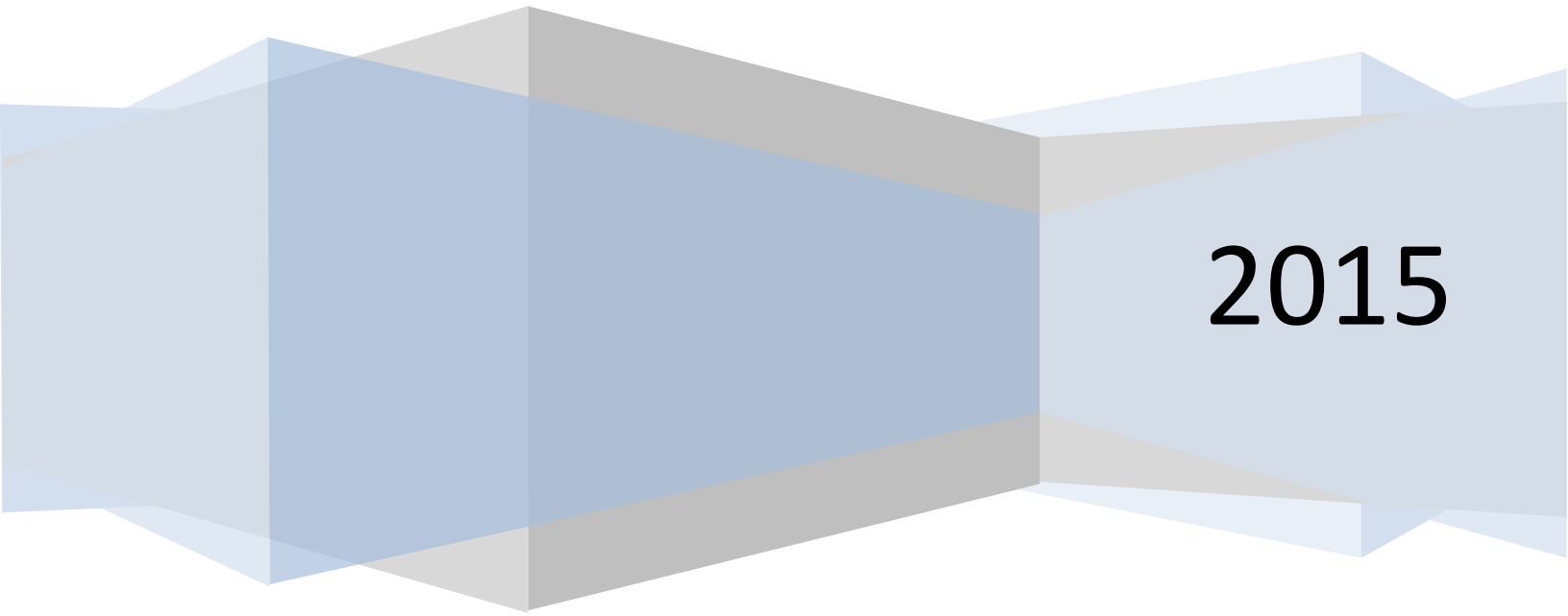
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ
ΕΘΝΙΚΟ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Εργασία MPI/OpenMP & Cuda: Συνέλιξη Εικόνων

Καρβουνάρη Σοφία Αριθμός Μητρώου:1115201000200

Κατσίνα Λευκοθέα Αριθμός Μητρώου:1115201000051



2015

Περιεχόμενα

1.ΕΙΣΑΓΩΓΗ

Στην πληροφορική, παράλληλα, κατανεμημένα ή ταυτόχρονα συστήματα ονομάζονται υπολογιστές οι οποίοι επιτρέπουν την ταυτόχρονη εκτέλεση πολλαπλών συνεργαζόμενων προγραμμάτων σε μία ή περισσότερες επεξεργαστικές μονάδες. Οι διαφορές μεταξύ αυτών των όρων είναι λεπτές, με την έμφαση να δίνεται άλλοτε στον σχεδιασμό και ανάλυση αλγορίθμων, άλλοτε στην κατασκευή υποστηρικτικού λογισμικού και άλλοτε στη σχεδίαση των υποδομών υλικού που απαιτούνται για την επίτευξη του ταυτοχρονισμού.

Στην παρούσα εργασία εξετάζουμε την λύση ενός προβλήματος που επιτρέπει την παράλληλη επεξεργασία δεδομένων, συγκεκριμένα την εφαρμογή φίλτρου συνέλιξης σε μια εικόνα, με την βοήθεια τριών διαφορετικών αρχιτεκτονικών. Σκοπός της εργασίας αποτελεί, εκτός από την παραγωγή κώδικα για τις τρεις αρχιτεκτονικές(Distributed Memory Programming with MPI, Shared Memory Programming with OpenMP, Αρχιτεκτονική Cuda), η μέτρηση και αξιολόγηση της απόδοσης των προγραμμάτων διαφορετικής αρχιτεκτονικής καθώς και η μελέτη κλιμάκωσης με αυξανόμενο μέγεθος εικόνας και αριθμό διεργασιών/threads.

Ο κώδικας που αναπτύχθηκε, επομένως αποτελείται από τρεις διαφορετικές υλοποιήσεις. Η πρώτη αναπτύχθηκε σε περιβάλλον MPI, στο οποίο δημιουργούμε <n> διεργασίες στις μηχανές του αρχείου machines, καθεμιά εκ των οποίων αναλαμβάνει την εφαρμογή του φίλτρου σε ένα τμήμα της αρχικής εικόνας. Η δεύτερη αναπτύχθηκε με συνδυασμό των αρχιτεκτονικών MPI και OpenMP, με τους υπολογισμούς στην εκτέλεση του φίλτρου να διαχωρίζεται ανάμεσα σε threads, για βελτίωση της απόδοσης. Η τρίτη και τελευταία υλοποίηση αφορά τον προγραμματισμό GPUs μέσω του περιβάλλοντος Cuda και πραγματοποιείται μέσω της δημιουργίας νημάτων/threads, καθένα εκ των οποίων αναλαμβάνει την εφαρμογή του φίλτρου σε ένα μόνο pixel της εικόνας. Περαιτέρω λεπτομέρειες για την υλοποίηση και τις σχεδιαστικές μας επιλογές θα δοθούν στη συνέχεια.

2. ΔΟΜΗ ΚΩΔΙΚΑ ΚΑΙ ΣΧΕΔΙΑΣΤΙΚΕΣ ΕΠΙΛΟΓΕΣ

2.1 Υλοποίηση MPI

Αρχικά έχουμε δημιουργήσει μια καρτεσιανή τοπολογία διεργασιών μέσω της συνάρτησης MPI_Cart_create() θέτοντας στην μεταβλητή reorder την τιμή 1, το οποίο επιτρέπει στην συνάρτηση να προχωρήσει σε πιθανή αναδιάταξη των διεργασιών στη νέα ομάδα, με ενδεχόμενη καλύτερη αντιστοιχία της τοπολογίας των διεργασιών στην πραγματική τοπολογία των επεξεργαστών του παράλληλου συστήματος. Επιπλέον, και οι δύο διαστάσεις χαρακτηρίζονται από περιοδικότητα, έτσι ώστε τα οριακά σημεία να έχουν ως γείτονες τα συμμετρικά τους. Στην συνέχεια, μετά από το διάβασμα της εικόνας από την διεργασία με τιμή τάξης μηδέν, πραγματοποιείται ο διαμοιρασμός των δεδομένων της εικόνας στις υπάρχουσες διεργασίες, μέσω της συνάρτησης MPI_Scatterv().Για τον διαμοιρασμό των στοιχείων

ιδιαίτερα χρήσιμα αποτελούν τα vector dataTypes που δημιουργούμε, ένα που προσδιορίζει την μορφή των δεδομένων που στέλνονται σε κάθε διεργασία από τον αρχικό πίνακα image και ένα που προσδιορίζει την μορφή των δεδομένων που λαμβάνονται από την κάθε διεργασία, έτσι ώστε να δημιουργείται στον πίνακα λήψης localImage και ένα εξωτερικό grid, ειδικό για την λήψη των στοιχείων από τους γείτονες. Στην συνέχεια πραγματοποιείται η κλήση της filterImage, η οποία αρχικά δηλώνει τον πίνακα του φίλτρου που θα χρησιμοποιηθεί. Για να αποφύγουμε αντιγραφές και να μπορούμε να έχουμε κάθε φορά στην διάθεση μας τα στοιχεία του πίνακα πριν και μετά την εφαρμογή του φίλτρου, δημιουργούμε ακόμη έναν πίνακα. Οι δύο πίνακες λειτουργούν εναλλάξ ως αποθήκες των παλιών και των τροποποιημένων τιμών, όπως φαίνεται και μέσα στην επανάληψη. Πριν ξεκινήσει η επανάληψη, πραγματοποιείται ο υπολογισμός των γειτόνων και η προετοιμασία της επικοινωνίας και με τους δύο πίνακες οι οποίοι χρησιμοποιούνται. Η προετοιμασία αυτή είναι δυνατή μέσω των συναρτήσεων MPI_Send_init και MPI_Recv_init, οι οποίες δημιουργούν μια συνεχή επικοινωνία(persistent communication) με τους γείτονες, βελτιώνοντας την απόδοση. Και σε αυτήν την περίπτωση, η χρήση vector dataTypes αποτρέπει τις αντιγραφές και την χρήση buffers που θα μείωναν σημαντικά την απόδοση. Με την έναρξη της επανάληψης πραγματοποιείται η έναρξη αποστολής στους γείτονες των στοιχείων που χρειάζονται και λήψης από τους γείτονες των στοιχείων που η τρέχουσα διεργασία απαιτεί. Στην διάρκεια της αναμονής, πραγματοποιείται επικάλυψη επικοινωνίας με υπολογισμούς, με το φιλτράρισμα του εσωτερικού κομματιού της τοπικής εικόνας, το οποίο διαθέτει όλους τους γείτονες που χρειάζονται. Μόλις ολοκληρωθεί η λήψη των στοιχείων από τους γείτονες φιλτράρεται και το εξωτερικό κομμάτι της τοπικής εικόνας. Μετά από STEPS αριθμό βημάτων πραγματοποιείται σύγκριση των εικόνων πριν και μετά την εφαρμογή του φίλτρου, ενώ με την συνάρτηση MPI_Reduce() ελέγχουμε αν όλες οι διεργασίες συμφωνούν στο αν θα πρέπει να σταματήσουμε την εφαρμογή του φίλτρου, σε περίπτωση μη περαιτέρω επίδρασης του στην εικόνα.

2.2 Υλοποίηση MPI σε συνεργασία με OpenMP

Για βελτίωση της απόδοσης του προγράμματος μας, έχουμε την δυνατότητα μέσω του περιβάλλοντος OpenMP, να δημιουργούμε σε κάθε μηχανή που τρέχει μια διεργασία MPI, έναν αριθμό νημάτων/threads που αναλαμβάνουν την παράλληλη πραγματοποίηση των υπολογισμών του φιλτραρίσματος της εικόνας. Για την καλύτερη αξιοποίηση αυτής της δυνατότητας, δημιουργούμε πριν ξεκινήσει η επανάληψη τα νήματα μέσω της εντολής #pragma omp parallel num_threads(thread_count), και πριν από κάθε for προσθέτουμε την εντολή #pragma omp for. Με αυτόν τον τρόπο αποτρέπουμε την δημιουργία/καταστροφή νημάτων και βελτιώνουμε την απόδοση. Τις εντολές που επιθυμούμε να πραγματοποιήσουμε μόνο από ένα νήμα αναλαμβάνει να εκτελέσει το master thread μέσω της εντολής #pragma omp master, ενώ όταν επιθυμούμε τον συγχρονισμό των νημάτων χρησιμοποιούμε την εντολή

#pragma omp barrier. Υπήρξε η σκέψη παράλληλης σύγκρισης των δύο πινάκων από τα threads, η οποία εγκαταλείφθηκε καθώς δεν είναι δυνατή η τοποθέτηση break σε for επανάληψη η οποία εκτελείται με OpenMP.

2.3 Υλοποίηση Cuda

Σε αυτήν την υλοποίηση η παραλληλία επιτυγχάνεται με τη χρήση της κάρτας γραφικών Nvidia. Κάθε thread που δημιουργείται αναλαμβάνει το φιλτράρισμα ενός μόνο pixel της εικόνας. Αφού πραγματοποιήσουμε τις απαραίτητες δεσμεύσεις και μεταφορές από την μνήμη host στην device, δημιουργούμε τα νήματα και τα blocks που απαιτούνται για την επίλυση του προβλήματος και εκκινείται η επανάληψη του φιλτραρίσματος. Μετά από συγκεκριμένο αριθμό βημάτων, αφού πρώτα ανακτηθούν οι πίνακες στο host, ελέγχονται οι δύο εικόνες t και t' για σύγκλιση, και αν η τελευταία έχει επιτευχθεί τερματίζουμε την επανάληψη, πραγματοποιούμε τις απαραίτητες αποδεσμεύσεις μνήμης και τερματίζουμε.

2.4 Μεταγλώττιση

MPI:
mpicc -o mpiprog main.c functions.c -lm
Parallel :mpiexec -f machines -n <n> mpiprog
Seriel: mpiexec -n <n> mpiprog

OPENMP:
mpicc -g -Wall -fopenmp ompprog main.c functions.c -lm
Parallel: mpiexec -f machines -n <n> ompprog num_of_threads
Serial: mpiexec -n <n> ompprog num_of_threads

CUDA:
apo qlogin -->
nvcc -o cudaprog main.cpp functions.cpp gpu_cudaProg.cu -w -lm
./cudaProg

Σημείωση: το αρχείο machines περιέχει τα μηχανήματα της σχολής (στο machines περιγράφεται και ο αριθμός των πυρήνων σε κάθε CPU (που ισοδυναμεί με τον αριθμό των ταυτόχρονων processes που εκτελούνται)).

3. ΜΕΤΡΗΣΕΙΣ, ΕΠΙΔΟΣΕΙΣ & ΚΛΙΜΑΚΩΣΗ

Όλες οι μετρήσεις γίνονται με την χρήση ενός αρχείου που έχουμε αρχικοποιήσει με γεννήτρια τυχαίων αριθμών μέσω ενός άλλου, μικρού προγράμματος. Η επιλογή αυτή έγινε για ευκολία στις μετρήσεις των διαφόρων μεγεθών προβλήματος, καθώς η σύγκλιση στην περίπτωση που διαβάζεται η δοθείσα εικόνα είναι πιο αργή και θέλει περισσότερα βήματα. Συμπεραίνουμε επομένως ότι στις μετρήσεις ιδιαίτερο ρόλο παίζει και η είσοδος. Επιπλέον, θεμιτό ήταν το να έχουμε πάντα μια σταθερή είσοδο για την σωστή πραγματοποίηση των μετρήσεων. Φροντίσαμε να μην πραγματοποιήσουμε τις μετρήσεις σε ώρα αιχμής, διότι όταν πολλοί χρήστες διαμοιράζονται τους διαθέσιμους πόρους, η απόδοση μειώνεται κατακόρυφα.

3.1 Υλοποίηση MPI

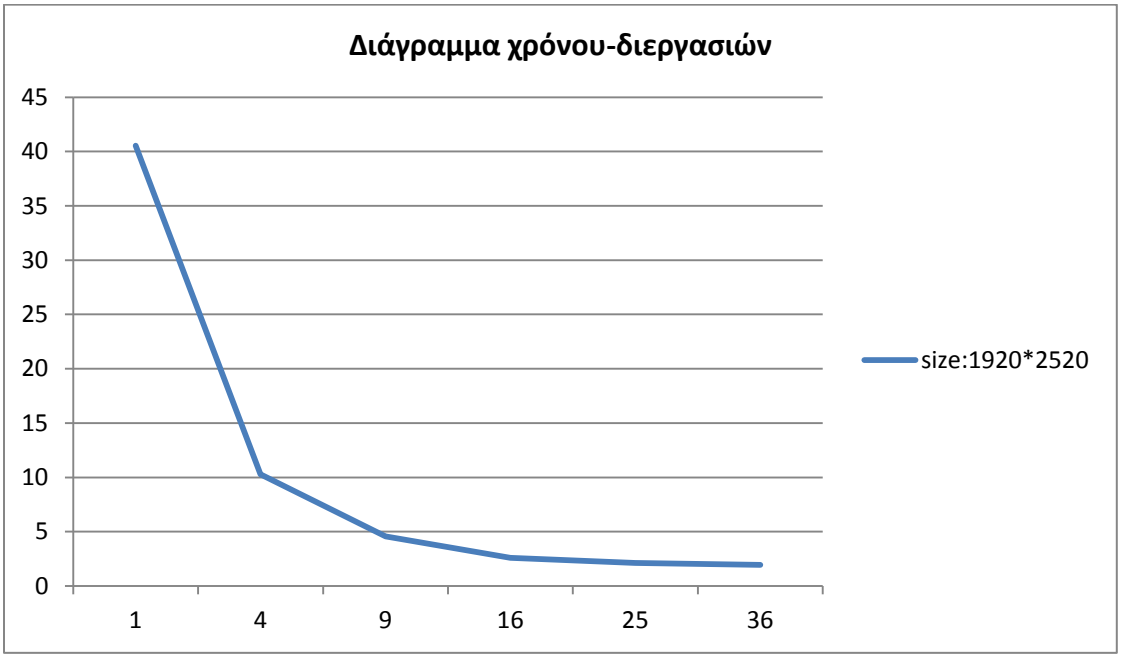
Οι παρακάτω μετρήσεις αφορούν μόνο το φιλτράρισμα της εικόνας, χωρίς το διάβασμα της εικόνας και την προετοιμασία της επικοινωνίας.

Στον παρακάτω πίνακα φαίνεται μια μελέτη κλιμάκωσης για αριθμό διεργασιών και μέγεθος προβλήματος, όταν η σύγκριση των πινάκων γίνεται κάθε ***5 steps.***

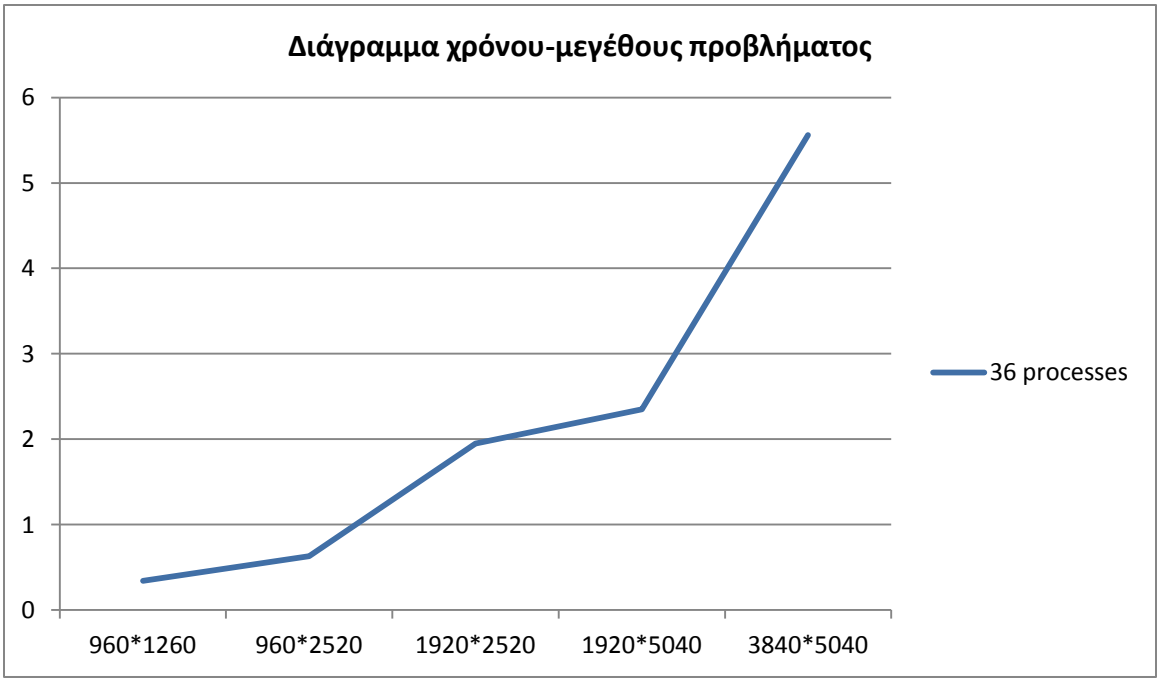
	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
1		9.328466	20.254834	40.554834	80.577378	161.735923
4		2.394623	5.164592	10.284639	20.349273	40.673927
9		1.059249	2.352984	4.559238	9.038964	18.252854
16		0.669273	1.353091	2.587398	5.117445	10.263491
25		0.478229	0.816947	2.139759	3.484528	6.515926
36		0.348389	0.637284	1.954676	2.351993	5.566719

Παρατηρούμε από τα αποτελέσματα ότι για κάθε σταθερή τιμή διάστασης ,αυξάνοντας τον αριθμό των διεργασιών έχουμε μείωση στο χρόνο εκτέλεσης. Αυτό είναι δικαιολογημένο, αφού έτσι αυξάνεται η παραλληλία, την οποία εκμεταλλευόμαστε μέσω της επικοινωνίας και της τοπολογίας διεργασιών. Στην περίπτωση που κρατήσουμε τον αριθμό των διεργασιών σταθερό και αυξήσουμε το μέγεθος της εικόνας, ο χρόνος γίνεται πολύ μεγαλύτερος, αφού τα δεδομένα προς επεξεργασία αυξάνονται. Τα βήματα είναι μόλις 5, συνεπώς η global επικοινωνία είναι αυξημένη και αυτό περιμένουμε ότι έχει επίδραση στους χρόνους ολοκλήρωσης των εργασιών. Σημειώνουμε ότι η αρχική εικόνα ολοκληρώνει την επεξεργασία της σε 65 βήματα.

Παρακάτω παρατίθεται ένα διάγραμμα για διαστάσεις εικόνας 1920*2520 για τη σχέση χρόνου-διεργασιών για 5 βήματα:



Επιπλέον παραθέτουμε ένα διάγραμμα για σταθερό αριθμό 36 διεργασιών και μεταβαλλόμενο μέγεθος προβλήματος.

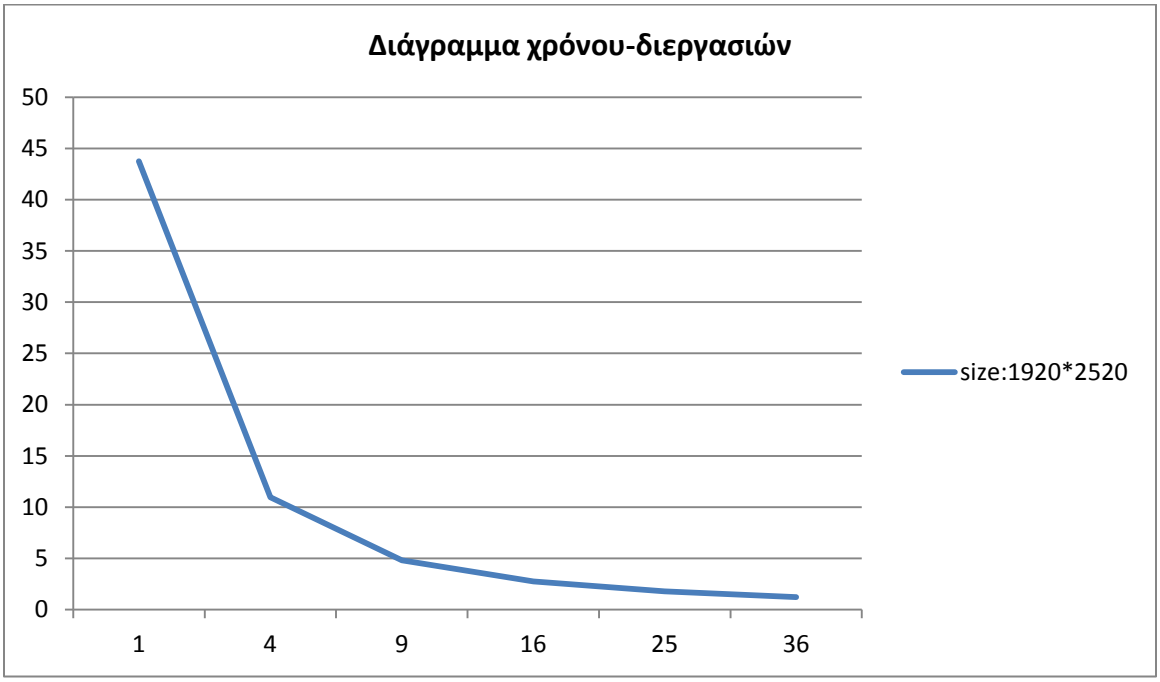


Στον παρακάτω πίνακα φαίνεται μια μελέτη κλιμάκωσης για αριθμό διεργασιών και μέγεθος προβλήματος, όταν η σύγκριση των πινάκων γίνεται κάθε ***35 steps.***

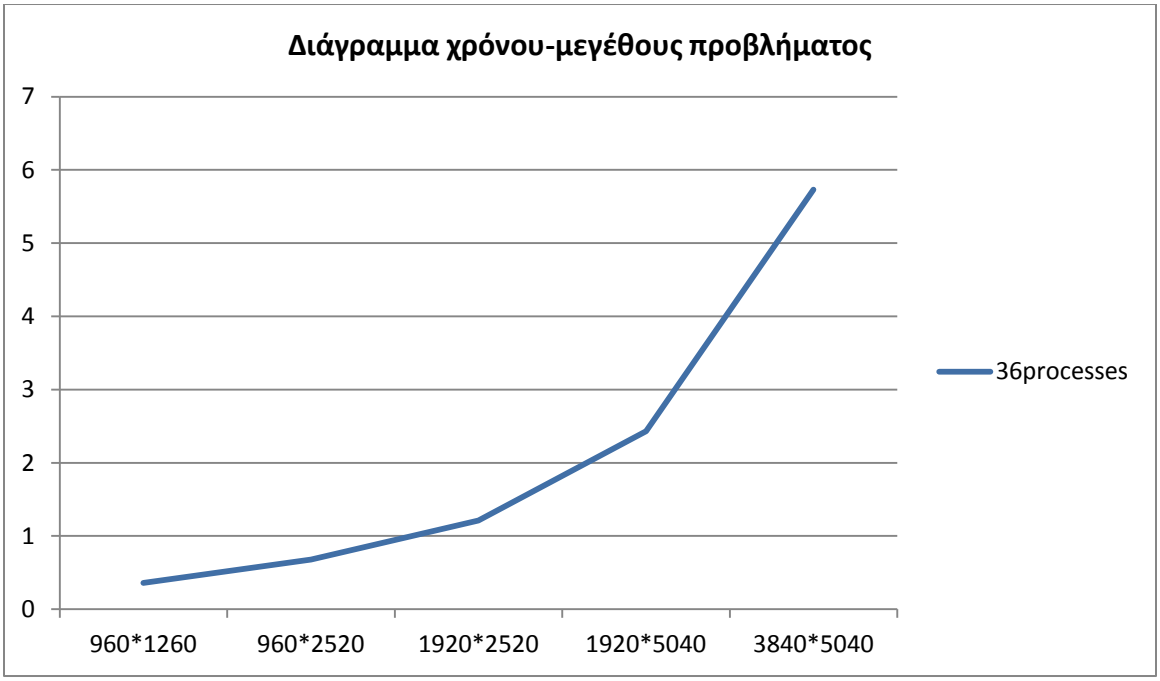
	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
1		10.962974	21.836544	43.730657	87.263973	174.163824
4		2.829461	5.492793	10.962668	22.273954	44.739542
9		1.273043	2.484992	4.826837	9.824843	19.943782
16		0.739627	1.692394	2.743948	5.539674	11.273945
25		0.419327	0.962179	1.792653	3.637492	7.0482329
36		0.362983	0.681733	1.213759	2.437295	5.739453

Η συμπεριφορά του χρόνου είναι ίδια με πριν, αφού αυξάνοντας τις διεργασίες για σταθερό μέγεθος προβλήματος η παραλληλία αυξάνεται και συνεπώς ο χρόνος μειώνεται. Αντίστοιχα, για σταθερό αριθμό διεργασιών με αυξανόμενο μέγεθος εικόνας, το πλήθος δεδομένων προς επεξεργασία αυξάνεται, άρα ο χρόνος που απαιτείται είναι πολύ περισσότερος. Επειδή τα βήματα είναι περισσότερα από την προηγούμενη περίπτωση ,η global επικοινωνία μειώνεται, όμως δεν παρατηρείται πάντα βελτίωση. Αυτό συμβαίνει διότι, λόγω των βημάτων ολοκλήρωσης του φιλτραρίσματος για την συγκεκριμένη είσοδο, σε αυτήν την περίπτωση των 35 βημάτων, γίνονται περιττές επαναλήψεις, που στην περίπτωση των 5 βημάτων δεν πραγματοποιούνται. Καταλήγουμε λοιπόν στο συμπέρασμα ότι οι περιττές επαναλήψεις επηρεάζουν περισσότερο την απόδοση από την global επικοινωνία.

Παρακάτω παρατίθεται ένα διάγραμμα για διαστάσεις εικόνας 1920*2520 για τη σχέση χρόνου-διεργασιών για 35 βήματα:



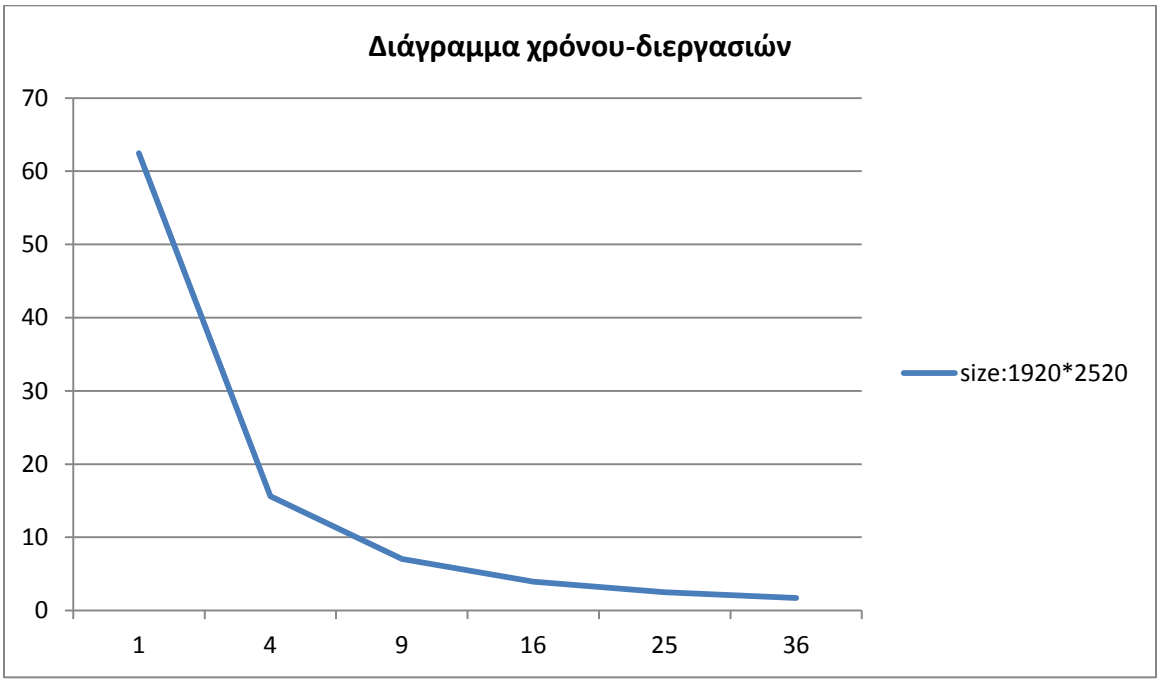
Επιπλέον παραθέτουμε ένα διάγραμμα για σταθερό αριθμό 36 διεργασιών και μεταβαλλόμενο μέγεθος προβλήματος.



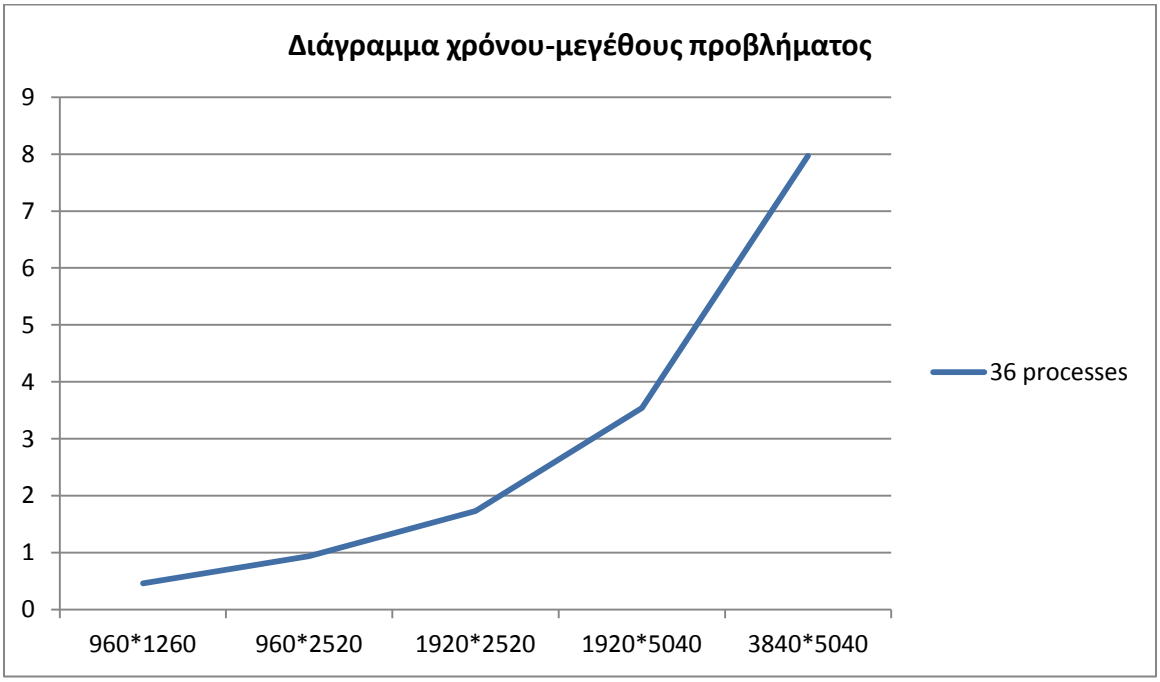
Στον παρακάτω πίνακα φαίνεται μια μελέτη κλιμάκωσης για αριθμό διεργασιών και μέγεθος προβλήματος, όταν η σύγκριση των πινάκων γίνεται κάθε ***50 steps.***

	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
1		15.573593	31.170495	62.464897	124.453923	248.841063
4		3.960398	7.852854	15.612743	31.253864	62.453695
9		1.777249	3.562349	7.069387	14.437832	27.955281
16		1.013169	1.995016	3.934956	7.853042	15.633794
25		0.664386	1.243776	2.512856	5.054075	10.038611
36		0.463483	0.942856	1.738663	3.542805	7.973992

Για άλλη μία φορά η συμπεριφορά του χρόνου για σταθερό μέγεθος προβλήματος/μεταβλητό αριθμό διεργασιών και σταθερό αριθμό διεργασιών/μεταβλητό μέγεθος εικόνας είναι δικαιολογημένα παρόμοια με τις προηγούμενες περιπτώσεις. Σε αυτήν την περίπτωση, οι περιττές επαναλήψεις που πραγματοποιούνται είναι αρκετές, με φανερή επίδραση στην απόδοση. Παρακάτω παρατίθεται ένα διάγραμμα για διαστάσεις εικόνας 1920*2520 για τη σχέση χρόνου-διεργασιών για 50 βήματα:



Επιπλέον παραθέτουμε ένα διάγραμμα για σταθερό αριθμό 36 διεργασιών και μεταβαλλόμενο μέγεθος προβλήματος.

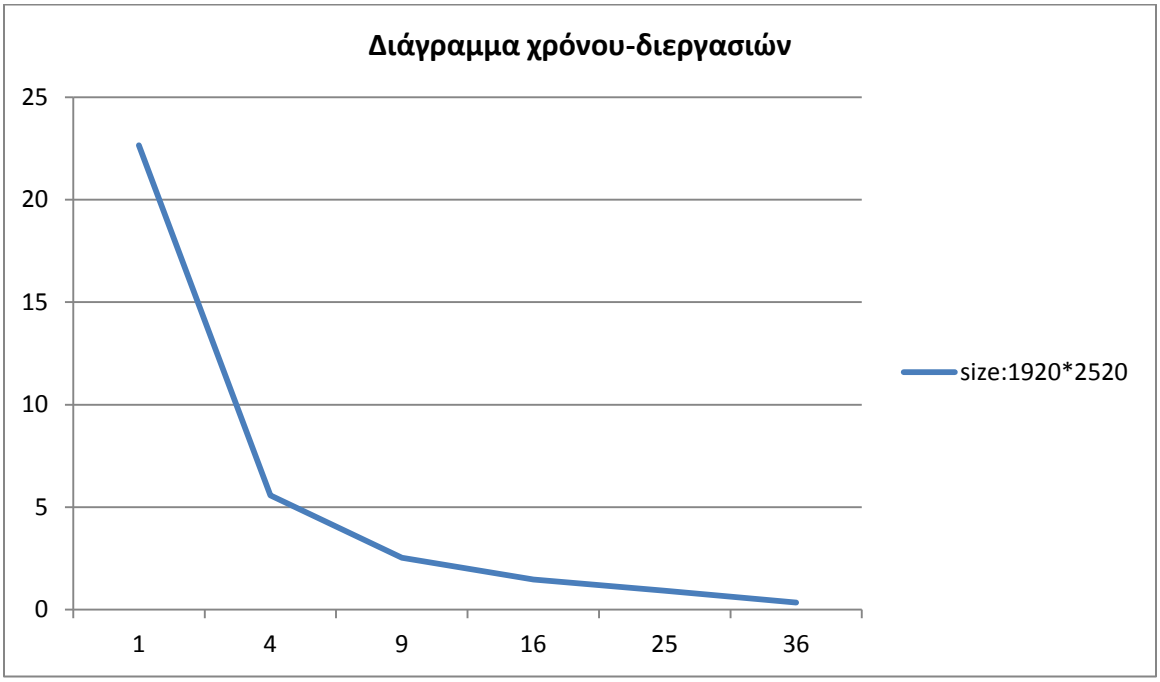


3.2 Υλοποίηση MPI σε συνεργασία με OpenMP

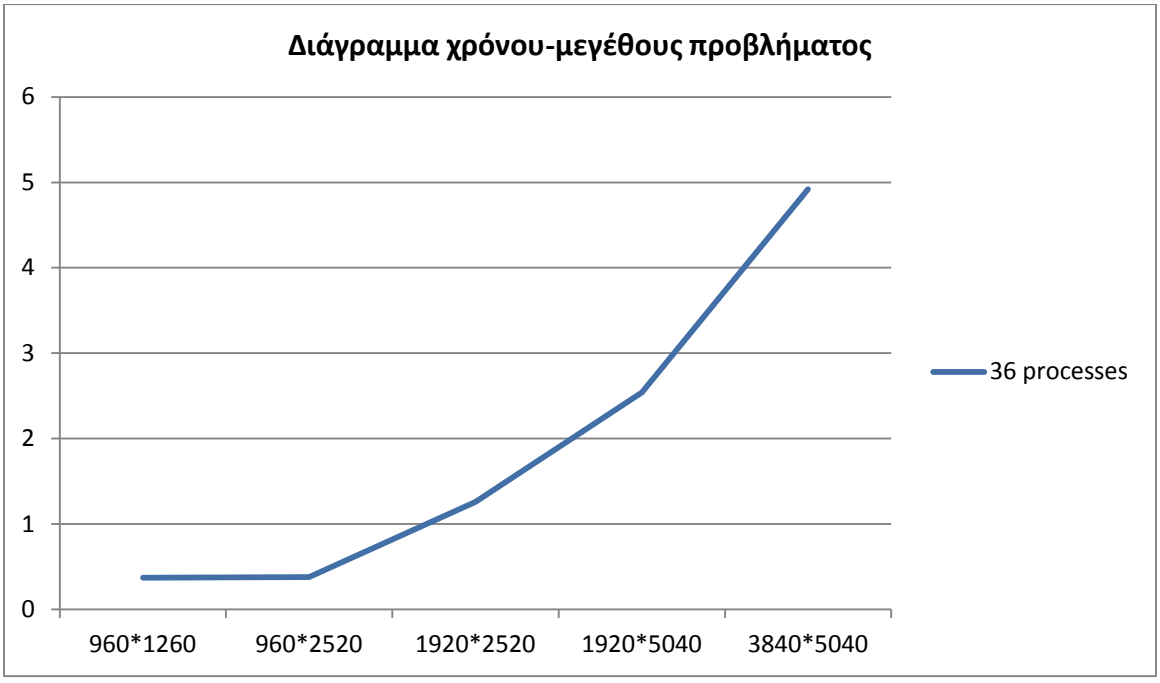
Στην περίπτωση του MPI εξετάσαμε την επίδραση της αύξησης των steps στην απόδοση, συνεπώς τώρα θα διατηρήσουμε τα steps σταθερά στην τιμή 35(μια ενδιάμεση περίπτωση) για την πραγματοποίηση των μετρήσεων στην περίπτωση του OpenMP.

	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
1		5.528493	34.765546	22.657485	44.376465	89.459134
4		2.739649	5.587465	11.645374	21.964529	44.629841
9		1.283064	2.547569	5.146453	9.862984	19.732569
16		0.734920	1.476756	2.856767	5.573049	11.583852
25		0.582096	0.928782	1.964753	3.538198	7.118409
36		0.372975	0.387189	1.265746	2.547295	4.928642

Παρακάτω παρατίθεται ένα διάγραμμα για διαστάσεις εικόνας 1920*2520 για τη σχέση χρόνου-διεργασιών:



Επιπλέον παραθέτουμε ένα διάγραμμα για σταθερό αριθμό 36 διεργασιών και μεταβαλλόμενο μέγεθος προβλήματος.



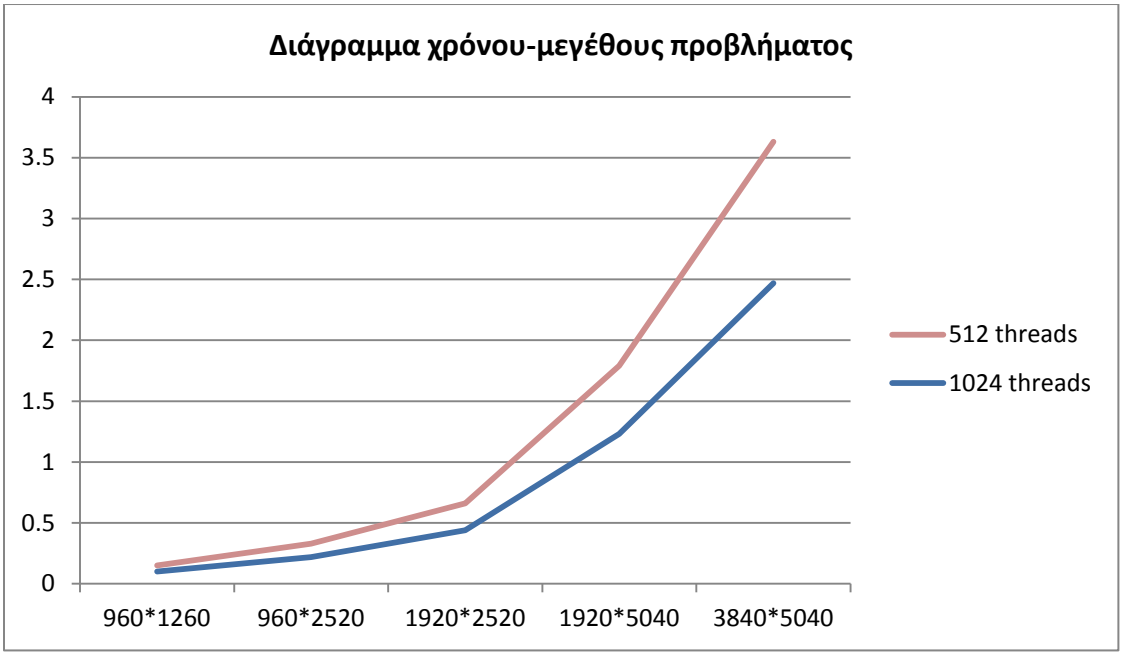
Παρατηρώντας τους χρόνους εκτέλεσης του προγράμματος MPI σε συνεργασία με OpenMP, δεν παρατηρούμε τη βελτίωση που αναμέναμε. Αυτό είναι δικαιολογημένο, λαμβάνοντας υπόψη μας το κόστος συγχρονισμού των νημάτων, που τελικά έχει αρνητικό αντίκτυπο στο τελικό χρόνο του προγράμματος. Υπάρχουν διάφορες τεχνικές που μπορούν να εφαρμοστούν προκειμένου η προσθήκη του OpenMP να έχει θετική επίδραση στην απόδοση, όμως αυτές βρίσκονται εκτός των ορίων αυτού του μαθήματος.

3.3 Υλοποίηση σε Cuda

Στην περίπτωση του MPI εξετάσαμε την επίδραση της αύξησης των steps στην απόδοση, συνεπώς τώρα θα διατηρήσουμε τα steps σταθερά στην τιμή 35(μια ενδιάμεση περίπτωση) για την πραγματοποίηση των μετρήσεων στην περίπτωση του Cuda.

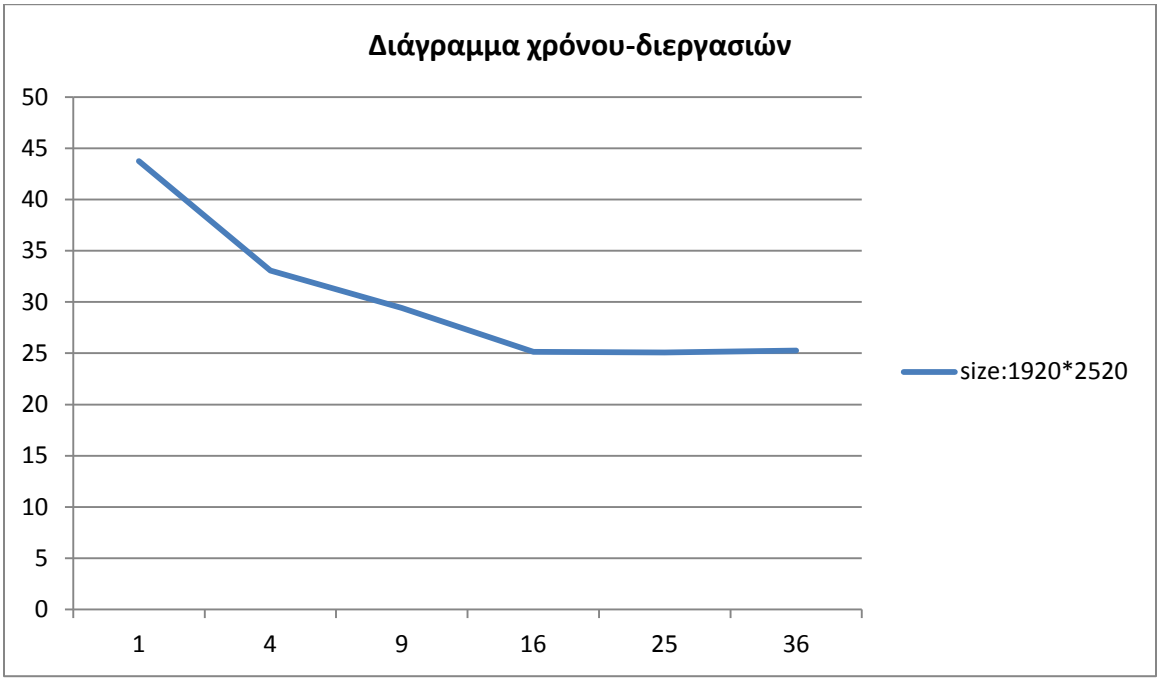
Threads per block:	**Image size**	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
	Secs					
1024		0.10	0.22	0.44	1.23	2.47
512		0.05	0.11	0.22	0.56	1.16

Στην περίπτωση της Cuda υλοποίησης δεν υπάρχει και ο επιπλέον χρόνος που απαιτείται για τον διαμοιρασμό των δεδομένων. Η απόδοση του προγράμματος είναι εξαιρετικά βελτιωμένη σε σχέση με τις δύο προηγούμενες υλοποιήσεις. Το γεγονός αυτό φανερώνει τις δυνατότητες που μας προσφέρει η χρήση των καρτών γραφικών μέσω της σωστής αξιοποίησης τους.



Προκειμένου να πραγματοποιήσουμε κάποιες επιπλέον συγκρίσεις θα πραγματοποιήσουμε μετρήσεις και για το σειριακό πρόγραμμα, που εκτελείται μεν με διεργασίες αλλά σε μία μόνο μηχανή για 35 βήματα.

	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
1		10.900784	21.772922	43.730657	107.941827	174.397922
4		5.571988	10.956004	33.088011	45.483955	87.484001
9		6.207995	12.379972	29.415992	44.487977	93.851994
16		5.795995	11.199985	25.139978	44.051987	87.839989
25		6.119988	11.987997	25.072001	48.487992	91.667992
36		6.620003	12.247994	25.252007	52.256005	93.463981

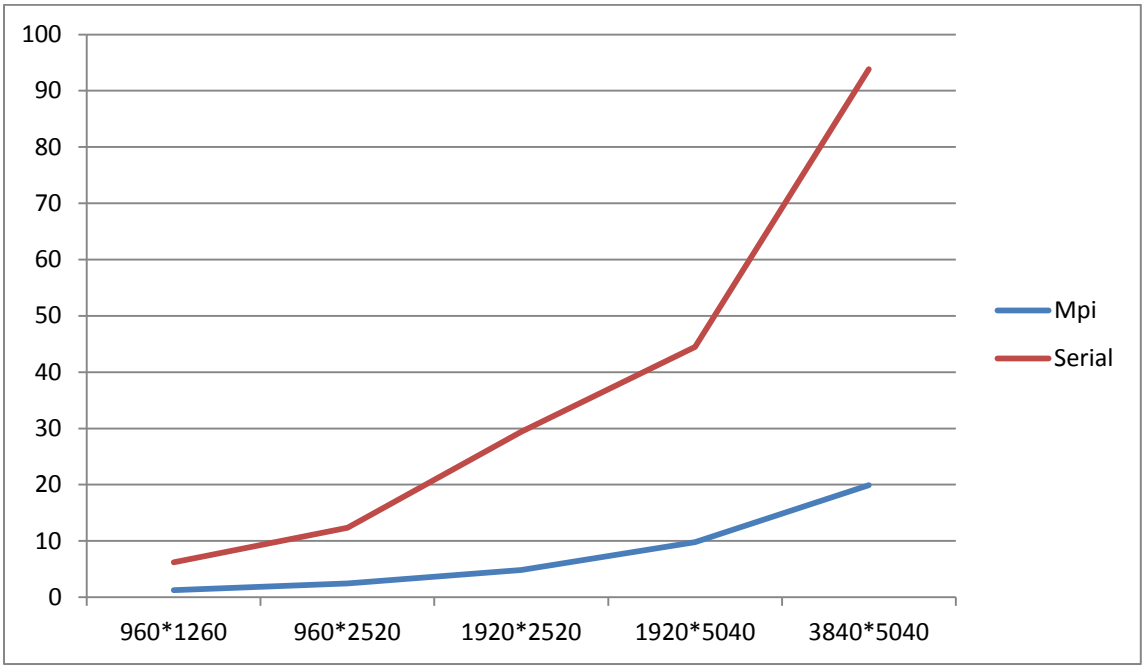


Παρατηρούμε ότι μετά από ένα σημείο, όσο και να αυξήσουμε τον αριθμό των διεργασιών η απόδοση παραμένει στάσιμη ή και χειρότερη. Αυτό είναι δικαιολογημένο, διότι οι πόροι σε μία μόνο μηχανή είναι περιορισμένοι και μετά από ένα σημείο η εναλλαγή ανάμεσα στις διεργασίες επιβαρύνει την απόδοση. Παρακάτω παραθέτουμε περισσότερες συγκρίσεις.

3.4 Συγκρίσεις αποδόσεων

Σε αυτό το σημείο μπορούμε να παραθέσουμε διαγράμματα συγκρίσεων μεταξύ των διαφορετικών υλοποιήσεων που έχουμε μελετήσει.

MPI vs. Serial με αριθμό διεργασιών: 9

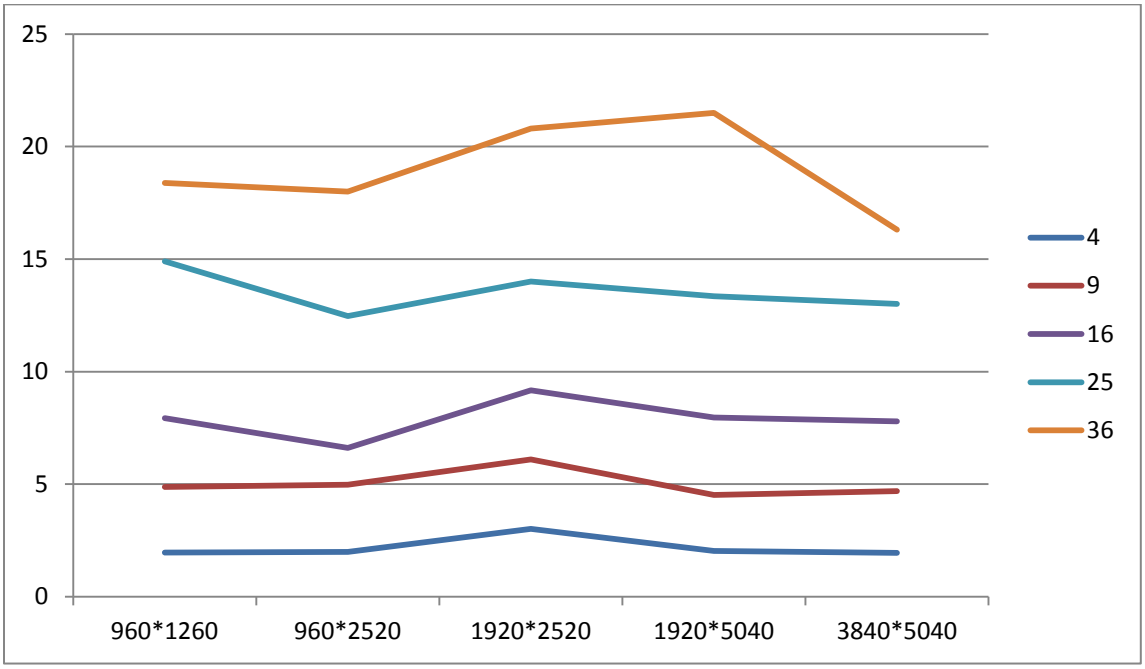


Παρουσιάζουμε έναν πίνακα που δείχνει την μεταβολή της επιτάχυνσης ανάλογα με το μέγεθος προβλήματος και τον αριθμό των διεργασιών.

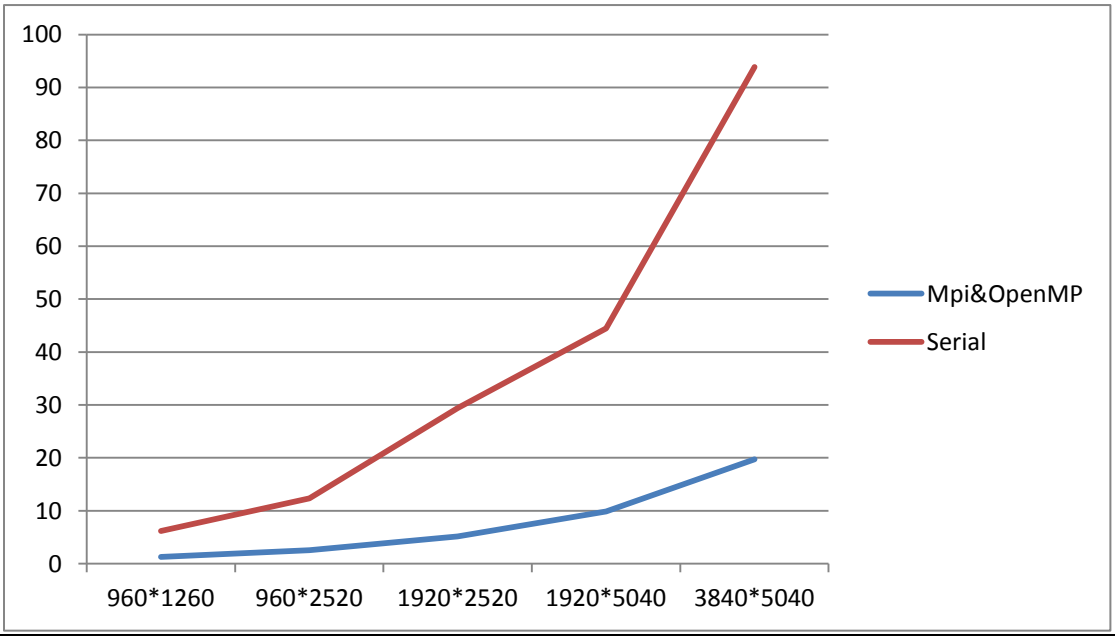
Speedup-Efficiency S=Serial/MPI Parallel

	Image size	960*1260 (1/4)	960*2520 (1/2)	1920*2520	1920*5040 (x2)	3840*5040 (x4)
processes	*Secs*					
4		1.97	1.99	3.01	2.04	1.95
9		4.88	4.98	6.1	4.52	4.7
16		7.93	6.62	9.17	7.96	7.79
25		14.9	12.47	14	13.35	13.01
36		18.38	18	20.8	21.5	16.31

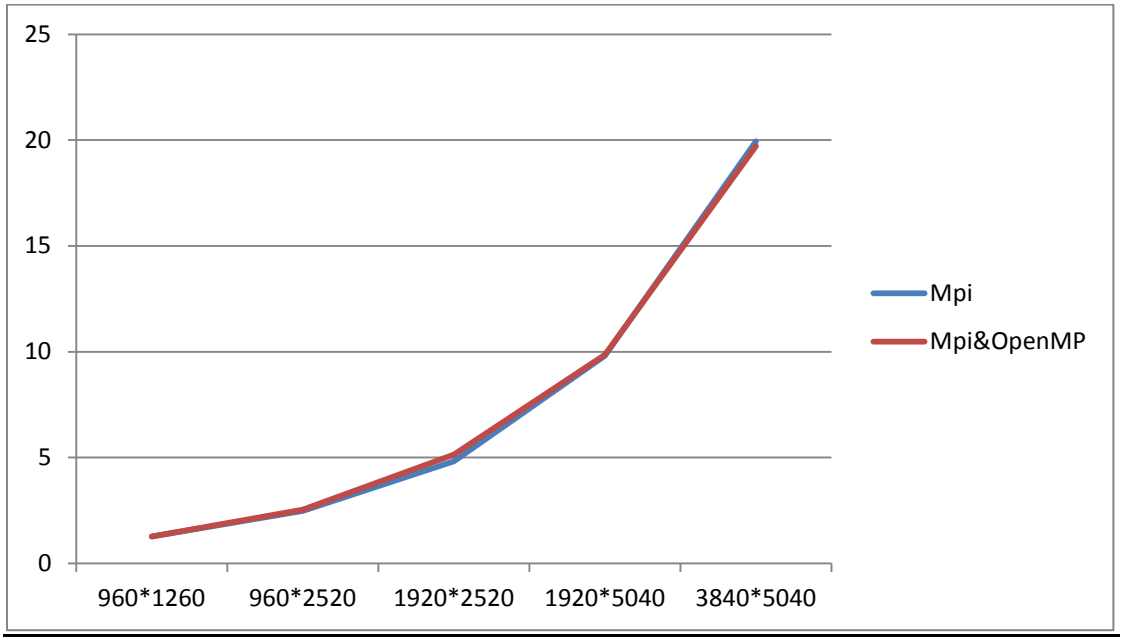
Παρατηρούμε ότι η επιτάχυνση που επιτυγχάνεται με την χρήση πολλών μηχανών είναι εξαιρετική και αυξάνεται με την αύξηση των processes.Μέγιστη επιτάχυνση παρουσιάζεται για μέγεθος εικόνας 1920*5040 και 36 διεργασίες. Παρακάτω παρουσιάζουμε και το αντίστοιχο διάγραμμα.



MPI&OPENMP vs. Serial με αριθμό διεργασιών: 9



MPI vs. MPI&OPENMP με αριθμό διεργασιών: 9



Στο παρακάτω διάγραμμα παρατίθεται μια τελική σύγκριση ανάμεσα σε όλες τις υλοποιήσεις και για τους καλύτερους δυνατούς χρόνους που έχουν επιτευχθεί στην καθεμιά, για αυξανόμενο μέγεθος προβλήματος και για αριθμό βημάτων 35.

Cuda vs. MPI vs MPI&OpenMP vs Serial