16	16.71052632	20.39814815	21.375	3.142857143
25	19.24242424	14.39869281	9	2.095238095
36	11.98113208	13.51533742	6.84	1.419354839

Πίνακας 5.5: **Επιταχύνσεις** σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών.



Εικόνα 5.6: Οθόνη εκτέλεσης γραφήματος που αναπαριστά την αποδοτικότητα(Efficiency) επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Efficiency Measurements MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	1	1	1	1
2	1.014376997	1.097111554	1.526785714	1.047619048
4	1.380434783	1.069417476	1.858695652	0.006111111
9	1.533816425	1.138501292	1.117647059	0.349206349
16	1.044407895	1.274884259	1.3359375	0.211538462
25	0.76969697	0.575947712	0.36	0.083809524
36	0.332809224	0.37542604	0.19	0.039426523

Πίνακας 5.6: Αποδοτικότητες σειριακού και παράλληλου τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών.

Μελέτη Κλιμάκωσης-Μετρήσεις για Υβριδικό σύστημα και Σύγκριση με MPI

Υπάρχουν δύο κύριοι λόγοι για τους οποίους μια υβριδική MPI + OpenMP εφαρμογή ενός παράλληλου προγράμματος μπορεί να είναι πιο αποτελεσματική. Το πρώτο από αυτά είναι ότι μειώνει τις **απαιτήσεις μνήμης** μιας αίτησης και το δεύτερο είναι ότι

βελτιώνει την συνολικότερη **απόδοσή** του προγράμματος, όπως θα δούμε και στις μετρήσεις του υβριδικού μοντέλου. Συνοπτικά με την χρήση της υβριδικής υλοποίησης συνοψίζονται τα εξής πλεονεκτήματα:

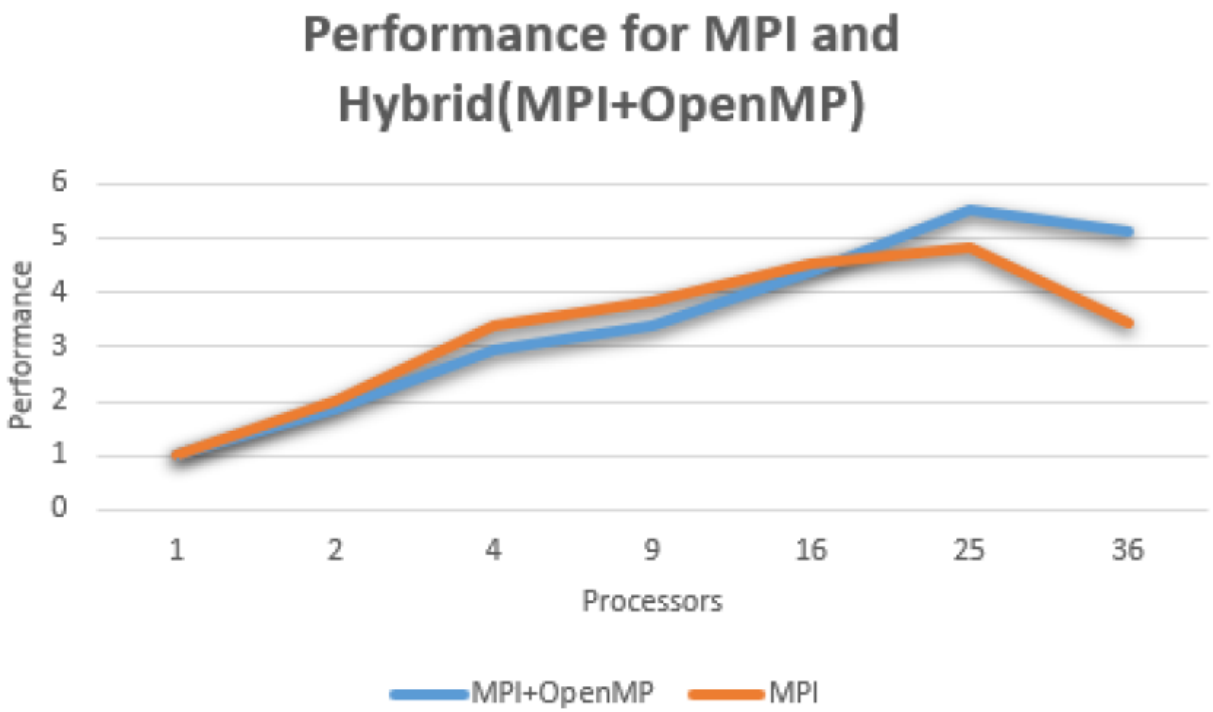
- **Ισορροπία φορτίου**: Η υβριδική προσέγγιση μειώνει τον αριθμό των απαιτούμενων διεργασιών με αποτέλεσμα να μειώνεται και η συμφόρηση του δικτύου.
- **Μείωση των μηνυμάτων**: Λόγο της βελτιωμένης απόδοσης και της μείωσης της καθυστέρησης μειώνεται ο συνολικός όγκος μεταφοράς των μηνυμάτων.
- **Αποδοτικότερη Δέσμευση Μνήμης**: Η μέθοδος του υβριδικού προγράμματος μειώνει τις απαιτήσεις της μνήμης και βελτιώνει την χρήση της κρυφής μνήμης(cache).
- **Βελτιωμένο Parallel I/0**: Επιτυγχάνεται ταχύτερος χρόνος κατά το διάβασμα/γράψιμο από την κρυφή μνήμη στον δίσκο.
- **Αποδοτικότερη συλλογική επικοινωνία**: Η μείωση του συνολικού όγκου δεδομένων των μηνυμάτων που μεταφέρονται οδηγεί στην μείωση της συλλογικής επικοινωνίας που αυτό συνεπάγεται στην μείωση της συμφόρησης του δικτύου.
- **Βελτιωμένη απόδοση**: (θα εξηγηθεί λεπτομερώς στην συνέχεια μέσα από γράφημα όπου θα συγκριθούν οι μετρήσεις για το πρόβλημα μας με χρήση τόσο του απλού μοντέλου όσο και του υβριδικού).

Βέβαια η χρήση του υβριδικού μοντέλου εκτός από τα πολλά θετικά πλεονεκτήματα που αναλύσαμε προηγουμένως έχει και μειονεκτήματα τα οποία αναλύονται ονομαστικά παρακάτω:

- **Overhead**: Δημιουργείται ένας επιπρόσθετος χρόνος κατά της διαδικασία κατασκευής η καταστροφής των νημάτων.
- **Πιο περιπλοκά προγράμματα**: Ο κώδικας γίνεται αρκετά πιο περίπλοκος και δυσνόητος γεγονός που δυσχεραίνει και την συνολική επίδοση του προγραμματιστή.
- **Περιορισμένη Παραλληλία**: Οι δυνατότητες παραλληλίας είναι περιορισμένες.
- **Υποστήριξη νημάτων**: Η χρήση τρίτων βιβλιοθηκών λογισμικού και η δημιουργία νημάτων για αυτό το σκοπό αυτό, μπορεί να δημιουργήσει συγκρούσεις με το υβριδικό μοντέλο.

Αφού τώρα έχουμε κατανοήσει πλήρως τα θετικά στοιχεία του υβριδικού μοντέλου καθώς έχουμε επισημάνει και τα μειονεκτήματα που υστερεί, θα εξετάσουμε πως κλιμακώνεται η συνολική του επίδοση εν συγκρίσει με το απλό μοντέλο (MPI). Γενικώς σε αρκετά παράλληλα προγράμματα παρουσιάζεται το πρόβλημα της κλιμάκωσης όταν ο αριθμός των διαθέσιμων πυρήνων είναι μικρός. Υπάρχει ένας συγκεκριμένος αριθμός πυρήνων (εξαρτάται βέβαια από το πρόβλημα) που μπορούμε να επιτύχουμε καλές επιδόσεις διότι έπειτα λόγω του ότι προστίθεται επιπρόσθετος χρόνος (Overhead) ο οποίος γίνεται αρκετά αισθητός, αυτό οδηγεί στην μείωση των επιδόσεων ακόμα και με μεγαλύτερο αριθμό πυρήνων. Η χρήση του υβριδικού συστήματος (MPI+OpenMP) μπορεί να

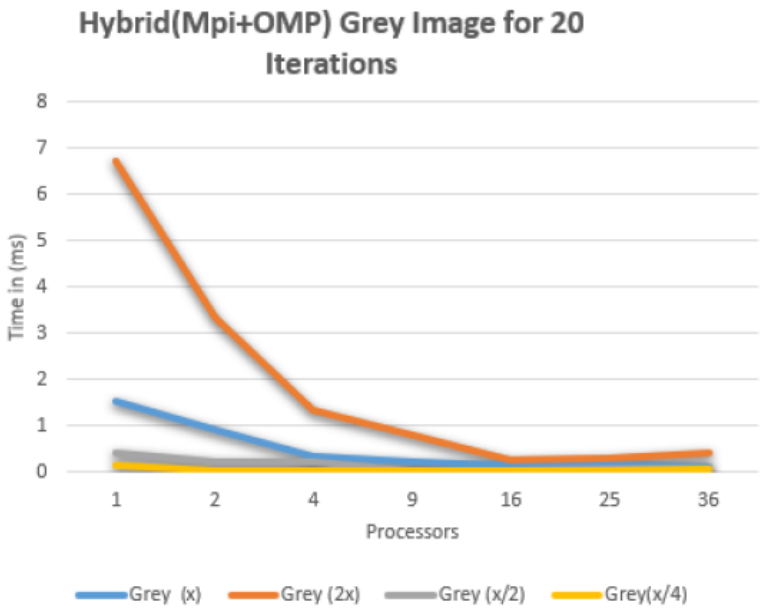
μειώσει τα Overheads που δημιουργούνται. Παρόλα αυτά, δεν σημαίνει ότι το υβριδικό μοντέλο θα κάνει πάντα outperform το απλό μοντέλο. Υπάρχουν κάποια περιπτώσεις που η χρήση νημάτων OpenMP δεν βελτιστοποιεί τον χρόνο εκτέλεσης, μπορεί να βοηθήσει όμως αισθητά στην μετέπειτα κλιμάκωση του προβλήματος.



Εικόνα 5.7: Οθόνη εκτέλεσης γραφήματος που αναπαριστά τις επιδόσεις για MPI και υβριδικά (MPI + OpenMP) συστήματα.

Ας πάρουμε για παράδειγμα το γράφημα της **Εικόνας 5.7** το οποίο αναπαριστά την επίδοση του παράλληλου προγράμματος μας για το κανονικό τμήμα της εικόνας και τον τρόπο με τον οποίο αυτό κλιμακώνεται τόσο για το απλό όσο και για το υβριδικό μοντέλο συναρτήση του αριθμού των επεξεργαστών. Στο γράφημα αυτό παρατηρούμε ότι για μικρό αριθμό πυρήνων η επίδοση του MPI κλιμακώνει καλυτέρα από το υβριδικό και αποδεικνύεται πιο αποδοτική. Βέβαια αυτό παύει να ισχύει μετά από ένα σημείο, όπου η συνολική επίδοσή του υβριδικού ξεπερνάει την επίδοση του απλού μοντέλο. Επιπροσθέτως, στο τέλος για 36 επεξεργαστές η επίδοση παρόλο που και στις δυο περιπτώσεις σε εκείνο το σημείο αρχίζει να μειώνεται, το υβριδικό αποδεικνύεται ταχύτερο και βέλτιστο σε αντίθεση με το απλό μοντέλο.

Β. Μετρήσεις MPI+ OpenMP Εικόνα χρώματος γκρι



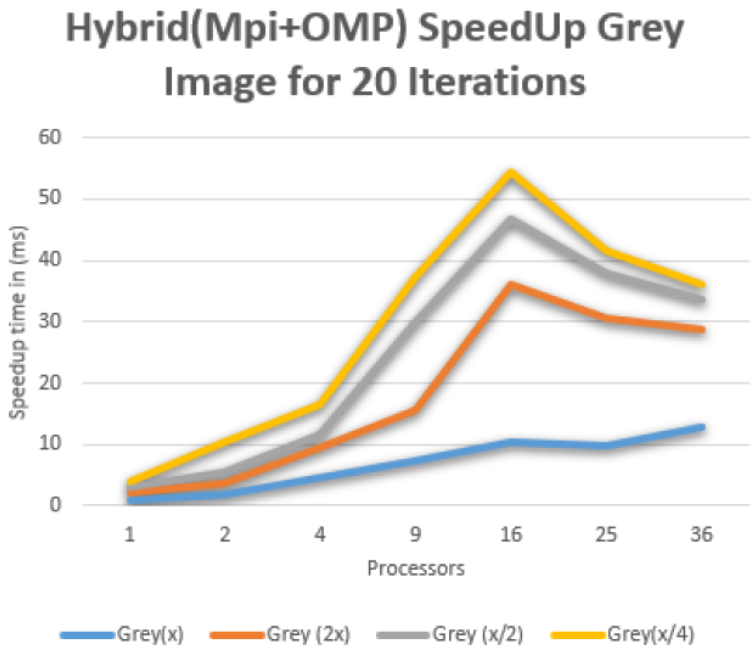
Εικόνα 5.8: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά τον χρόνο εκτέλεσης επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

	EKPA Cluster Time Measurements Hybrid MPI 20 Iterations Grey Image			
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	1.54	6.73	0.43	0.15
2	0.91	3.34	0.24	0.03
4	0.34	1.34	0.22	0.03
9	0.21	0.82	0.03	0.02
16	0.15	0.26	0.04	0.02
25	0.16	0.32	0.06	0.04
36	0.12	0.42	0.09	0.06

Πίνακας 5.8: Χρόνοι εκτέλεσης σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών.

Σύμφωνα με το γράφημα της **Εικόνας 5.8** καθώς αυξάνουμε το πλήθος των επεξεργαστών οι χρόνοι εκτέλεσης αρχικά μειώνονται. Όμως μετά από κάποιο σημείο οι χρόνοι εκτέλεσης παρατηρούμε ότι αρχίζουν να χειροτερεύουν αντί να βελτιώνονται σύμφωνα

από τον **Πίνακα 5.8** όπως για παράδειγμα φαίνεται με την χρήση 25, 36 επεξεργαστών αντίστοιχα όπου οι χρόνοι εκεί χειροτερεύουν αντί να καλυτερεύουν.



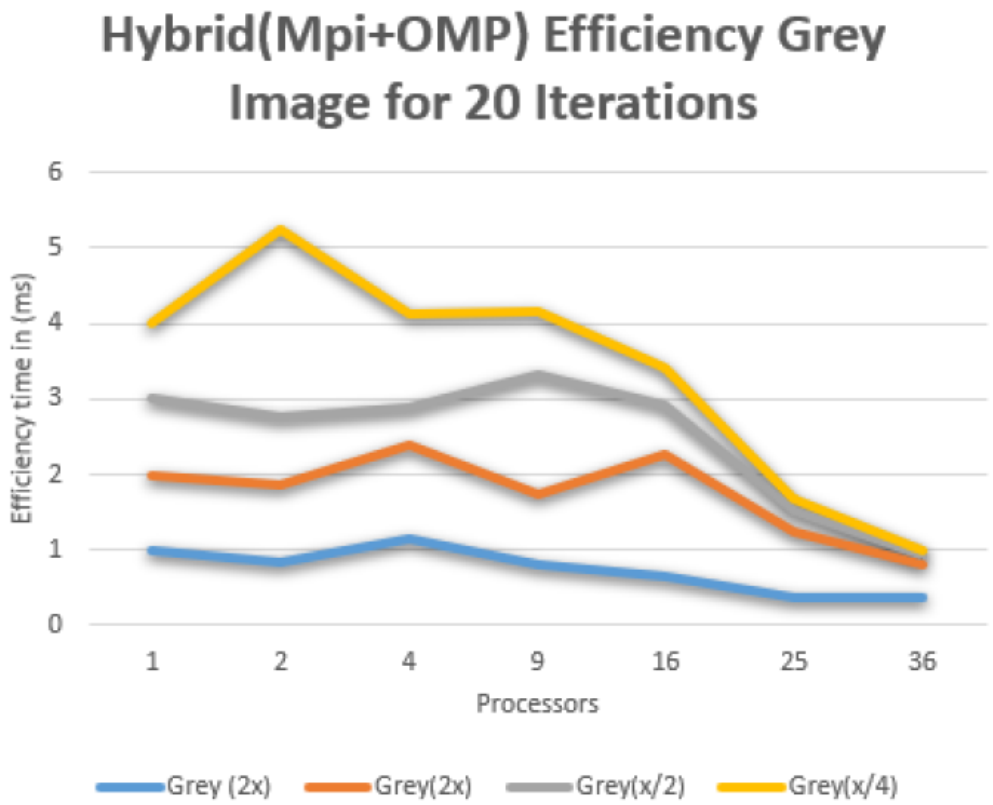
Εικόνα 5.9: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά την **επιτάχυνση(Speedup)** επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster SpeedUp Measurements Hybrid MPI 20 Iterations Grey Image				
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	1	1	1	1
2	1.692307692	2.01497006	1.791666667	5
4	4.529411765	5.02238806	1.954545455	5
9	7.333333333	8.207317073	14.33333333	7.5
16	10.26666667	25.88461538	10.75	7.5
25	9.625	21.03125	7.166666667	3.75
36	12.83333333	16.02380952	4.777777778	2.5

Πίνακας 5.9: **Επιταχύνσεις** σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών.

Στην **Εικόνα 5.9** παρουσιάζεται η επιτάχυνση κατάλληλου τμήματος εικόνας συναρτήση του αριθμού των επεξεργαστών. Παρατηρούμε ότι η επιτάχυνση αυξάνεται σταδιακά έως ότου και την χρήση 16 διεργασιών, στην συνέχεια όμως με την χρήση 25,36 διεργασιών παρατηρούμε ότι η επιτάχυνση μειώνεται σταδιακά. Για μεγάλο αριθμό

διεργασιών και σχετικά μικρό τμήμα διαμέρισης της εικόνας η επιτάχυνση ελαττώνεται σημαντικά. Παρόλο την μείωση που έχει υποστεί επιτυγχάνει να κλιμακώνει πολύ καλυτέρα από τις αντίστοιχες μετρήσεις που παρουσιάστηκαν για την επιτάχυνση στο απλό μοντέλο.



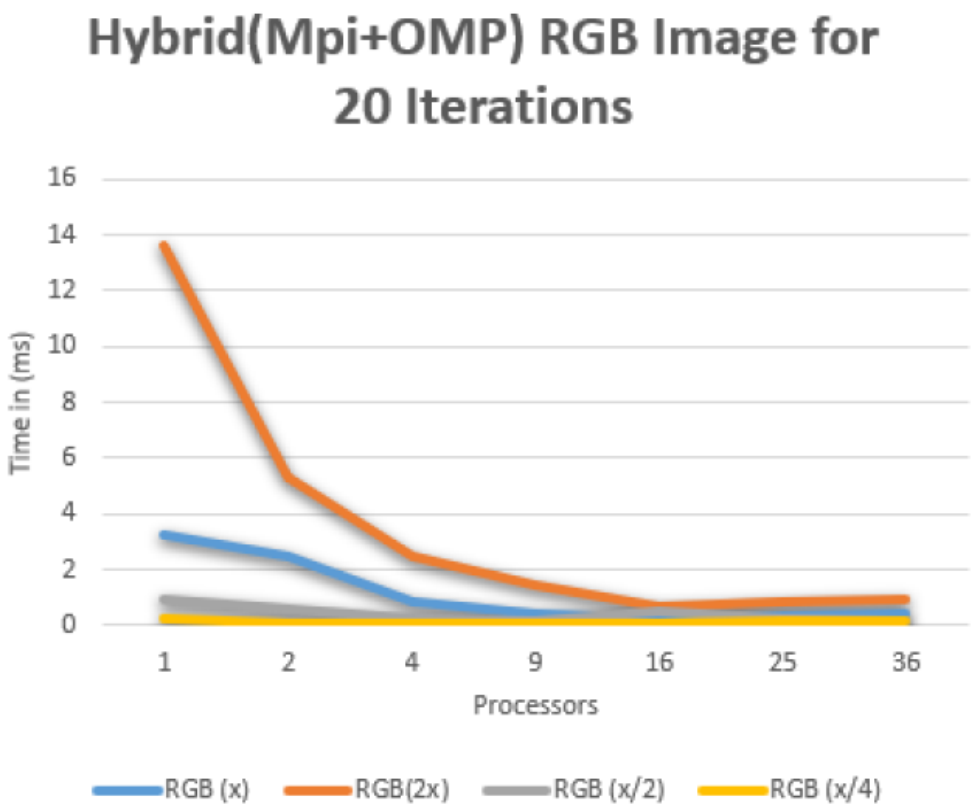
Εικόνα 5.10: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά την **αποδοτικότητα(Efficiency)** επιλεγμένου τμήματος για τη γκρι εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Efficiency Measurements Hybrid MPI 20 Iterations Grey Image				
Processors	Grey 1920*2520 (x)	Grey 1920*5040 (2x)	Grey 1920*1260 (x/2)	Grey 1920*630 (x/4)
1	1	1	1	1
2	0.846153846	1.00748503	0.895833333	2.5
4	1.132352941	1.255597015	0.488636364	1.25
9	0.814814815	0.911924119	1.592592593	0.833333333
16	0.641666667	1.617788462	0.671875	0.46875
25	0.385	0.84125	0.286666667	0.15
36	0.356481481	0.44510582	0.132716049	0.069444444

Πίνακας 5.10: Αποδοτικότητες σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της γκρι εικόνας δοθέντος αριθμού επεξεργαστών

Σύμφωνα με την ανάλυση του γραφήματος της **Εικόνας 5.10** παρατηρούμε ότι όταν το μέγεθος του προβλήματος αυξάνεται κατά το ίδιο ποσοστό με το πλήθος των διεργασιών ή πυρήνων η αποδοτικότητα τείνει να σταθεροποιηθεί κοντά στο 1 με πολύ μικρές διακυμάνσεις. Τέτοια προγράμματα όπως είδαμε και προηγουμένως που διατηρούν σταθερή αποδοτικότητα όταν το μέγεθος του προβλήματος αυξάνεται ταυτόχρονα με το πλήθος των διεργασιών (όπως γίνεται και στην περίπτωση μας) ονομάζονται **ασθενώς επεκτάσιμα(weakly scalable)**.

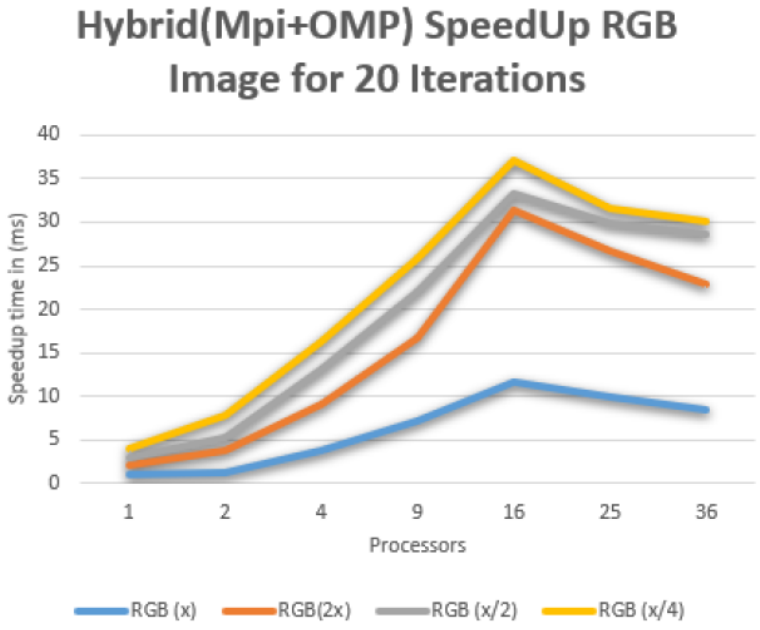
Β. Μετρήσεις MPI+ OpenMP Εικόνα χρώματος RGB



Εικόνα 5.11: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά τον χρόνο εκτέλεσης επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Time Measurements Hybrid MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	3.27	13.62	0.93	0.22
2	2.52	5.34	0.62	0.09
4	0.87	2.51	0.23	0.07
9	0.46	1.43	0.17	0.06
16	0.28	0.69	0.48	0.06
25	0.33	0.81	0.29	0.13
36	0.39	0.94	0.16	0.15

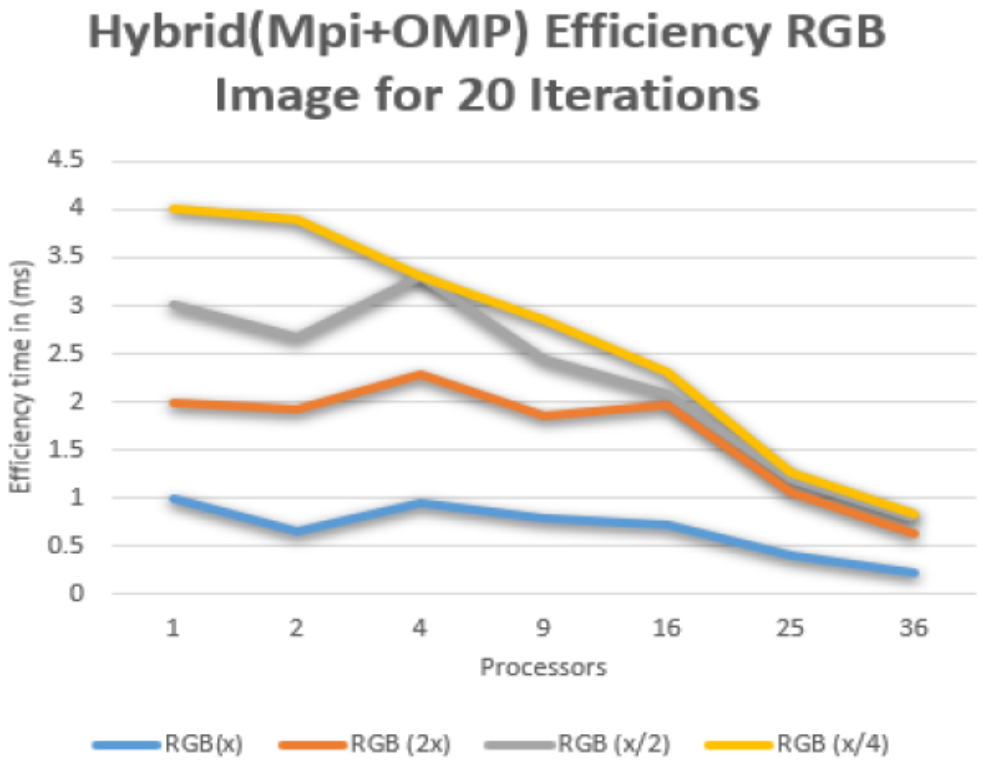
Πίνακας 5.11: Χρόνοι εκτέλεσης σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών Εικόνα.



Εικόνα 5.12: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά την **επιτάχυνση(Speedup)** επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster SpeedUp Measurements Hybrid MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	1	1	1	1
2	1.297619048	2.550561798	1.5	2.444444444
4	3.75862069	5.426294821	4.043478261	3.142857143
9	7.108695652	9.524475524	5.470588235	3.666666667
16	11.67857143	19.73913043	1.9375	3.666666667
25	9.909090909	16.81481481	3.206896552	1.692307692
36	8.384615385	14.4893617	5.8125	1.466666667

Πίνακας 5.12: Επιταχύνσεις σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών.



Εικόνα 5.13: Οθόνη εκτέλεσης υβριδικού γραφήματος που αναπαριστά την **αποδοτικότητα(Efficiency)** επιλεγμένου τμήματος για τη RGB εικόνα συναρτήση με τον αριθμό των επεξεργαστών.

EKPA Cluster Efficiency Measurements Hybrid MPI 20 Iterations RGB Image				
Processors	RGB 1920*2520 (x)	RGB 1920*5040 (2x)	RGB 1920*1260 (x/2)	RGB 1920*630 (x/4)
1	1	1	1	1
2	0.648809524	1.275280899	0.75	1.222222222
4	0.939655172	1.356573705	1.010869565	0.003055556
9	0.789855072	1.058275058	0.607843137	0.407407407
16	0.729910714	1.233695652	0.12109375	0.229166667
25	0.396363636	0.672592593	0.128275862	0.067692308
36	0.232905983	0.40248227	0.161458333	0.040740741

Πίνακας 5.13: Αποδοτικότητες σειριακού και παράλληλου υβριδικού τμήματος για το αντίστοιχο τμήμα της RGB εικόνας δοθέντος αριθμού επεξεργαστών.

6. Συμπεράσματα

Συνοψίζοντας όλη την μελέτη που πραγματοποιήθηκε σε αυτή την εργασία μπορούμε να καταλήξουμε στο εύλογο συμπέρασμα ότι, η χρήση του υβριδικού συστήματος κατά τον σχεδιασμό παράλληλων προγραμμάτων έχει αρκετά θετικά στοιχεία και γενικότερα υπερισχύει αλλά αυτό δεν σημαίνει ότι θα πρέπει να χρησιμοποιείται σε όλες τις περιπτώσεις καθώς δεν προσφέρει πάντα την καλύτερη λύση. Δηλαδή θα πρέπει να εξετάζεται η κάθε περίπτωση αναλόγως το δοθέν πρόβλημα. Επιπλέον, θα πρέπει εφαρμοστούν μια σειρά από κανόνες τα οποία να προσδιορίζουν ποιο μοντέλο είναι καλύτερο για υιοθέτηση στο εκάστοτε πρόβλημα. Παρακάτω ακολουθούν κάποια συμπεράσματα που θα πρέπει να εφαρμόζουμε, έτσι ώστε να οδηγηθούμε στο ποιο αποτελεσματικό μοντέλο κατά την διαδικασία της σχεδίασης . Για παράδειγμα τίθεται το εξής ερώτημα: Είναι το απλό μοντέλο (MPI) ή το υβριδικό μοντέλο (MPI+OpenMp) αποδοτικότερο και πως θα εξετάζεται σε κάθε περίσταση; Υπάρχουν εναλλακτικές τεχνικές να ενσωματώσουμε για να επωφεληθούμε;

Συνοπτικά θα πρέπει:

- Κάθε προγραμματιστής να θέτει σαφή κίνητρα: είναι η μείωση των απαιτήσεων μνήμης αυτό που επιθυμούμε ή η βελτίωση της απόδοσης (ή και τα δυο)?
- Εάν το κύριο κίνητρο είναι να μειώσουμε τις απαιτήσεις μνήμης, πρέπει να εκτιμηθεί αν θα υπάρχει πιθανή εξοικονόμηση από τη μείωση της μεταφοράς δεδομένων.
- Εάν το κύριο κίνητρο είναι η απόδοση, πρέπει να βεβαιωθούμε ότι έχουμε κατανόηση πλήρως να αξιολογούμε τα πιθανά σημεία συμφόρησης στον κώδικα MPI (π.χ. είναι η έλλειψη επεκτασιμότητας ή έλλειψη ανεπαρκούς παραλληλισμού ή επικοινωνίας) και να αναλογιστούμε αν η υβριδική έκδοση θα μπορούσε να βοηθήσει για τη λύση των παραπάνω προβλημάτων.
- Εξετάζουμε αρχικά την εφαρμογή της τεχνική Master-Only σαν αρχική εφαρμογή

και μεταφέρουμε αργότερα (πιθανά μόνο τμήματα του κώδικα) σε άλλα μοντέλο, εάν η απόδοση είναι ανεπαρκής

- Αν επιθυμούμε να εκτελέσουμε τον υβριδικό κώδικα σε συστήματα με NUMA κόμβους, θα πρέπει να εξετάσουμε την αρχικοποίηση μεγάλων δομών δεδομένων κατά τη διαδικασία του παραλληλισμού με την χρήση του OpenMP, για να επωφεληθούμε από την πολιτική κατανομής.
- Εάν τρέχουμε ένα εκτελέσιμο κώδικα σε σύστημα με NUMA κόμβους, πρέπει να ελέγξουμε ότι, στο σύστημα το MPI έχει ρυθμιστεί σωστά έτσι ώστε να τοποθετηθούν και να δεσμευτούν λογικά και οι αντίστοιχες διαδικασίες και τα νήματα που ορίζονται.