ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ : ΕΡΓΑΣΙΑ ΑΚΑΔΗΜΑΪΚΟΥ ΕΤΟΥΣ 2017-2018

ΦΟΙΤΗΤΗΣ : ΣΙΜΟΥ ΟΔΥΣΣΕΑΣ

ΑΜ : 1115200800290

ΗΜΕΡΟΜΗΝΙΑ : 10/2018

1. Εισαγωγή

Σκοπός της εργασίας / Σύντομη περίληψη

Σκοπός της εργασίας ήταν να δημιουργηθούν προγράμματα σε γλώσσα C που θα παραλληλοποιούν την διεργασία συνέλιξης οποιασδήποτε εικόνας ενός από δύο τύπους (ασπρόμαυρη/grey και έγχρωμη/rgb), με ένα φίλτρο 3x3 pixel. Πρακτικά δηλαδή στον υπολογισμό της συνέλιξης κάθε pixel εμπλέκονται τα περιφερειακά του σε απόσταση ένα. Τα παράλληλα προγράμματα που αναπτύχθηκαν χρησιμοποιούν :

- Τη βιβλιοθήκη MPI (Message Passing Interface) για συστήματα κατανεμημένης μνήμης, όπου πρακτικά τα processes που δημιουργούνται έχουν τη δική τους μνήμη και επικοινωνούν μέσω μηνυμάτων.
- Από κοινού τη βιβλιοθήκη MPI καθώς και την OpenMP, με την τελευταία να αφορά συστήματα κοινής μνήμης όπου δημιουργούνται threads με πρόσβαση σε όλη τη μνήμη κάθε process.
- *Ο σχεδιασμός προγράμματος σε Cuda δεν έγινε λόγω μη ύπαρξης κάρτας γραφικών για προγραμματισμό και εκτέλεση*

Κάθε ένα από τα προγράμματα που αναπτύχθηκαν δοκιμάστηκε στις εικόνες που δόθηκαν καθώς και σε υποπολλαπλάσια και πολλαπλάσια αυτών. Λόγω προβλήματος χώρου στο χώρο των users των Linux η rgb εικόνα δοκιμάστηκε μόνο μέχρι διπλάσιο μέγεθος και η grey μέχρι τετραπλάσιο. Η εκτέλεση έγινε μέσω ssh στα Linux της σχολής. Κατά τη διάρκεια των δοκιμών μόνο οι 29 από τις 30 μηχανές ήταν διαθέσιμες (εκτός η linux21).

Μεταγλώττιση και εκτέλεση

Η μεταγλώττιση έγινε μέσω scripts tcsh τα οποία έκαναν compile και execute για δοθέν : α) σύνολο αρχείων εικόνων , β) σύνολο αριθμών επαναλήψεων για έλεγχο σύγκλισης (π.χ. convIters = 30 , 2…) , γ) σύνολο επιθυμητών processes (π.χ np = 1, 2 , 4, 8 …) και δ) σύνολο μηχανών linux που θα εκτελούνταν (μέσω αρχείου machines όπου δίνονταν ο αριθμός διεργασιών που θα εκτελεστούν σε κάθε μηχανή). Η εκτέλεση γίνεται 3 φορές για κάθε μεμονωμένο σετ του παραπάνω καρτεσιανού γινομένου ώστε να μην υπάρχει περίπτωση σφάλματος λόγω τυχαίας καθυστέρησης. Ως χρόνος run time λήφθηκε ο μικρότερος από τις 3 εκτελέσεις κάθε περίπτωσης.

Φυσικά για τις ανάγκες μεταγλώττισης και εκτέλεσης του προγράμματος μία φορά, αρκεί ένα make file ή η μεταγλώττιση και η εκτέλεση μέσω εντολών στη γραμμή εντολών. Για παράδειγμα :

Compilation

MPI : mpicc –g –o mpi_conv mpi_image_conv.c –lm

OpenMP+MPI : mpicc –g –fopenmp opmp_mpi_conv op_mpi_image.c -lm

Execution

MPI :

mpiexec –f machines –np 8 ./mpi_conv waterfall_1920_2520.raw 2520 1920 30 2 grey

OpenMP+MPI :

mpiexec –f ompmachines –np 8 ./opmp_mpi_conv waterfall_1920_2520.raw 2520 1920 30 2 grey

Η εντολή εκτέλεσης είναι ίδια και στα δύο προγράμματα (αλλάζει μόνο το εκτελέσιμο και το αρχείο machines)

Πρόσθετα

Στην υλοποίηση MPI και OpenMP+MPI συμπεριλήφθηκε true-parallel IO level 3, δηλαδή collective , noncontiguous requests με χρήση derived datatypes κάτι το οποίο επιτάχυνε το χρόνο ανάγνωσης και εγγραφής των εικόνων δραστικά. Να σημειωθεί ότι αρχικά (δεν θα παρατεθεί) δοκιμάστηκε και χρησιμοποιήθηκε parallel IO level 1, ορίζοντας για κάθε process μεμονωμένο δείκτη αρχείου που μετατοπιζόταν με seek_set για ανάγνωση των δεδομένων αποκλειστικά του process, κάτι το οποίο είχε ως αποτέλεσμα 6-10 φορές μεγαλύτερο χρόνο ανάγνωσης και εγγραφής. Έτσι προχώρησα στη λύση του level 3.

2. Σχεδιασμός προγραμμάτων

Γενικά

Προκειμένου να σχεδιαστεί ο τρόπος παραλληλοποίησης του προγράμματος υλοποίησης της συνέλιξης χρησιμοποιήθηκε η μεθοδολογία του Foster που χωρίζει τον σχεδιασμό σε τέσσερα βήματα :

a) Διαμέριση του προβλήματος σε tasks
b) Αναγνώριση των αναγκών επικοινωνίας μεταξύ των tasks
c) Σύνθεση των tasks σε μεγαλύτερα (αν είναι προφανές ότι απαιτείται το προηγούμενο για να γίνει το επόμενο)
d) Αντιστοίχιση των σύνθετων tasks στις διεργασίες/πυρήνες

Τα παραπάνω με την προϋπόθεση ότι ελαχιστοποιείται η επικοινωνία (που είναι και η πιο κοστοβόρα για το MPI) μεταξύ των διεργασιών και ότι υπάρχει load balancing στην κατανομή των tasks σε κάθε διεργασία.

Διαμερισμός σε tasks

Στη συγκεκριμένη περίπτωση ο κύριος υπολογισμός είναι η συνέλιξη σε κάθε pixel. Επομένως ήταν φυσικό ουσιαστικά να κάνουμε data partitioning αφού ο υπολογισμός είναι ενός τύπου

(για τον τύπο εικόνας που επιλέγεται). Έτσι διαμερίζουμε την εικόνα σε blocks και αναθέτουμε σε κάθε διεργασία ένα «συμπαγές» τμήμα της να επεξεργαστεί (να υπολογίσει δηλαδή τη συνέλιξη των pixel που την απαρτίζουν). Με τον όρο «συμπαγές» εννοούμε ότι τα pixel κάθε υπο-εικόνας γειτνιάζουν φυσικά έτσι ώστε τα pixel που απαιτούνται για τον υπολογισμό της συνέλιξης κάθε pixel να είναι κατά το μέγιστο βαθμό στην ίδια διεργασία και να λείπουν μονάχα τα περιφερειακά της υπο-εικόνας.

Προκειμένου να υπάρχει load balancing επιλέχθηκε κάθε block να έχει ίδιο αριθμό pixels και να μην γίνεται split με τρόπου που δεν ικανοποιεί τη συνθήκη αυτή.

Ο χρήστης δίνει ως όρισμα του/των προγραμμάτων MPI και OpenMP+MPI τον αριθμό διεργασιών που επιθυμεί και εφόσον ικανοποιείται η παραπάνω συνθήκη γίνεται ο βέλτιστος διαμερισμός (βλ. πιο κάτω).

Αναγνώριση αναγκών επικοινωνίας

Δεδομένου ότι κάθε διεργασία λαμβάνει ένα «συμπαγές» τμήμα της εικόνας ελαχιστοποιείται για την υπο-εικόνα η ανάγκη επικοινωνίας για αίτηση γειτονικών pixel που συμμετέχουν στους υπολογισμούς. Οι ανάγκες επικοινωνίας που απομένουν είναι τα pixel που βρίσκονται στο πλαίσιο/frame της υπο-εικόνας και των γειτονικών της (North, South, West, East, NorthWest, NorthEast, SouthWest, SouthEast).

Τα pixel του frame πρέπει να σταλούν από τις γείτονες διεργασίες. Για να ελαχιστοποιηθεί ο αριθμός των pixel των πλαισίων που εμπλέκονται στις επικοινωνίες πρέπει να βρεθεί για δεδομένο αριθμό διεργασιών np ο καλύτερος διαμοιρασμός row & column splits. Αποδεικνύεται μαθηματικά (με επίλυση της πρώτης παραγώγου ίσης με μηδέν και έλεγχο της τιμής της δεύτερης παραγώγου) για την περίπτωση συνεχών μεταβλητών ότι ο αν nc, nr, np, x, y ο αριθμός των στηλών, γραμμών, διεργασιών, row splits, column splits τότε ο αριθμός y column splits που ελαχιστοποιεί το άθροισμα των περιμέτρων των frames που δημιουργούνται είναι :

$$y = \sqrt{\frac{nc\ np}{nr}}$$

οπότε ο αριθμός των row splits x προκύπτει ως x = np/y.

Αν και υλοποιήθηκε αρχικά συνάρτηση για την εύρεση των βέλτιστων αυτών y,x (commented στον κώδικα ως THEORETICAL APPROACH), η επίλυση αυτή προσκρούει στο γεγονός ότι στην περίπτωσή μας οι x,y παίρνουν διακριτές τιμές. Δοκιμάστηκε αφού βρίσκονται τα βέλτιστα x,y να εξετάζεται η βέλτιστη περίμετρος που προκύπτει από τα ceil , floor τους αλλά και πάλι η προσέγγιση αυτή κάνει πολύ narrow την αναζήτηση και οδηγεί σε ατελέσφορες (non load balanced) τιμές. Εν τέλει η συνάρτηση ***column_splits*** κάνει εξοντωτική έρευνα των πιθανών συνδυασμών που ικανοποιούν το load balancing κριτήριο και επιλέγει αυτή με την καλύτερη περίμετρο. Αυτό δεν επηρεάζει σημαντικά τον χρόνο εκτέλεσης αφού οι διεργασίες δεν είναι περισσότερες από 120.

Σύνθεση tasks σε μεγαλύτερα

Εδώ η σύνθεση ουσιαστικά δεν απαιτείται καθώς το έργο υπολογισμού είναι ένα. Επιλέχθηκε όμως να μην παραλληλοποιούνται τα ενδιάμεσα loop υπολογισμού της συνέλιξης κάθε pixel καθώς πρόκειται για υπολογισμό μικρής έκτασης. Στην περίπτωση του OpenMP γίνεται παραλληλοποίηση των εσωτερικών loop υπολογισμών συνέλιξης όλων των pixel μίας υποεικόνας, ενώ στο απλό MPI το χειριζόμαστε ως ένα task.

Αντιστοίχιση σε διεργασίες

Η αντιστοίχιση των υπο-εικόνων στις διεργασίες έγινε με αρίθμηση που εκκινούσε από το κάτω αριστερά τμήμα της εικόνας , κινείτο οριζόντια δεξιά μέχρι το δεξί άκρο της και μετά ανέβαινε στο ακριβώς από πάνω αριστερό τμήμα πάλι, για να συνεχίσει με τον ίδιο τρόπο. Πρακτικά ακολουθήθηκε η δομή μίας δι-διάστατης array όπου π.χ. η πρώτη γραμμή περιέχει τα στοιχεία (0 1 2 3), η επόμενη από πάνω τα (4 5 6 7) κ.ο.κ. . Η αναγνώριση των γειτονικών διεργασιών στο παραπάνω σύστημα γίνεται με τον καθορισμό των γειτόνων (N, S, W, E, NW, NE, SW, SE) στη συνάρτηση ***establish_neighbors*** που επιστρέφει για κάθε διεργασία τον αριθμό της γειτονικής διεργασίας σε κάθε σημείο του ορίζοντα και -1 αν δεν υπάρχει γείτονας.

3. Υλοποίηση

Γενική δομή κώδικα (MPI & OpenMP/MPI)

Στη γενική μορφή περιλαμβάνει :

1. ***Get parameters*** *(by Process 0)*

2. ***Calculate best split*** *(row/columns)*

3. ***Broadcast parameters*** *to processes*

4. ***Define vector datatypes*** *(row, column, corner)*

5. ***Read sub-image*** *to sub-array for each process* ***using parallel IO (level 3)***

6. ***Initializations : Define starting array positions*** *for every neighboring edge/corner (send/receive)*

7. ***While (currentIteration < maxIterations)*** *{*

ISend neighboring edges*/corners (when applicable)*

IReceive neighboring edges*/corners (when applicable)*

Calculate convolution *for all not in edges elements* ***(inner)***

While (not every edge/corner has been received) *{*

Check with MPI_Test *(immediate return)* ***all edges to be reveived***

If any received calculate edge

}

Wait (blocking) that all edges have been sent

If (currentIteration % convIters == 0) {

check for convergence by MPI_Reduce (changedImageBit)

}

}

8. ***Write full image using parallel IO (level 3)***

Σημείωση :

Για το OpenMP ουσιαστικά η μόνη διαφοροποίηση είναι ότι στη συνάρτηση ***calculate_convolution*** που υπολογίζει τη συνέλιξη για εύρος γραμμών και στηλών που της δίνεται, η εκτέλεση των υπολογισμών αυτών γίνεται παράλληλα με threads. **Η τιμή των threads δίνεται hard-coded στην main συνάρτηση ίση με 4.**

Περιγραφή MPI

Οι βασικές μεταβλητές image name, number of processes, συνολικός αριθμός columns/rows (pixels), max iterations, convergence check iterations, image type διαβάζονται από τη γραμμή εντολών από την master διεργασία μηδέν και, μετά από έλεγχο ότι το αρχείο εικόνας

υφίσταται και ότι είναι εφικτό το split διατηρώντας load balancing, υπολογίζονται τα row/column splits. Στη συνέχεια όλες οι προηγούμενες μεταβλητές γίνονται broadcast σε όλες τις διεργασίες.

Για την αποθήκευση των υπο-εικόνων χρησιμοποιείται μονοδιάστατη array με πλήθος στοιχείων ανάλογο με τον αριθμό γραμμών, στηλών και τύπο εικόνας (1 ή 3 array elements για εικόνα grey / rgb αντίστοιχα). Προκειμένου κάθε διεργασία να λαμβάνει από τις γειτονικές της τα απαιτούμενα pixel για τους υπολογισμούς στο πλαίσιό της, η array συμπεριέλαβε και μία «άλω» (halo ή ghost άρεα όπως αναφέρεται βιβλιογραφικά), ώστε στα στοιχεία αυτά να αποθηκεύονται τα λαμβανόμενα pixel. Οι διεργασίες που δεν είχαν γείτονες σε κάποια κατεύθυνση έκαναν ανάθεση στα pixel της halo των τιμών των pixel στο πλαίσιό τους μετά από κάθε επανάληψη.

Για την αποφυγή αντιγραφών γίνεται χρήση vector datatypes : imgRowType, imgColType, imgCornerType. Ουσιαστικά αναθέτουμε non-contiguous περιοχές για να λάβουμε γραμμή/στήλη λαμβάνοντας υπόψιν την halo καθώς και τα ενδιάμεσα elements που αντιστοιχούν σε άλλη διεργασία. Το imgCornerType είναι τετριμένη περίπτωση vector καθώς αποτελείται από ένα μόνο pixel (ή τρία στην περίπτωση rgb), αλλά χρησιμοποιήθηκε για να υπάρχει ενοποιημένη προσέγγιση. Με τη χρήση των vector datatypes γίνεται εφικτό στις send/receive επικοινωνίες να δίνεται ένας pointer στην αρχική θέση της global array και να λαμβάνεται μόνο μία γραμμή/στήλη/στοιχείο χωρίς στοιχεία άλλων διεργασιών και halos.

Για τον υπολογισμό της συνέλιξης γίνεται χρήση δύο array υπο-εικόνων σε κάθε διεργασία. Η πρώτη κρατά τα τρέχοντα στοιχεία και η δεύτερη αυτά που υπολογίζονται. Για αποφυγή copying οι pointer τους εναλλάσονται στο τέλος κάθε iteration.

Το φίλτρο δίνεται hard-coded ως ένας 3x3 float πίνακας μέσα στον κώδικα αν και θα μπορούσε να είναι παράμετρος στη γραμμή εντολών.

Η MPI επικοινωνία επιλέχθηκε να είναι immediate ώστε να αποφεύγονται αναμονές. Έτσι ουσιαστικά κάθε διεργασία αφού «διαβάσει» στην array της την υπο-εικόνα που της αντιστοιχεί, κάνει Isend, Irecv πρώτα όλων των περιφερειακών της στοιχείων (όπου εφαρμόζεται) και μετά προχωράει μέσω της calculate_convolution στον υπολογισμό της συνέλιξης όλων των ενδιάμεσων στοιχείων της πλην των περιφερειακών. Έτσι δίνει χρόνο στις γειτονικές διεργασίες να στείλουν τα απαιτούμενα στοιχεία.

Στη συνέχεια με ένα βρόχο while γίνεται συνεχής immediate έλεγχος με MPI_Test όλων των αναμενόμενων από τη διεργασία receive. Αν κάποιο από αυτά έχει ολοκληρωθεί τότε η γίνεται υπολογισμός της συνέλιξης για τη γραμμή/στήλη/γωνιακό στοιχείο που λήφθηκε και μετά ο έλεγχος των υπολοίπων συνεχίζεται. Επίσης αν δεν υπάρχει γείτονας γίνεται άμεσα ο υπολογισμός της συνέλιξης και assignment των pixel πλαισίου στα pixel της halo ώστε να είναι διαθέσιμα για τον επόμενο iteration.

Εφόσον όλα έχουν ληφθεί και γίνει οι υπολογισμοί γίνεται blocking Wait για να διαπιστωθεί αν έχουν σταλεί όλα τα στοιχεία στους γείτονες. Όταν έχουν σταλεί όλα προχωράμε στο επόμενο iteration.

Ανάλογα με τον αριθμό των convergence iterations γίνεται έλεγχος με MPI_Reduce σε μία μεταβλητή ***fullImageChanged*** που λαμβάνεται από κάθε διεργασία. Αυτή έχει τιμή 1 αν έστω και ένα pixel έχει αλλάξει και 0 αν όχι. Στο reduce επιλέγεται η max τιμή και έτσι καθίσταται γνωστό αν η εικόνα παρέμεινε ίδια ή όχι. Να σημειωθεί ότι ο έλεγχος αλλαγής του κάθε pixel κατά τη συνέλιξη κάθε iteration γίνεται κατά τον υπολογισμό της συνέλιξης, αποφεύγοντας έτσι να απαιτείται να ληφθεί η συνολική εικόνα στο reduce και να γίνει έλεγχος τότε (κάτι που θα εισήγαγε μεγάλο κόστος).

Parallel IO – level 3

Για την υλοποίηση της παράλληλης ανάγνωσης/εγγραφής από/στο αρχείο εικόνας χρησιμοποιήθηκαν οι συναρτήσεις MPI_File_set_view και MPI_Type_create_subarray. Ουσιαστικά ορίστηκαν αρχικά δύο νέα derived types : α) filetype που προσδιόριζε πως κάθε διεργασία βλέπει την εικόνα και β) memtype που προσδιόριζε πως εμφανίζεται η εικόνα στη μνήμη.

Πρακτικά η create_subarray εισήγαγε αρχικά ένα filetype που όριζε το «τμήμα» της εικόνας το οποίο μπορούσε να δει κάθε process ως ένα ενιαίο block. Αυτό έγινε θεωρώντας την global array της εικόνας ως μία δι-διάστατη array με rows/columns όσα και τα row/column splits και μοιράζοντας την στα processes ανάλογα με τα global start_indices και local sizes. Στη συνέχεια η συνάρτηση File_set_view καθόρισε μέσω αυτού του τύπου το πώς το process βλέπει το αρχείο, δηλαδή ουσιαστικά σαν ένα block localRows x localColumns.

Στη συνέχεια ορίστηκε ένας επαυξημένος τύπος memtype , πάλι μέσω της create_subarray, που βασιζόταν στον filetype αλλά περιείχε την halo που απαιτείτο από τα processes.

Έτσι πλέον το πώς διαβαζόταν το αρχείο στη μνήμη από κάθε process ήταν στον πλήρη έλεγχο του MPI που έχοντας όλη την πληροφορία των ζητούμενων κάθε φορά μπορούσε να αριστοποιήσει την επικοινωνία της ανάγνωσης. Το αντίστοιχο ίσχυε και για την εγγραφή.

Περιγραφή OpenMP + MPI

Για το OpenMP ουσιαστικά η μόνη διαφοροποίηση από τα προηγούμενα είναι ότι στη συνάρτηση ***calculate_convolution*** που υπολογίζει τη συνέλιξη για εύρος γραμμών και στηλών που της δίνεται (είτε κύρια εσωτερικά pixel είτε στο πλαίσιο), γίνεται χρήση ***parallel for*** με ***collapse*** των δύο εσωτερικών loops (γραμμής, στήλης) και εκτελούν τους υπολογισμούς 4 νήματα (hard coded). Οι μεταβλητές που είναι shared, private έχουν οριστεί ρητώς στην περίπτωσή αυτή. Επιλέχθηκε ***schedule (static,1)*** που έδινε καλύτερους χρόνους/αποτελέσματα και ουσιαστικά μοιράζει στατικά τα νήματα, ενώ τα chunks του for loop μοιράζονται όσο το δυνατόν πιο καλά.

4. Εκτελέσεις – Μετρήσεις Απόδοσης / Επιτάχυνσης – Αποτελέσματα

- Η απόδοση των προγραμμάτων έγινε και με τις δύο εικόνες grey/rgb, για διάφορα μεγέθη και με διαφορετικό πλήθος διεργασιών. Λόγω του περιορισμένου user space στα linux δεν έγιναν μετρήσεις για 4πλάσιο μέγεθος της rgb εικόνας.
- Για να εξεταστεί η επίδραση των Reduce μεταβλήθηκε και το πλήθος των convergence iterations και ακολούθως γίνεται σύγκριση.
- Επιλέχθηκε ένας αριθμός επαναλήψεων ίσος με 30 ώστε η επιτάχυνση να είναι μετρήσιμη για πολλές διεργασίες. Στους πίνακες που ακολουθούν φαίνεται το είδος των εκτελέσεων που έγιναν (κάθε εκτέλεση επαναλήφθηκε 3 φορές και σημειώθηκε η καλύτερη τιμή) καθώς και τα αποτελέσματα χρόνων εκτέλεσης για MPI και για OpenMP+MPI. Να σημειωθεί ότι παρατίθενται για βασικές περιπτώσεις και οι χρόνοι ανάγνωσης / εγγραφής.
- Για την περίπτωση του openMP δεν εξετάστηκαν οι περιπτώσεις με reduce καθώς θεωρήθηκε ότι η επίπτωσή του είναι εμφανής από τις μετρήσεις του απλού MPI.

MPI

Times/Speedups/Efficiencies/Scalability

Παρατίθενται στους ακόλουθους πίνακες τα αποτελέσματα runtime, speedup, efficiency για το MPI πρόγραμμα.

Problem size	**1,2**	**2,4**	**4,8**	**9,7**	**19,4**	**3,6**	**7,3**	**14,5**	**29,0**
Processes / Run times (secs)	**grey (630 x 1920)**	**Grey (1260 x 1920)**	**Grey (2520 x 1920)**	**Grey (5040 x 1920)**	**Grey (10080 x 1920)**	**Rgb (630 x 1920)**	**Rgb (1260 x 1920)**	**Rgb (2520 x 1920)**	**Rgb (5040 x 1920)**
1	1,72	4,06	8,11	16,20	32,37	5,73	11,74	21,79	41,76
2	0,88	2,10	4,05	8,08	16,12	2,74	6,93	11,23	22,52
4	0,47	1,06	2,13	4,26	8,51	1,45	2,77	5,82	11,58
6	0,31	0,71	1,42	2,84	5,68	1,50	2,24	3,88	7,75
8	0,23	0,54	1,07	2,13	4,26	1,31	3,00	2,67	5,81
9	---	0,48	0,95	1,90	3,79	1,19	1,23	2,35	4,70
16	0,12	0,27	0,54	1,07	2,16	0,63	0,67	1,34	2,94
25	0,07	0,17	0,35	0,69	1,40	0,44	0,44	0,86	1,70
32	0,08	0,19	0,37	0,59	1,17	0,18	0,46	0,80	1,43
36	---	0,26	0,34	0,60	1,00	0,36	0,41	0,67	1,25
64	0,16	0,27	0,38	0,48	0,88	0,21	0,53	0,78	1,04
72	---	0,34	0,34	0,45	0,81	0,34	0,48	0,79	1,18
100	0,24	0,30	0,34	0,37	0,63	0,27	0,35	0,57	0,70
108	----	0,31	0,33	0,40	0,65	0,34	0,34	0,56	0,87

Πίνακας 1 : MPI – χρόνοι εκτέλεσης για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

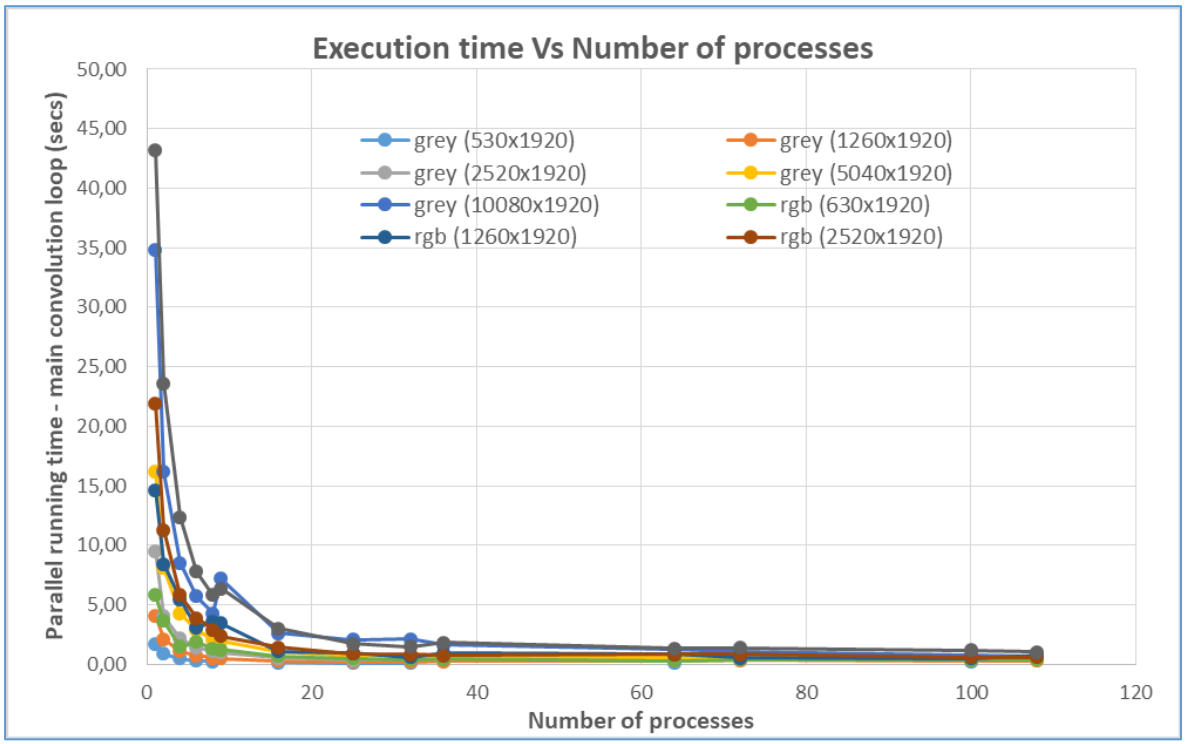
Problem size	1,2	2,4	4,8	9,7	19,4	3,6	7,3	14,5	29,0
Processes / Speedup	grey (630 x 1920)	Grey (1260 x 1920)	Grey (2520 x 1920)	Grey (5040 x 1920)	Grey (10080 x 1920)	Rgb (630 x 1920)	Rgb (1260 x 1920)	Rgb (2520 x 1920)	Rgb (5040 x 1920)
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
2	1,95	1,93	2,00	2,00	2,01	2,09	1,69	1,94	1,85
4	3,66	3,83	3,81	3,80	3,80	3,95	4,24	3,74	3,61
6	5,55	5,72	5,71	5,70	5,70	3,82	5,24	5,62	5,39
8	7,48	7,52	7,58	7,61	7,60	4,37	3,91	8,16	7,19
9	---	8,46	8,54	8,53	8,54	4,82	9,54	9,27	8,89
16	14,33	15,04	15,02	15,14	14,99	9,10	17,52	16,26	14,20
25	24,57	23,88	23,17	23,48	23,12	13,02	26,68	25,34	24,56
32	21,50	21,37	21,92	27,46	27,67	31,83	25,52	27,24	29,20
36	---	15,62	23,85	27,00	32,37	15,92	28,63	32,52	33,41
64	10,75	15,04	21,34	33,75	36,78	27,29	22,15	27,94	40,15
72	---	11,94	23,85	36,00	39,96	16,85	24,46	27,58	35,39
100	7,17	13,53	23,85	43,78	51,38	21,22	33,54	38,23	59,66
108	---	13,10	24,58	40,50	49,80	16,85	34,53	38,91	48,00

Πίνακας 2 : **MPI** – **speedup** για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

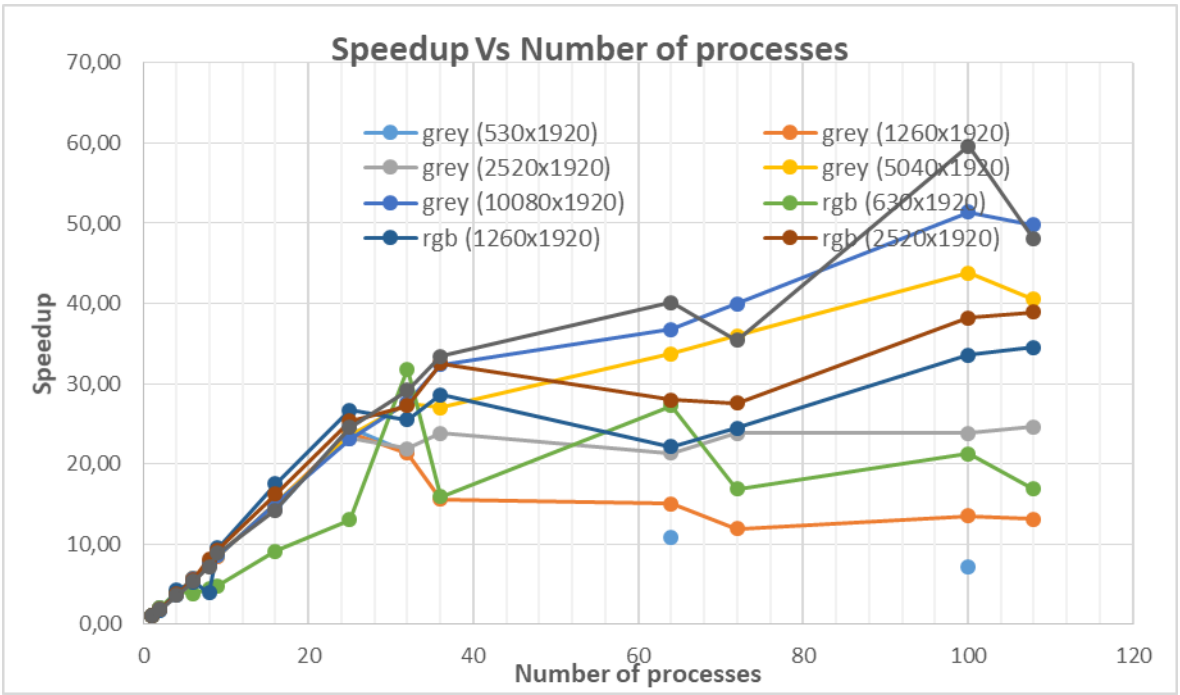
Problem size	1,2	2,4	4,8	9,7	19,4	3,6	7,3	14,5	29,0
Processes / Efficiency	grey (630 x 1920)	Grey (1260 x 1920)	Grey (2520 x 1920)	Grey (5040 x 1920)	Grey (10080 x 1920)	Rgb (630 x 1920)	Rgb (1260 x 1920)	Rgb (2520 x 1920)	Rgb (5040 x 1920)
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
2	0,98	0,97	1,00	1,00	1,00	1,05	0,85	0,97	0,93
4	0,91	0,96	0,95	0,95	0,95	0,99	1,06	0,94	0,90
6	0,92	0,95	0,95	0,95	0,95	0,64	0,87	0,94	0,90
8	0,93	0,94	0,95	0,95	0,95	0,55	0,49	1,02	0,90
9		0,94	0,95	0,95	0,95	0,54	1,06	1,03	0,99
16	0,90	0,94	0,94	0,95	0,94	0,57	1,10	1,02	0,89
25	0,98	0,96	0,93	0,94	0,92	0,52	1,07	1,01	0,98
32	0,67	0,67	0,68	0,86	0,86	0,99	0,80	0,85	0,91
36		0,43	0,66	0,75	0,90	0,44	0,80	0,90	0,93
64	0,17	0,23	0,33	0,53	0,57	0,43	0,35	0,44	0,63
72		0,17	0,33	0,50	0,56	0,23	0,34	0,38	0,49
100	0,07	0,14	0,24	0,44	0,51	0,21	0,34	0,38	0,60
108		0,12	0,23	0,38	0,46	0,16	0,32	0,36	0,44

Πίνακας 3 : **MPI** – **efficiency** για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

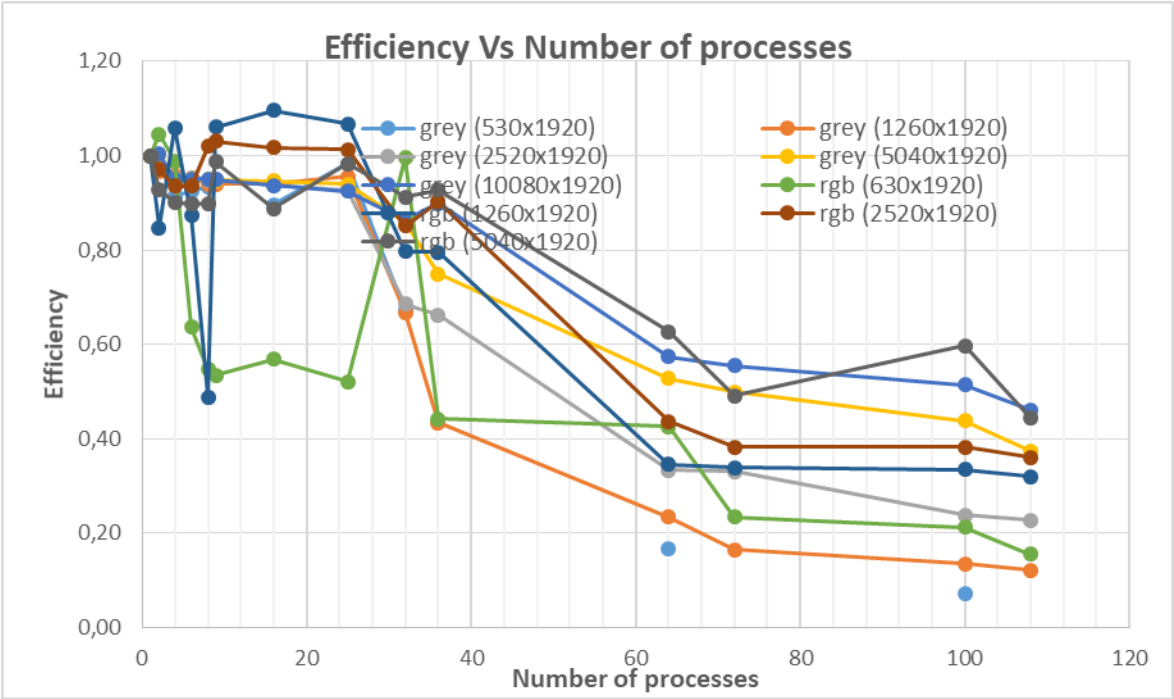
Στα παρακάτω διαγράμματα φαίνονται γραφικά τα αποτελέσματα των παραπάνω αποτελεσμάτων.



Διάγραμμα 1 : **MPI** –Χρόνος εκτέλεσης κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)



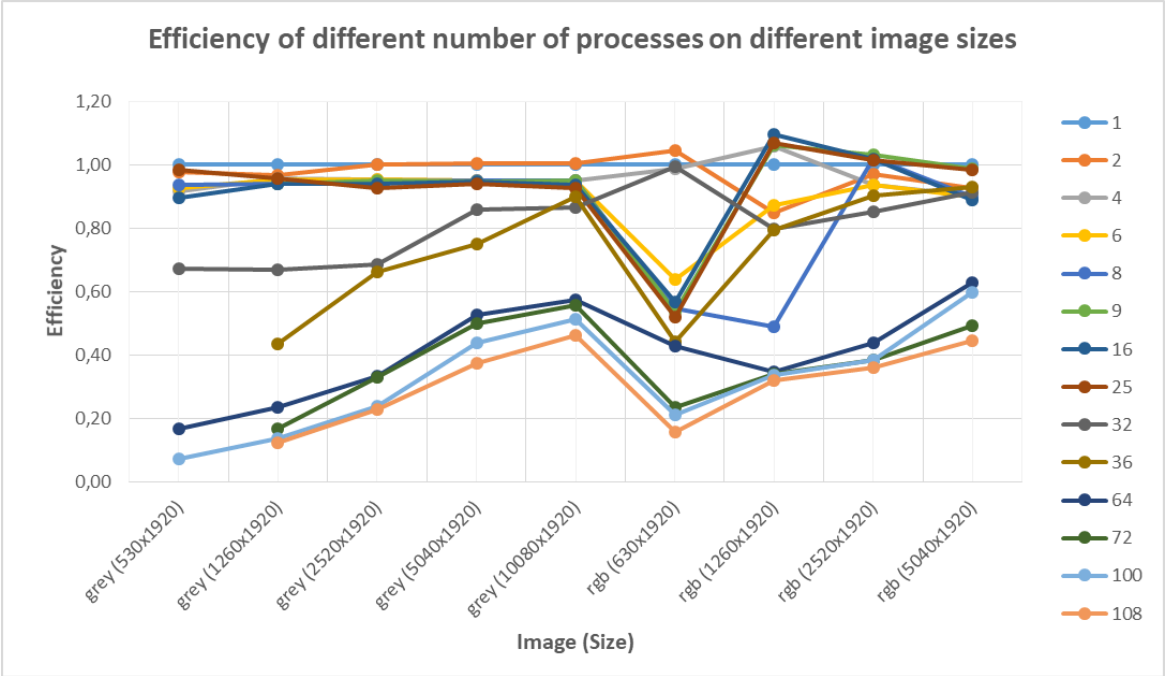
Διάγραμμα 2 : **MPI** –Speedup (Επιτάχυνση) κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)



<u>Διάγραμμα 3</u> : **MPI** –Efficiency (Αποδοτικότητα) κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)

Είναι εμφανές ότι οι χρόνοι και το speedup αυξάνονται σημαντικά στις πρώτες αυξήσεις του αριθμού των διεργασιών (σχεδόν γραμμικά μέχρι 25 διεργασίες) και ακολούθως όπως αναμενόταν ο ρυθμός speedup και το efficiency μειώνονται. Ο ρυθμός μείωσης είναι μικρότερος όσο αυξάνει το μέγεθος του προβλήματος.

Στο επόμενο διάγραμμα βλέπουμε πως επιδρά το μέγεθος της εικόνας στο efficiency ανάλογα με τον αριθμό των διεργασιών.



<u>Διάγραμμα 4</u> : **MPI** –Efficiency για διαφορετικού μεγέθους εικόνες με διαφορετικούς αριθμούς processes.

Τόσο από τα διαγράμματα, όσο και τους πίνακες παρατηρούμε ότι για να διατηρήσουμε το speedup στα ίδια επίπεδα το μέγεθος της εικόνας, μέχρι τις 25 διεργασίες δεν χρειάζεται να αλλάξουμε σημαντικά το μέγεθος της εικόνας, δηλαδή είναι σχεδόν γραμμικό. Σε αυτό το πεδίο το πρόβλημα θα μπορούσε να θεωρηθεί strongly scalable. Όμως με την περαιτέρω αύξηση του αριθμού διεργασιών η αύξηση μεγέθους που απαιτείται είναι πολύ μεγάλη (μεγαλύτερη από διπλάσια για >25 διεργασίες) και μετά από ένα σημείο δεν βλέπουμε δυνατότητα να επιτευχθεί αντίστοιχο efficiency. Προφανώς η επίδραση της αυξημένης ανάγκης για επικοινωνία μεταξύ των διαφορετικών processes αρχίζει να επιβαρύνει σημαντικά τους χρόνους εκτέλεσης και να γίνεται το κύριο bottleneck της συνέλιξης.

Επίδραση convergence (reduce)

Στον επόμενο πίνακα φαίνονται το ποσοστό αύξησης χρόνου εκτέλεσης για την περίπτωση convergence check ανά 2 επαναλήψεις σε σχέση με την περίπτωση χωρίς. Από τον πίνακα αυτό φαίνεται ότι κατά κανόνα όσο αυξάνει το πλήθος των διεργασιών ο χρόνος εκτέλεσης αυξάνει έως και πάνω από 1 φορά (>100%), καθώς η ανάγκη συχρονισμού και επικοινωνίας των διεργασιών, επιβραδύνει τον βαθμό παράλληλων υπολογισμών. Πρακτικά είναι σαν να βάζουμε ένα barrier περιμένοντας όλες τις διεργασίες να στείλουν το αποτέλεσμά τους.

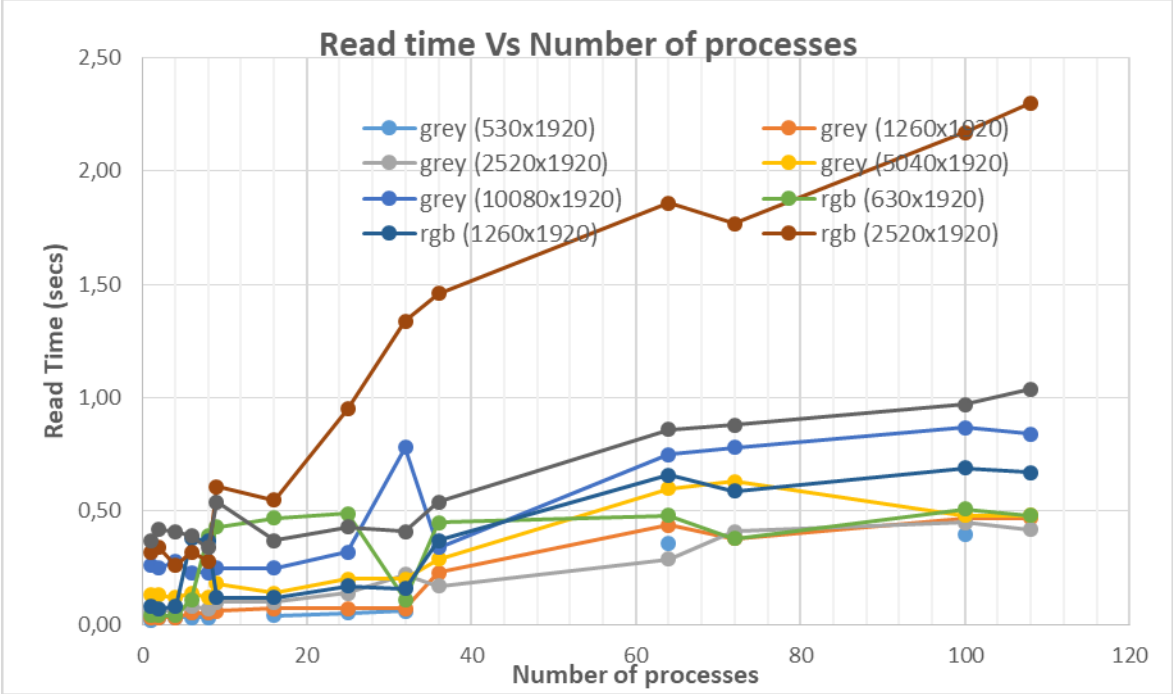
Problem size	**1,2**	**2,4**	**4,8**	**9,7**	**19,4**	**3,6**	**7,3**	**14,5**	**29,0**
Processes / % difference in RunTime	**grey (630 x 1920)**	**Grey (1260 x 1920)**	**Grey (2520 x 1920)**	**Grey (5040 x 1920)**	**Grey (10080 x 1920)**	**Rgb (630 x 1920)**	**Rgb (1260 x 1920)**	**Rgb (2520 x 1920)**	**Rgb (5040 x 1920)**
1	-0,58	0,00	0,00	-0,12	0,12	0,35	-4,94	-2,25	3,04
2	0,00	0,00	-0,25	0,12	0,25	7,30	-20,92	-1,69	-3,20
4	0,00	0,94	1,88	0,70	1,88	0,00	115,16	0,17	0,95
6	0,00	0,00	2,11	6,69	1,23	26,67	56,25	0,00	0,13
8	0,00	0,00	0,93	2,35	2,35	-48,85	-6,33	9,36	0,17
9	---	2,08	2,11	13,16	1,85	-38,66	156,10	0,85	0,21
16	0,00	103,70	0,00	7,48	0,93	-28,57	179,10	2,99	-1,02
25	14,29	194,12	2,86	7,25	2,14	88,64	184,09	3,49	2,35
32	37,50	178,95	8,11	40,68	30,77	72,22	69,57	11,25	35,66
36	---	34,62	11,76	18,33	47,00	116,67	112,20	19,40	41,60
64	281,25	133,33	78,95	87,50	88,64	219,05	33,96	51,28	87,50
72	---	55,88	88,24	108,89	101,23	150,00	56,25	39,24	54,24
100	158,33	130,00	105,88	116,22	84,13	181,48	108,57	78,95	114,29
108	---	161,29	103,03	87,50	66,15	173,53	129,41	50,00	48,28

Πίνακας 4 : MPI – % διαφορά run-time *για convergence check ανά 2 επαναλήψεις σε σχέση με την περίπτωση χωρίς convergence check (set to 30)*, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

Parallel reading/writing

Στο επόμενο διάγραμμα παρουσιάζονται οι χρόνοι ανάγνωσης των αρχείων από parallel IO για διάφορα μεγέθη προβλήματος και αριθμούς διεργασιών. Τα αποτελέσματα είναι αντίστοιχα και για την περίπτωση του writing. Χωρίς καμία εξαίρεση οι χρόνοι αυξάνονται όσο αυξάνεται το πλήθος διεργασιών. Μόνο στις πρώτες αυξήσεις αριθμού διεργασιών φαίνεται να υπάρχει μία μικρή μείωση (έως σταθερότητα). Αυτό μοιάζει παράξενο αλλά στην πραγματικότητα δεδομένου ότι υπάρχει μία ανάγκη συντονισμού των διεργασιών στην ανάγνωση/εγγραφή του αρχείου, αυτό δημιουργεί ένα overhead με την αύξηση του αριθμού διεργασιών.

Να σημειωθεί ότι δοκιμή που έγινε με παράλλο IO, το οποίο όμως δημιουργούσε ξεχωριστούς δείκτες αρχείων και seek set, είχε ως αποτέλεσμα πολύ μεγαλύτερους χρόνους ανάγνωσης (6-10 φορές το ελάχιστο).



<u>Διάγραμμα 5</u> : **MPI** –Χρόνοι reading για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)

OPENMP + MPI

Times/Speedups/Efficiencies/Scalability

Παρατίθενται στους ακόλουθους πίνακες τα αποτελέσματα runtime, speedup, efficiency για το OPENMP+MPI πρόγραμμα.

Problem size	**1,2**	**2,4**	**4,8**	**9,7**	**19,4**	**3,6**	**7,3**	**14,5**	**29,0**
Processes / Run times (secs)	**grey (630 x 1920)**	**Grey (1260 x 1920)**	**Grey (2520 x 1920)**	**Grey (5040 x 1920)**	**Grey (10080 x 1920)**	**Rgb (630 x 1920)**	**Rgb (1260 x 1920)**	**Rgb (2520 x 1920)**	**Rgb (5040 x 1920)**
1	0,78	1,64	3,06	7,13	16,85	1,75	3,52	7,06	13,83
2	0,95	0,79	1,56	10,02	11,53	0,90	1,78	3,54	7,02
4	0,21	0,39	0,78	1,55	3,09	0,47	0,89	1,78	3,53
6	0,15	0,28	0,53	1,03	4,85	0,33	0,60	1,19	2,37
8	0,12	0,21	0,40	0,79	1,54	0,24	0,46	0,89	1,79
9	0,11	0,20	0,35	0,69	1,36	0,22	0,40	0,80	1,59
16	5,73	0,73	2,91	2,20	3,64	1,10	1,13	3,06	5,04
25	0,40	0,58	0,93	1,37	2,88	1,11	0,73	1,96	2,24
32	6,57	6,61	6,43	6,08	7,18	6,26	6,54	6,72	7,48
36	6,62	6,57	6,62	6,96	8,63	6,51	6,52	8,24	11,79
64	9,74	10,18	9,40	10,47	10,27	10,18	9,36	9,48	13,46
72	10,68	10,46	10,54	11,28	11,30	10,52	10,48	10,82	11,75
100	12,84	14,61	14,31	15,12	14,66	13,44	14,37	14,07	20,28
108	14,66	15,12	14,76	15,48	14,21	15,05	15,39	14,55	14,80

Πίνακας 5 : OpenMP+MPI – χρόνοι εκτέλεσης για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

Problem size	**1,2**	**2,4**	**4,8**	**9,7**	**19,4**	**3,6**	**7,3**	**14,5**	**29,0**
Processes / Speedup	**grey (630 x 1920)**	**Grey (1260 x 1920)**	**Grey (2520 x 1920)**	**Grey (5040 x 1920)**	**Grey (10080 x 1920)**	**Rgb (630 x 1920)**	**Rgb (1260 x 1920)**	**Rgb (2520 x 1920)**	**Rgb (5040 x 1920)**
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
2	0,82	2,08	1,96	0,71	1,46	1,94	1,98	1,99	1,97
4	3,71	4,21	3,92	4,60	5,45	3,72	3,96	3,97	3,92
6	5,20	5,86	5,77	6,92	3,47	5,30	5,87	5,93	5,84
8	6,50	7,81	7,65	9,03	10,94	7,29	7,65	7,93	7,73
9	7,09	8,20	8,74	10,33	12,39	7,95	8,80	8,83	8,70
16	0,14	2,25	1,05	3,24	4,63	1,59	3,12	2,31	2,74
25	1,95	2,83	3,29	5,20	5,85	1,58	4,82	3,60	6,17
32	0,12	0,25	0,48	1,17	2,35	0,28	0,54	1,05	1,85
36	0,12	0,25	0,46	1,02	1,95	0,27	0,54	0,86	1,17
64	0,08	0,16	0,33	0,68	1,64	0,17	0,38	0,74	1,03
72	0,07	0,16	0,29	0,63	1,49	0,17	0,34	0,65	1,18
100	0,06	0,11	0,21	0,47	1,15	0,13	0,24	0,50	0,68
108	0,05	0,11	0,21	0,46	1,19	0,12	0,23	0,49	0,93

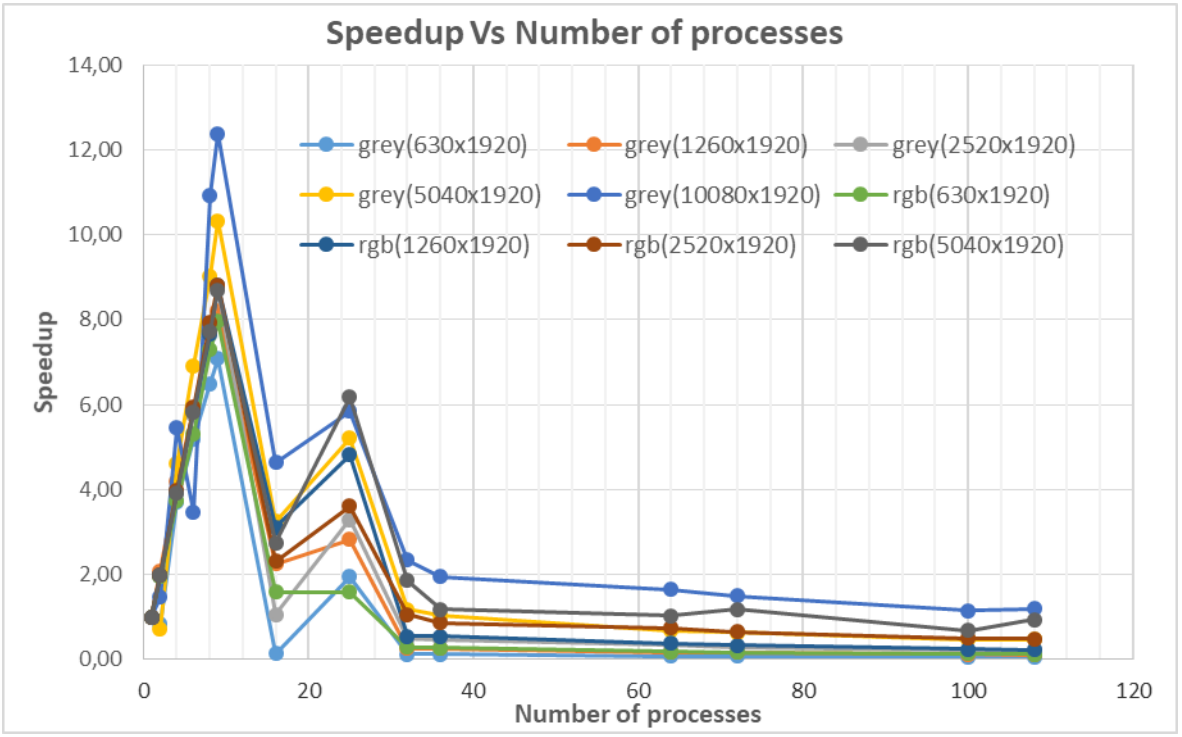
Πίνακας 6 : OpenMP+MPI – speedup για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

Problem size	1,2	2,4	4,8	9,7	19,4	3,6	7,3	14,5	29,0
Processes / Efficiency	grey (630 x 1920)	Grey (1260 x 1920)	Grey (2520 x 1920)	Grey (5040 x 1920)	Grey (10080 x 1920)	Rgb (630 x 1920)	Rgb (1260 x 1920)	Rgb (2520 x 1920)	Rgb (5040 x 1920)
1	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
2	0,41	1,04	0,98	0,36	0,73	0,97	0,99	1,00	0,99
4	0,93	1,05	0,98	1,15	1,36	0,93	0,99	0,99	0,98
6	0,87	0,98	0,96	1,15	0,58	0,88	0,98	0,99	0,97
8	0,81	0,98	0,96	1,13	1,37	0,91	0,96	0,99	0,97
9	0,79	0,91	0,97	1,15	1,38	0,88	0,98	0,98	0,97
16	0,01	0,14	0,07	0,20	0,29	0,10	0,19	0,14	0,17
25	0,08	0,11	0,13	0,21	0,23	0,06	0,19	0,14	0,25
32	0,00	0,01	0,01	0,04	0,07	0,01	0,02	0,03	0,06
36	0,00	0,01	0,01	0,03	0,05	0,01	0,01	0,02	0,03
64	0,00	0,00	0,01	0,01	0,03	0,00	0,01	0,01	0,02
72	0,00	0,00	0,00	0,01	0,02	0,00	0,00	0,01	0,02
100	0,00	0,00	0,00	0,00	0,01	0,00	0,00	0,01	0,01
108	0,00	0,00	0,00	0,00	0,01	0,00	0,00	0,00	0,01

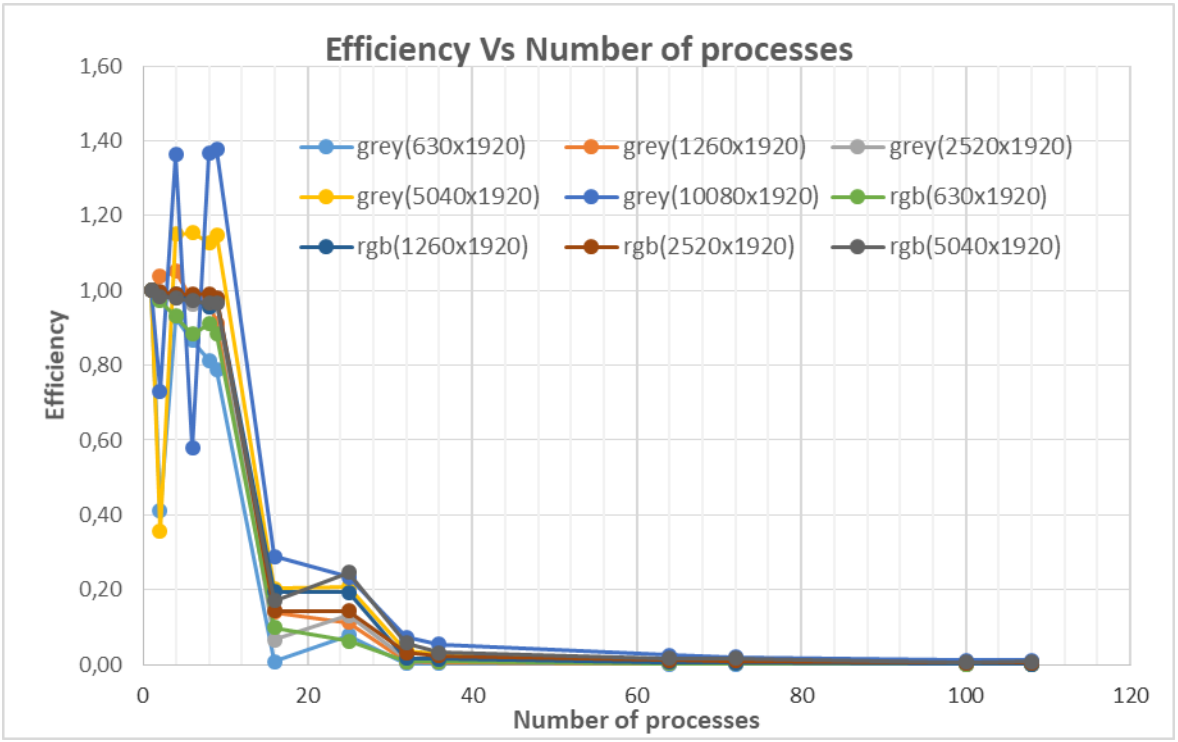
Πίνακας 7 : OpenMP+MPI – efficiency για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

Στα επόμενα διαγράμματα φαίνονται γραφικά τα αποτελέσματα των παραπάνω αποτελεσμάτων.

Διάγραμμα 6 : **OpenMP + MPI -** Χρόνος εκτέλεσης κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)



Διάγραμμα 7 : **OpenMP + MPI -** Speedup (Επιτάχυνση) κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)



Διάγραμμα 8 : **OpenMP + MPI -** Efficiency (Αποδοτικότητα) κυρίου υπολογισμού (όχι read/write) για εικόνες διαφόρων μεγεθών και διαφορετικό πλήθος διεργασιών (1 – 108)

Παρατηρούμε μία επιτάχυνση για μικρούς αριθμούς διεργασιών (έως 9), και στη συνέχεια η απόδοση μειώνεται δραστικά, πιθανότατα λόγω θεμάτων χρονισμού μεταξύ πολλών διεργασιών και λίγων threads και ανταγωνισμού στη μνήμη των threads.

Problem size	1,2	2,4	4,8	9,7	19,4	3,6	7,3	14,5	29,0
Delta (OMP-MPI)/MPI – in run times (%)	grey (630 x 1920)	Grey (1260 x 1920)	Grey (2520 x 1920)	Grey (5040 x 1920)	Grey (10080 x 1920)	Rgb (630 x 1920)	Rgb (1260 x 1920)	Rgb (2520 x 1920)	Rgb (5040 x 1920)
1	-55%	-60%	-62%	-56%	-48%	-69%	-70%	-68%	-67%
2	8%	-62%	-61%	24%	-28%	-67%	-74%	-68%	-69%
4	-55%	-63%	-63%	-64%	-64%	-68%	-68%	-69%	-70%
6	-52%	-61%	-63%	-64%	-15%	-78%	-73%	-69%	-69%
8	-48%	-61%	-63%	-63%	-64%	-82%	-85%	-67%	-69%
9		-58%	-63%	-64%	-64%	-82%	-67%	-66%	-66%
16	4675%	170%	439%	106%	69%	75%	69%	128%	71%
25	471%	241%	166%	99%	106%	152%	66%	128%	32%
32	8113%	3379%	1638%	931%	514%	3378%	1322%	740%	423%
36		2427%	1847%	1060%	763%	1708%	1490%	1130%	843%
64	5988%	3670%	2374%	2081%	1067%	4748%	1666%	1115%	1194%
72		2976%	3000%	2407%	1295%	2994%	2083%	1270%	896%
100	5250%	4770%	4109%	3986%	2227%	4878%	4006%	2368%	2797%
108		4777%	4373%	3770%	2086%	4326%	4426%	2498%	1601%

Πίνακας 8 : OpenMP+MPI vs MPI in run times (difference %) για 30 iterations ***χωρίς convergence check (set to 30)***, για διάφορες εικόνες/μεγέθη και αριθμούς διεργασιών

Παρατηρούμε ότι μέχρι και τις 9 διεργασίες ο χρόνος εκτέλεσης επιταχύνεται στο combined OpenMP+MPI κατά 0,5 με 1 φορά. Μέχρι δηλαδή τα (1x4 threads) x 9 = 36 threads υπάρχει επιτάχυνση σε σχέση με τα (1x4 processes) x 9 = 36 processes. Σε μεγαλύτερα μεγέθη ο χρόνος αυξάνεται δραματικά και το efficiency μειώνεται ταχέως. Στην πράξη φαίνεται τα threads να ανταγωνίζονται για τη κοινή μνήμη και να δημιουργούν θέματα α-συγχρονισμού που επιβραδύνουν τη διαδικασία.

5. ΕΠΕΚΤΑΣΕΙΣ

Τα προγράμματα που αναπτύχθηκαν επιδέχονται φυσικά βελτίωση.

Πιθανές βελτιώσεις/επεκτάσεις :

- Όσον αφορά τον κώδικα του παραλληλισμού θα μπορούσε να προστεθεί ένα σύστημα συντεταγμένων για να δημιουργηθεί ένα virtual Cartesian process grid. Με τη δημιουργία του οι τοπολογίες των διεργασιών θα ήταν άμεσα διαθέσιμες ενώ οι send/receive θα απλοποιούνταν για τις διεργασίες που δεν έχουν γείτονες και ο κώδικας θα γινόταν πιο ευανάγνωστος.
- Τα προγράμματα θα μπορούσαν να δέχονται φίλτρα οποιουδήποτε μεγέθους. Αυτό θα απαιτούσε να αυξηθεί η «άλως» της επικοινωνίας κατά τον αριθμό των επιθυμητών κελιών για κάθε διάσταση φίλτρου (π.χ. για 5x5 θα έπρεπε η άλως να περιλαμβάνει από 2 pixel δεξιά αριστερά της εικόνας και να αποστέλλονται 2 γραμμές/στήλες). Με τον

τρόπο που δομήθηκε η αρίθμηση στο πρόγραμμα (ενιαία διαχείριση grey/rgb) η μετατροπή αυτή θα απαιτούσε λίγες αλλαγές.

- Για οποιαδήποτε φίλτρα θα μπορούσε να γίνει εισαγωγή από τη γραμμή εντολών
- Το openmp επίσης θα μπορούσε να λαμβάνει την τιμή των threads από τη γραμμή εντολών

6. ΠΑΡΑΡΤΗΜΑ – ΕΠΙΣΥΝΑΠΤΟΜΕΝΑ

Στο αρχείο που υποβάλλεται συμπεριλαμβάνονται :

1. Η παρούσα αναφορά
2. Ο κώδικας των δύο προγραμμάτων
3. Το log των τρεξιμάτων των δύο προγραμμάτων
4. Το script που χρησιμοποιήθηκε για compilation , τρέξιμο
5. Ένα simple make file για κάθε περίπτωση