ΕΘΝΙΚΟ ΚΑΙ
ΚΑΠΟΔΙΣΤΡΙΑΚΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΑΘΗΝΩΝ

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Παράλληλα Συστήματα (2017-2018)

Εργασία MPI-MPI/OPENMP: Συνέλιξη εικόνων

Μπούσιου Κυριακή – 1115201400120

Γεωργάρα Βασιλική – 1115201400026

1. Εισαγωγή

Η εργασία αυτή έγινε στα πλαίσια του μαθήματος Παράλληλα Συστήματα και αφορά τη συνέλιξη δισδιάστατων εικόνων, είτε ασπρόμαυρων είτε έγχρωμων, με παραλληλοποιημένο πρόγραμμα, ώστε να δούμε τα πλεονεκτήματα του παράλληλου προγραμματισμού, κυρίως ως προς την ταχύτητα.

Η πρώτη προσέγγιση έγινε με το MPI και η δεύτερη με συνδυασμό MPI και OPENMP, ενώ η τρίτη προσέγγιση με CUDA δεν υλοποιήθηκε, λόγω του ότι δεν ήταν διαθέσιμη η κάρτα γραφικών από το πανεπιστήμιο κι εμείς δεν είχαμε πρόσβαση σε κάποια.

2. Δομή Κώδικα

Η εργασία υλοποιήθηκε σε C. Υπάρχει ένας φάκελος MPI_OPENMP στον οποίο περιέχονται όλα τα αρχεία κώδικα και το makefile, καθώς και οι εικόνες τις οποίες θα επεξεργαστούμε. Πιο αναλυτικά:

main.c: από εκεί ξεκινάει το πρόγραμμα και καλούνται όλες οι απαραίτητες συναρτήσεις που βρίσκονται στα υπόλοιπα αρχεία.

convolution.c: περιλαμβάνει τη συνάρτηση για την κανονικοποίηση του φίλτρου και τη συνάρτηση που υλοποιεί τη συνέλιξη για κάποια pixel.

communications.c: περιλαμβάνει συναρτήσεις για τον υπολογισμό των γειτόνων, για την αποστολή και παραλαβή δεδομένων και για τα αντίστοιχα wait.

parallelIO.c: περιλαμβάνει 2 συναρτήσεις, μία για παράλληλο διάβασμα και μία για παράλληλο γράψιμο μετά την επεξεργασία.

general.c: άλλες συναρτήσεις, όπως αυτή για το διάβασμα των παραμέτρων, για τον υπολογισμό του χρόνου και για την υλοποίηση του allreduce.

3. Οδηγίες για τη μεταγλώττιση και εκτέλεση

Υπάρχει makefile, οπότε η μεταγλώττιση γίνεται με την εντολή make mpi για να μεταγλωττιστεί το πρόγραμμα ώστε να τρέχει μόνο MPI, και make openmp για να ώστε να τρέχει και MPI και OpenMP. Επίσης, υπάρχει και η εντολή make clean για καθάρισμα των αντικειμενικών αρχείων, του εκτελέσιμου και της επεξεργασμένης εικόνας.

Για την εκτέλεση, τρέχουμε την εντολή mpirun με παραμέτρους τον αριθμό των επεξεργαστών, το εκτελέσιμο αρχείο, το αρχείο της εικόνας, τον τύπο(rgb ή grey) και έπειτα το πλάτος, το ύψος, τον αριθμό των επαναλήψεων και προαιρετικά έναν αριθμό που μας δείχνει κάθε πόσες επαναλήψεις θα γίνεται το allReduce. Ένα παράδειγμα εκτέλεσης είναι το παρακάτω:

Mpirun –np 4 ./exec waterfall_grey_1920_2520.raw grey 1920 2520 20

4.1 Εξήγηση υλοποίησης

Αρχικά, η διεργασία 0 διαβάζει και αποθηκεύει όλα τα ορίσματα της γραμμής εντολών και τα κάνει broadcast και στις υπόλοιπες διεργασίες. Στη συνέχεια κανονικοποιείται το φίλτρο, εκτός και αν το άθροισμα των στοιχείων του πίνακα είναι 0, οπότε το φίλτρο μένει ως έχει. Μετά υπολογίζεται το blockOffset, το οποίο είναι η πρώτη θέση της εικόνας από την οποία ξεκινάει κάθε διεργασία. Γίνεται παράλληλο διάβασμα με τη συνάρτηση parallel_read και δεσμεύεται ο χώρος για τις δύο εικόνες (t0 για πριν το φίλτρο και t1 μετά την εφαρμογή του). Έχουμε χρησιμοποιήσει 2 MPI_Datatypes, έναν vector για τις γραμμές και έναν contiguous για τις στήλες. Στη συνέχεια υπολογίζονται οι γείτονες, οι οποίοι είναι 8 για κάθε διεργασία. Στην αρχή τα αρχικοποιούμε με -3 και ύστερα, αν υπάρχουν οι γείτονες παίρνουν τον αριθμό της διεργασίας. Πριν ξεκινήσει η επανάληψη, υπολογίζουμε και αποθηκεύουμε τον χρόνο.

Για κάθε φορά που εφαρμόζουμε το φίλτρο, πρώτα καλούμε τη συνάρτηση send_receive η οποία στέλνει στους γείτονες και λαμβάνει από αυτούς τα κομμάτια που χρειάζεται. Αυτό γίνεται με Isend και Irecv που είναι non blocking συναρτήσεις ώστε ενώ περιμένουμε να ολοκληρωθούν να επεξεργαστούμε τα εσωτερικά pixel. Γίνεται η συνέλιξη για τα εσωτερικά pixel και μετά για τα εξωτερικά πρώτα περιμένουμε με MPI_Wait να λυθούν τα δεδομένα από τους γείτονες, ύστερα κάνουμε τη συνέλιξη των εξωτερικών pixel και τέλος περιμένουμε και για την αποστολή των δεδομένων στους γείτονες. Έπειτα αν έχουμε δώσει αριθμό allReduce ελέγχουμε αν βρισκόμαστε σε επανάληψη που πρέπει να κάνουμε allReduce, δηλαδή να δούμε αν υπάρχουν μεταβολές στα t0 και t1 αλλιώς να διακόψουμε την επανάληψη. Πριν κλείσουμε την επανάληψη, κάνουμε swap τους δείκτες για t0 και t1.Μετά την επανάληψη ξανά υπολογίζουμε τον χρόνο, βρίσκουμε τη διαφορά με τον αρχικό και στέλνουμε στη διεργασία 0 τον χρόνο μας ώστε αυτή να βρει τον μεγαλύτερο και να τον εκτυπώσει. Τέλος καλείται η parallel_write για παράλληλο γράψιμο της εικόνας στο αρχείο εξόδου και ελευθερώνουμε όλες τις δομές που έχουμε δεσμεύσει δυναμικά.

4.2 Παραδοχές

Α) Οι συναρτήσεις μας είναι γενικές για rgb και για grey, δηλαδή έχουμε μια μεταβλητή type η οποία γίνεται defined 3 ή 1 αντίστοιχα και όλες μας οι τιμές γίνονται επί type, όπου χρειάζεται για να μην γράφουμε τις ίδιες συναρτήσεις 2 φορές και αυτό εμφανίζει βελτίωση στην απόδοση.

Β) έχουμε χρησιμοποιήσει κάποιες extern μεταβλητές, τον αριθμό γραμμών και στηλών κάθε μπλοκ και το type, διότι αυτές χρησιμοποιούνται συχνά στις συναρτήσεις ώστε να μην τις περνάμε σαν ορίσματα και χάνουμε από χρόνο.

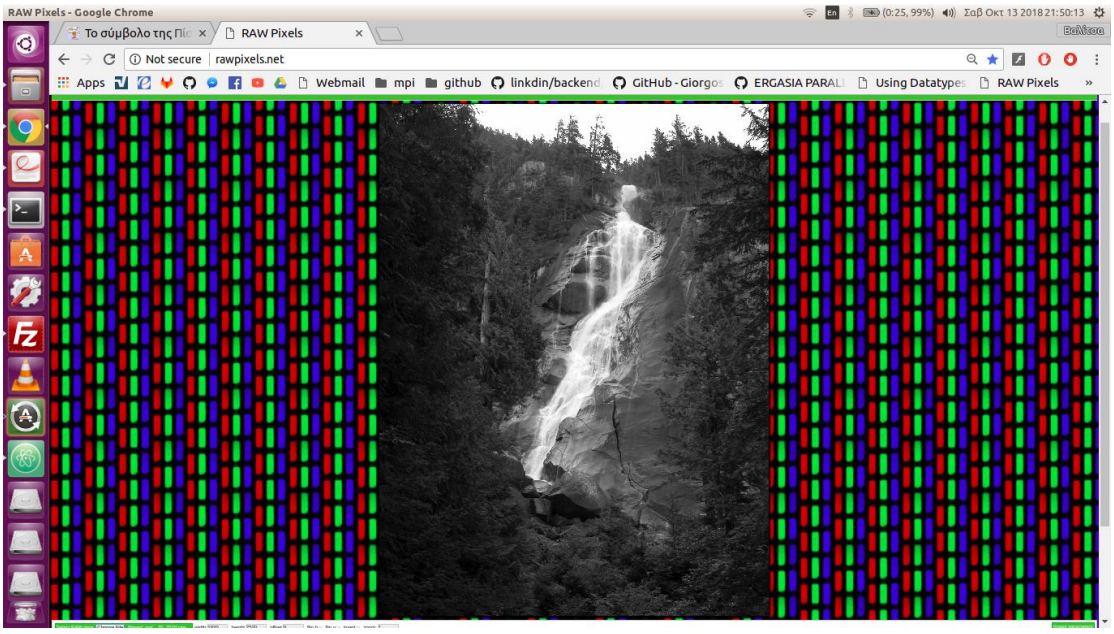
Γ) Οι πίνακες t0 και t1 είναι μονοδιάστατοι τύπου uint8_t διότι χειριζόμαστε δεδομένα που παίρνουν τιμές από 0 ως 255 και είναι το πιο αποδοτικό.

Δ) Τα MPI_Request αρχικοποιούνται σε MPI_REQUEST_NULL ώστε όταν καλείται η MPI_Wait να μην γίνεται ο έλεγχος για το αν υπάρχουν οι γείτονες και χάνεται χρόνος.

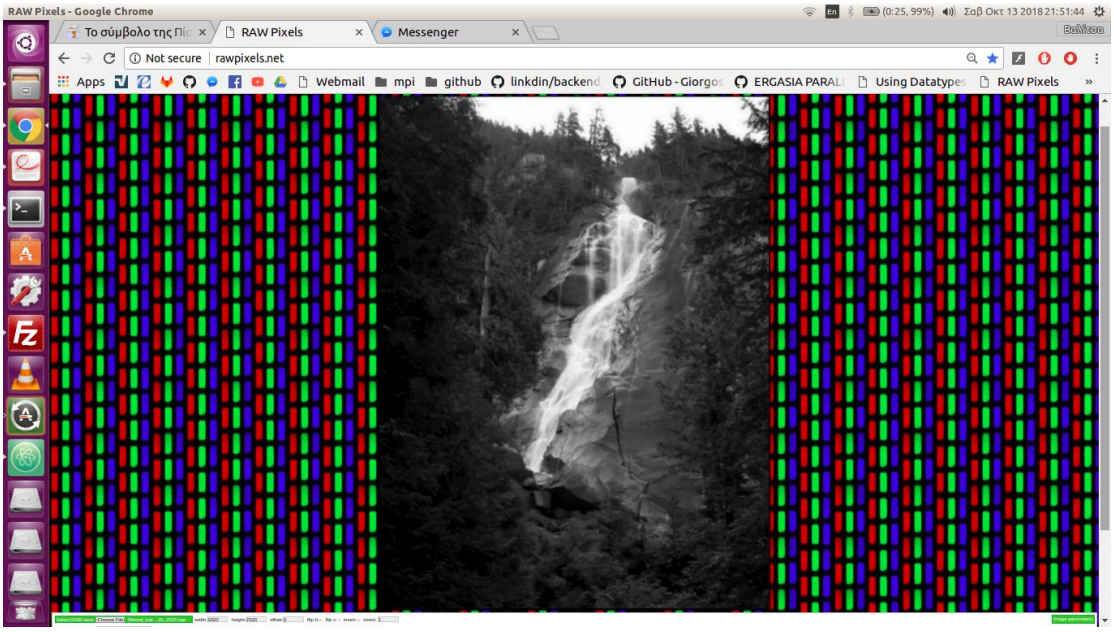
Ε) Για το OPENMP χρησιμοποιούμε την εντολή ifdef _OPENMP για να είναι ενσωματωμένο στον ίδιο κώδικα με το MPI αλλά να μεταγλωττίζεται μόνο αν δώσουμε την εντολή make openmp.

5. Ενδεικτικά αποτελέσματα

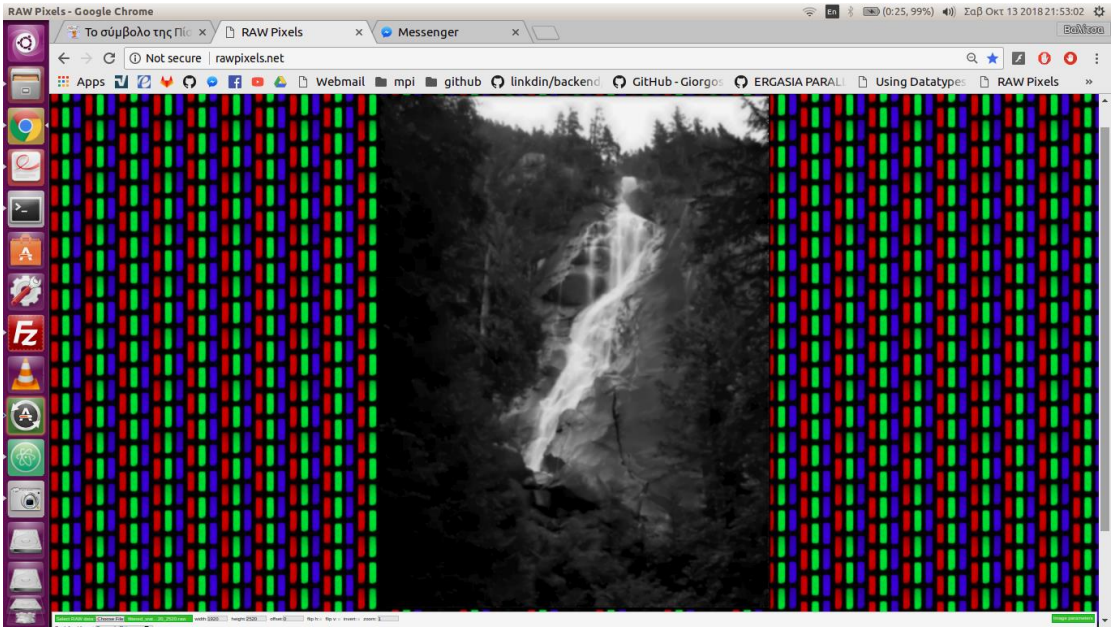
Grey Original



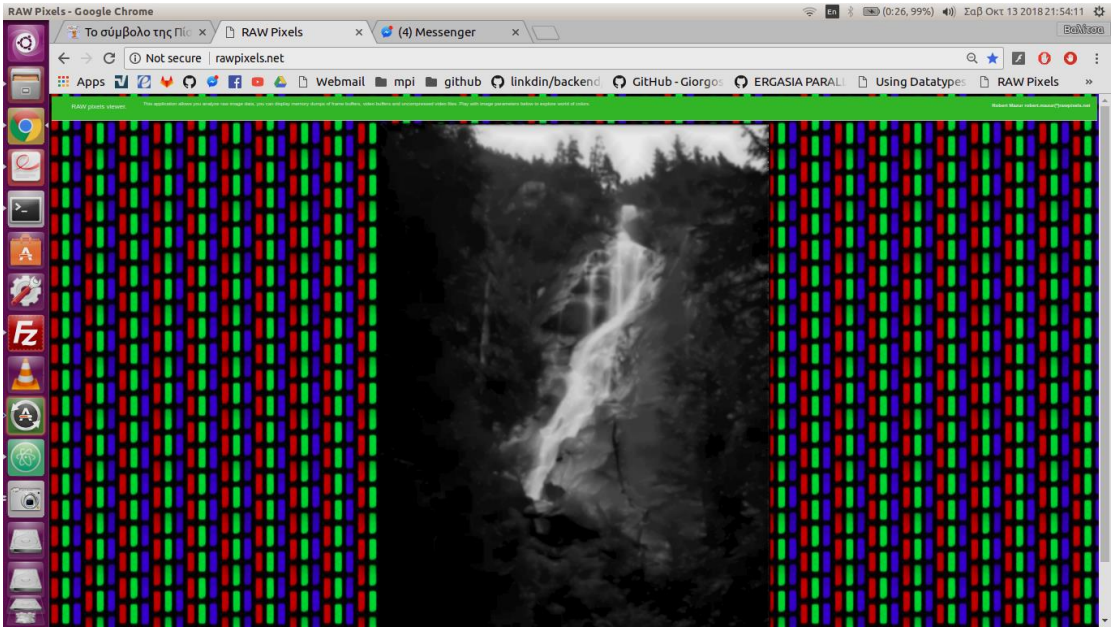
Grey 20 iterations (Gaussian blur)



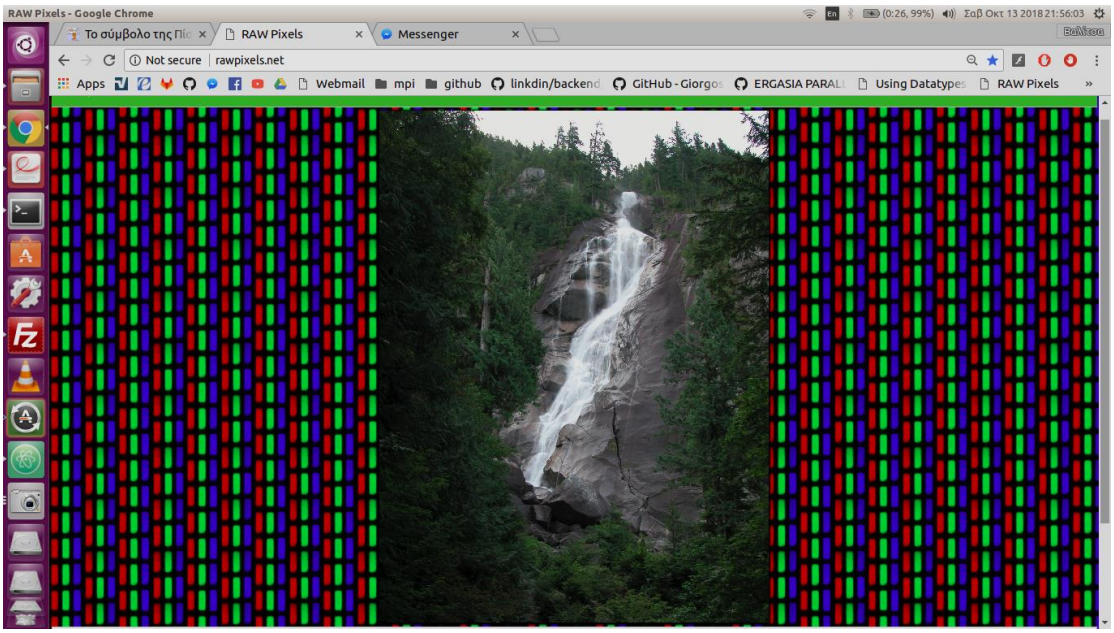
Grey 40 iterations (Gaussian blur)



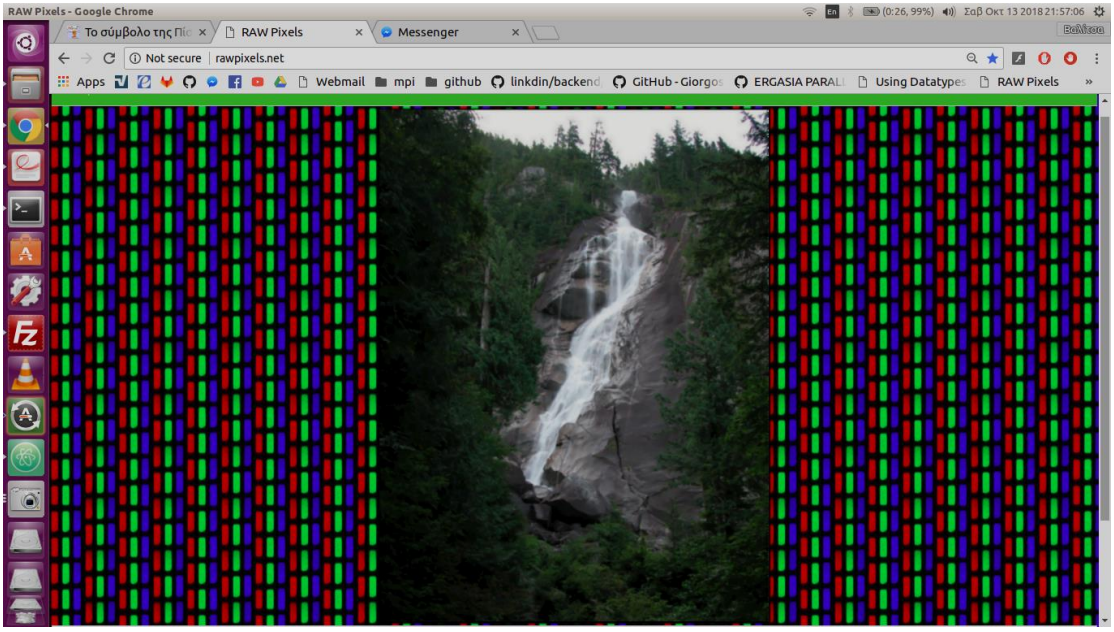
Grey 60 iterations (Gaussian blur)



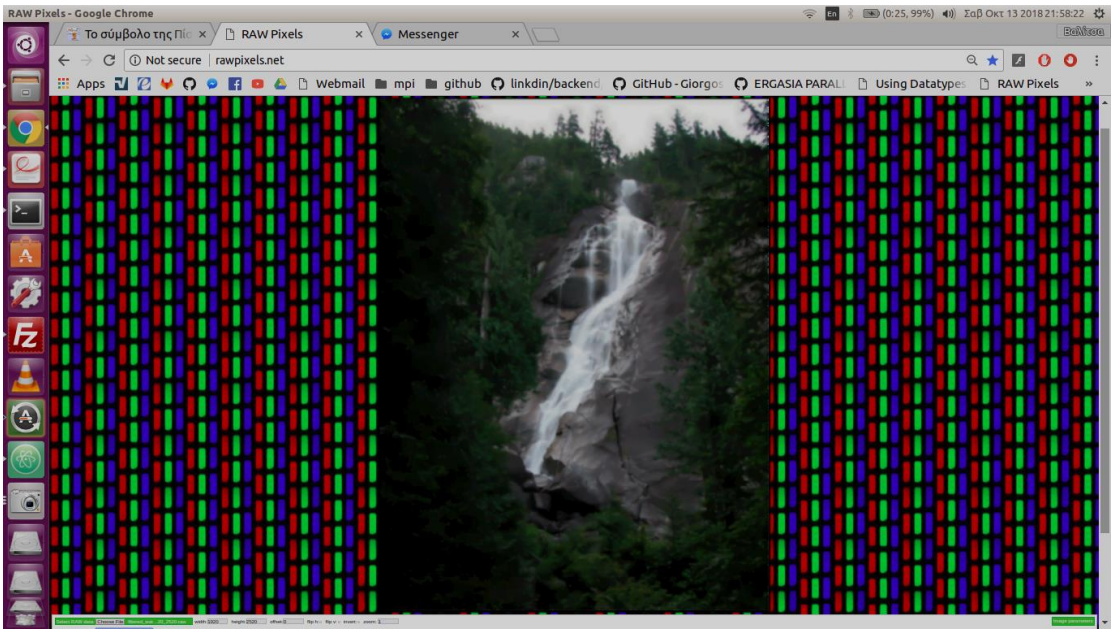
RGB original



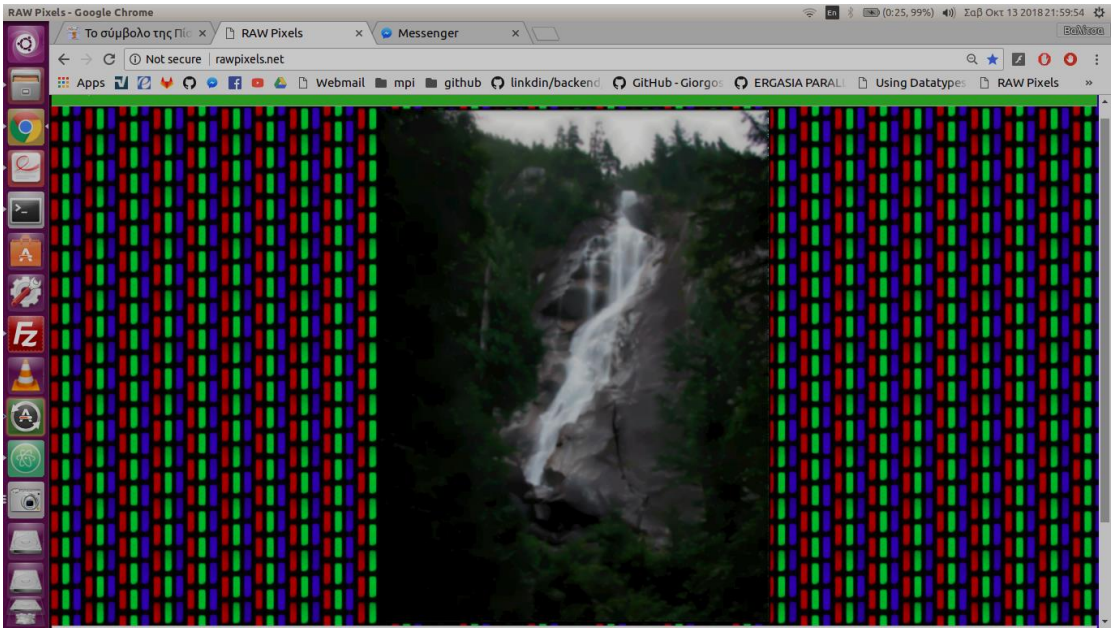
RGB 20 iterations (Gaussian blur)



RGB 40 iterations (Gaussian blur)



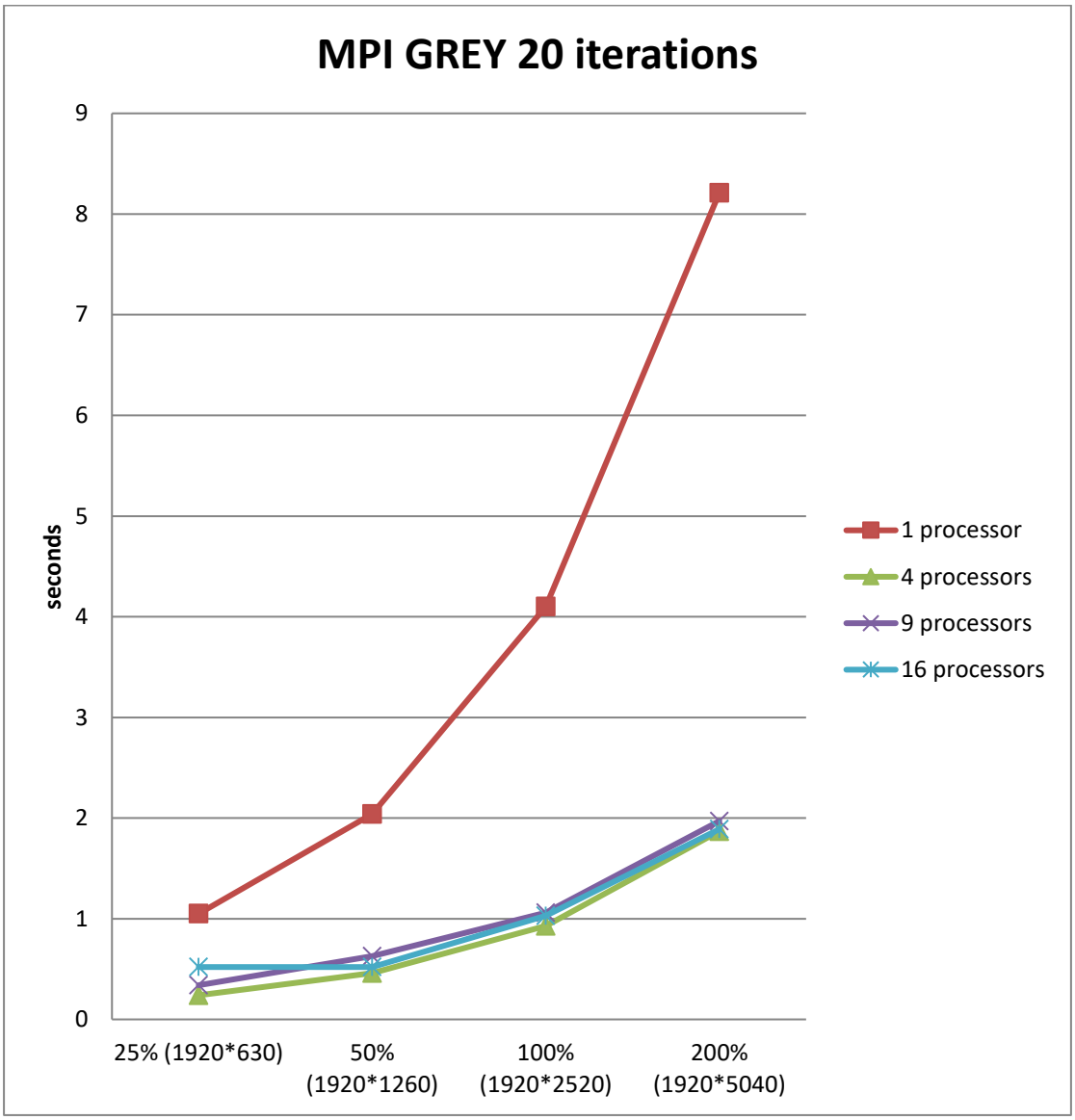
RGB 60 iterations (Gaussian blur)



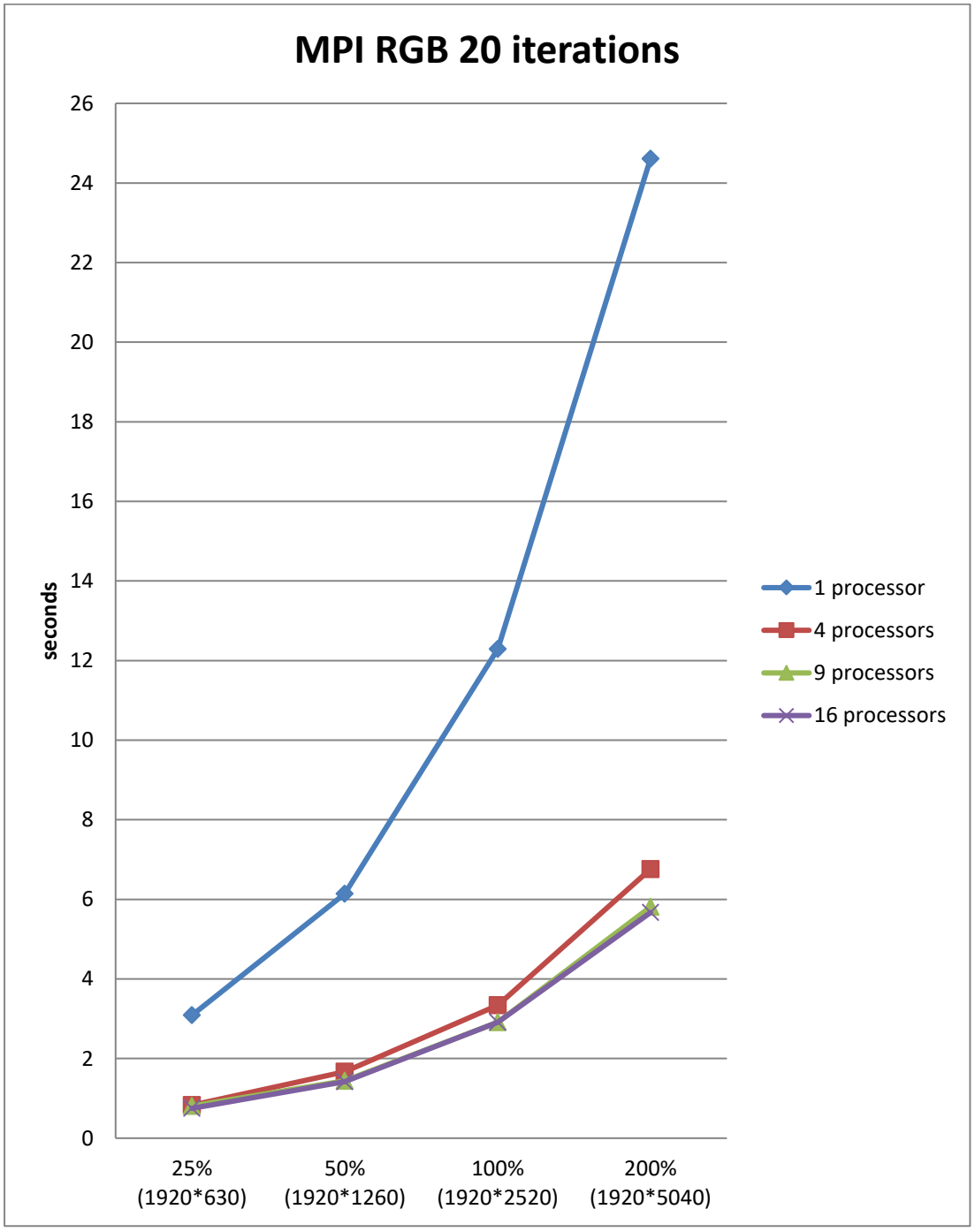
6. Μετρήσεις

Παρακάτω παρουσιάζονται κάποιοι πίνακες μαζί με τα αντίστοιχα διαγράμματα για όλα τα test που τρέξαμε.

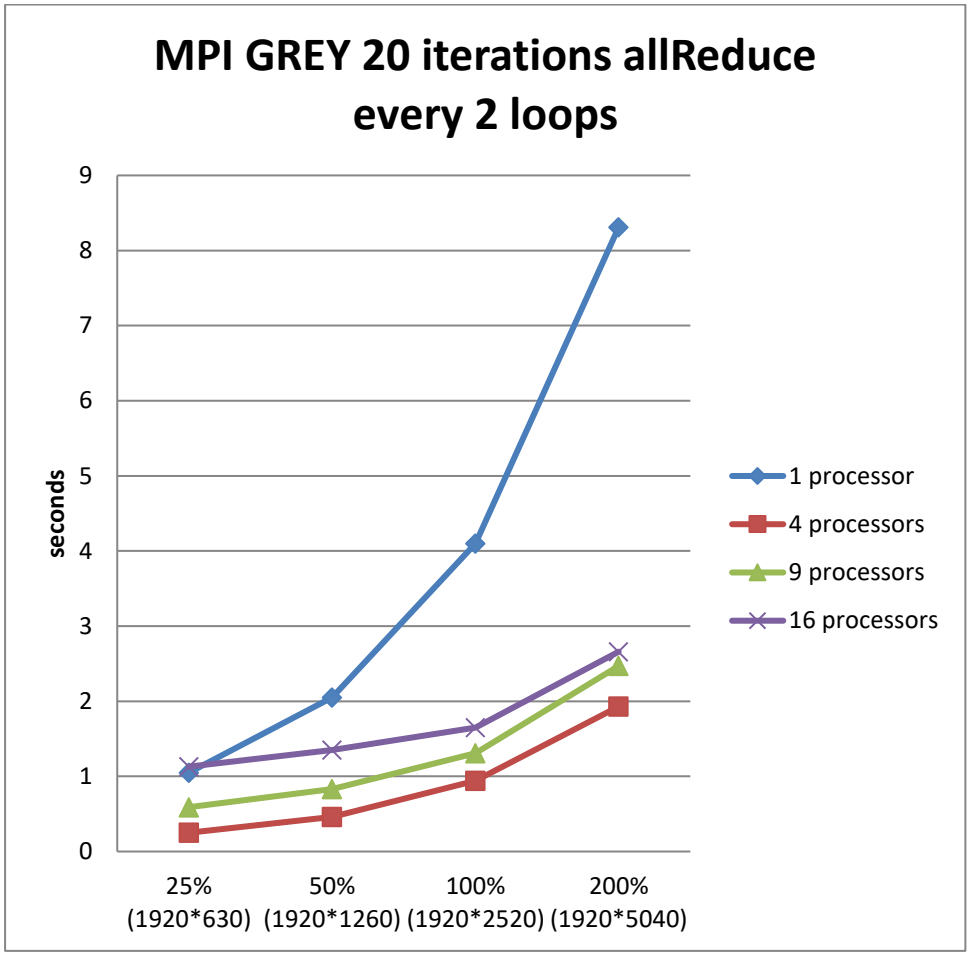
MPI-GREY 20 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	1.05	0.24	0.34	0.52
50% (1920*1260)	2.04	0.46	0.63	0.52
100% (1920*2520)	4.1	0.93	1.06	1.03
200% (1920*5040)	8.21	1.87	1.97	1.89



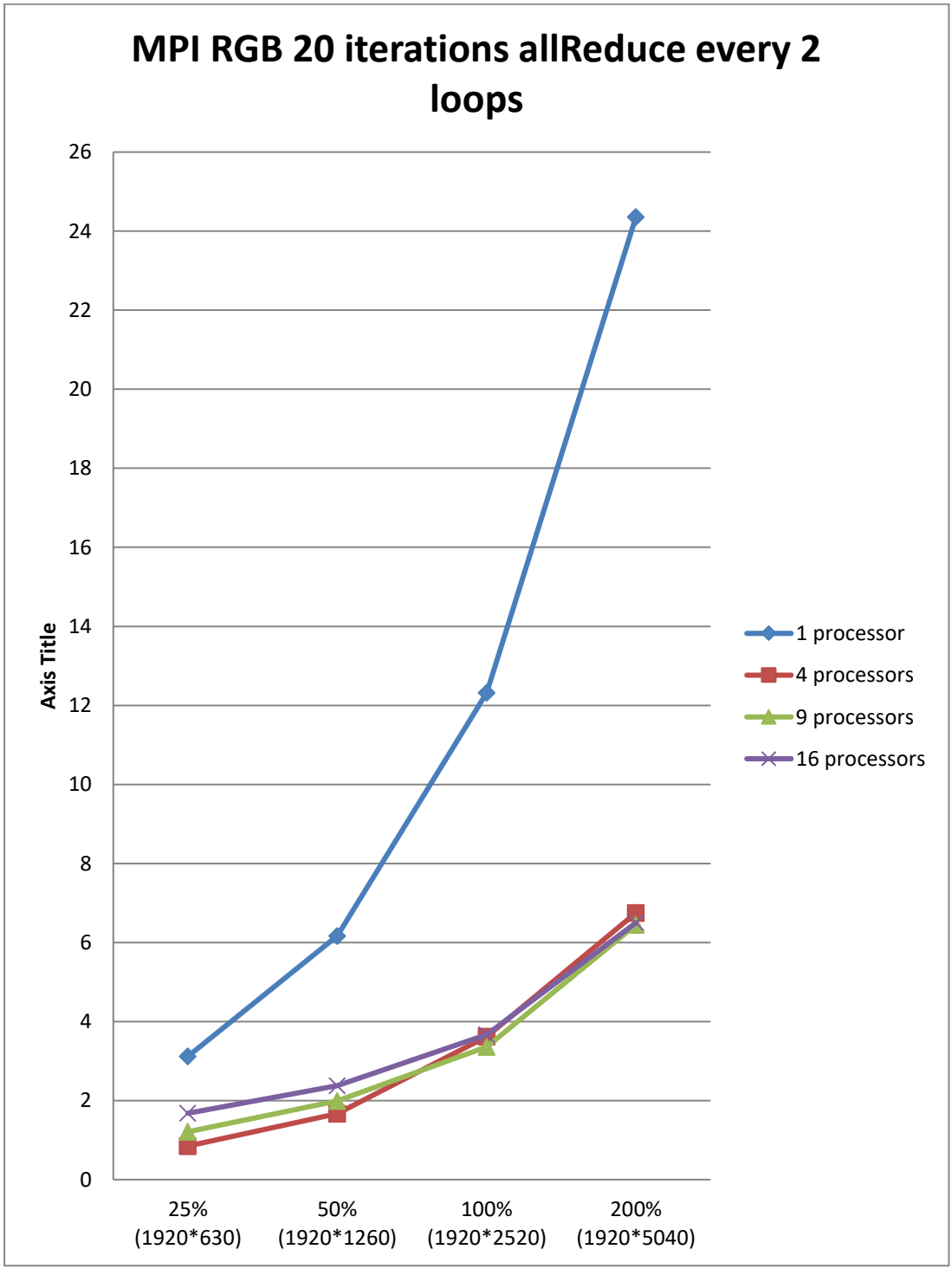
MPI-RGB 20 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	3.09	0.83	0.81	0.75
50% (1920*1260)	6.14	1.67	1.44	1.42
100% (1920*2520)	12.29	3.34	2.91	2.91
200% (1920*5040)	24.61	6.76	5.81	5.67



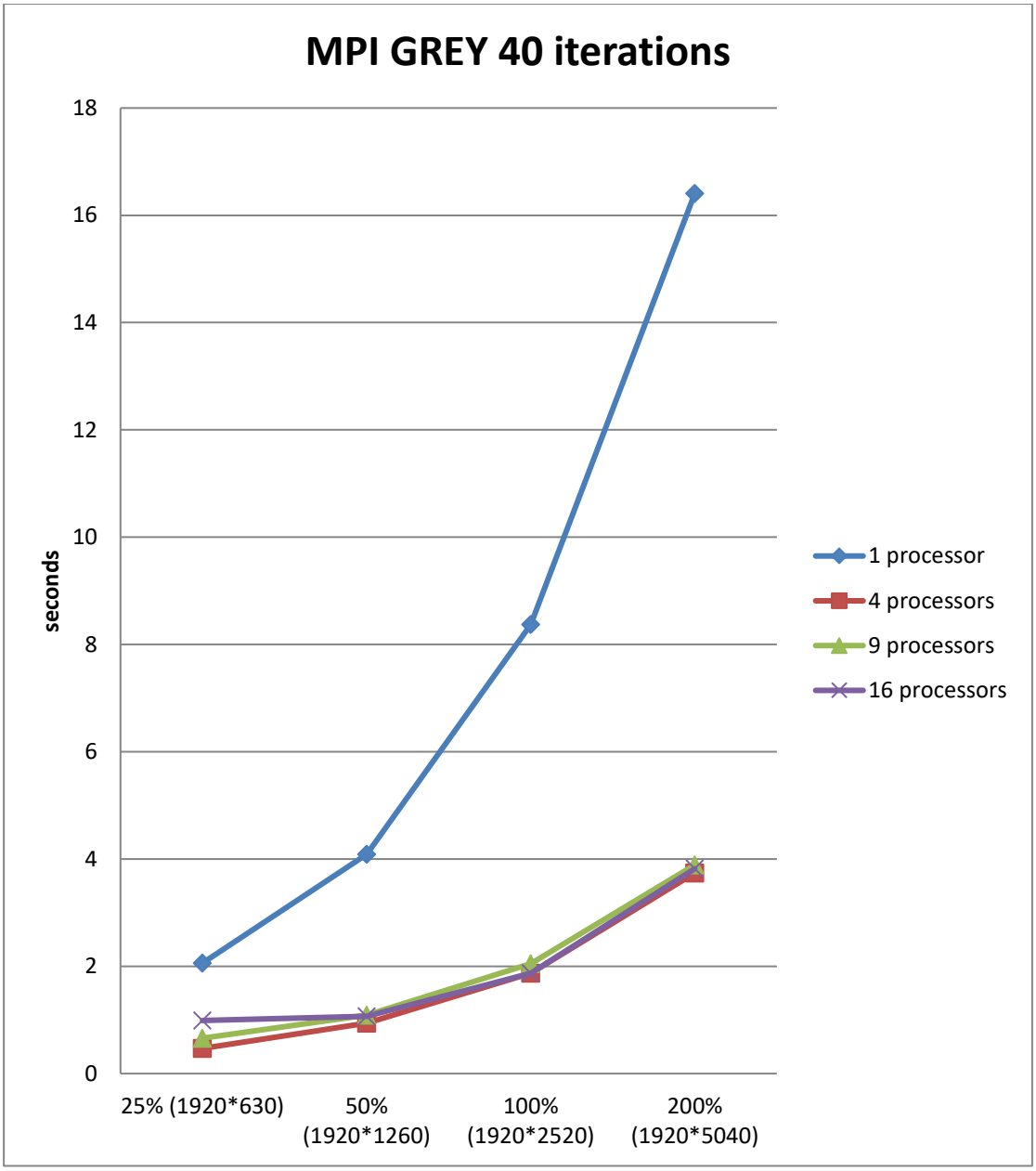
MPI-GREY 20 iterations	allReduce every 2 loops			
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	1.05	0.25	0.59	1.13
50% (1920*1260)	2.05	0.46	0.83	1.35
100% (1920*2520)	4.1	0.94	1.31	1.65
200% (1920*5040)	8.31	1.93	2.47	2.66



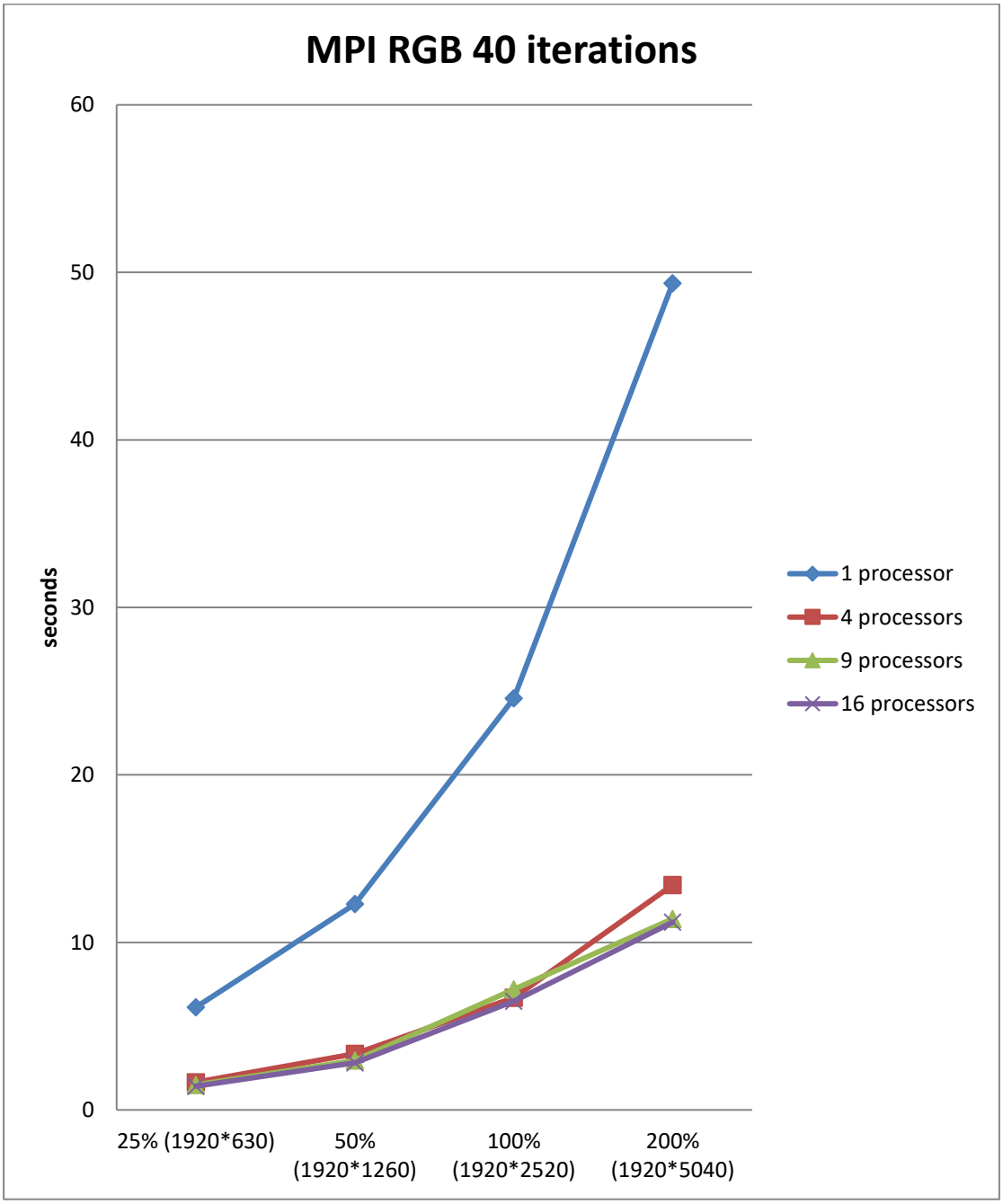
MPI-RGB 20 iterations	allReduce every 2 loops			
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	3.12	0.85	1.21	1.68
50% (1920*1260)	6.17	1.67	1.99	2.38
100% (1920*2520)	12.32	3.62	3.36	3.67
200% (1920*5040)	24.36	6.75	6.45	6.5



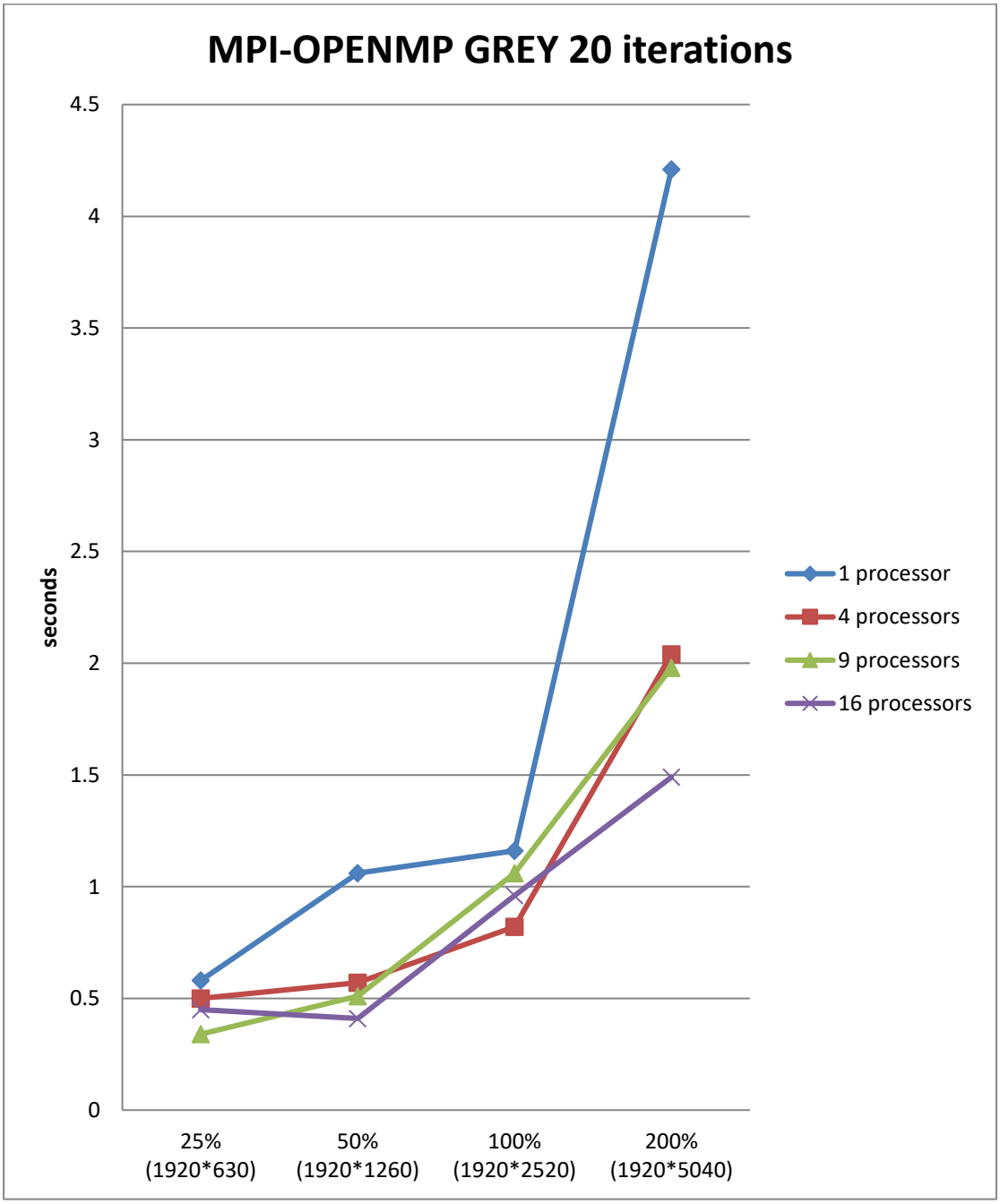
MPI-GREY 40 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	2.06	0.47	0.66	0.99
50% (1920*1260)	4.09	0.94	1.09	1.07
100% (1920*2520)	8.37	1.87	2.05	1.87
200% (1920*5040)	16.41	3.74	3.89	3.82



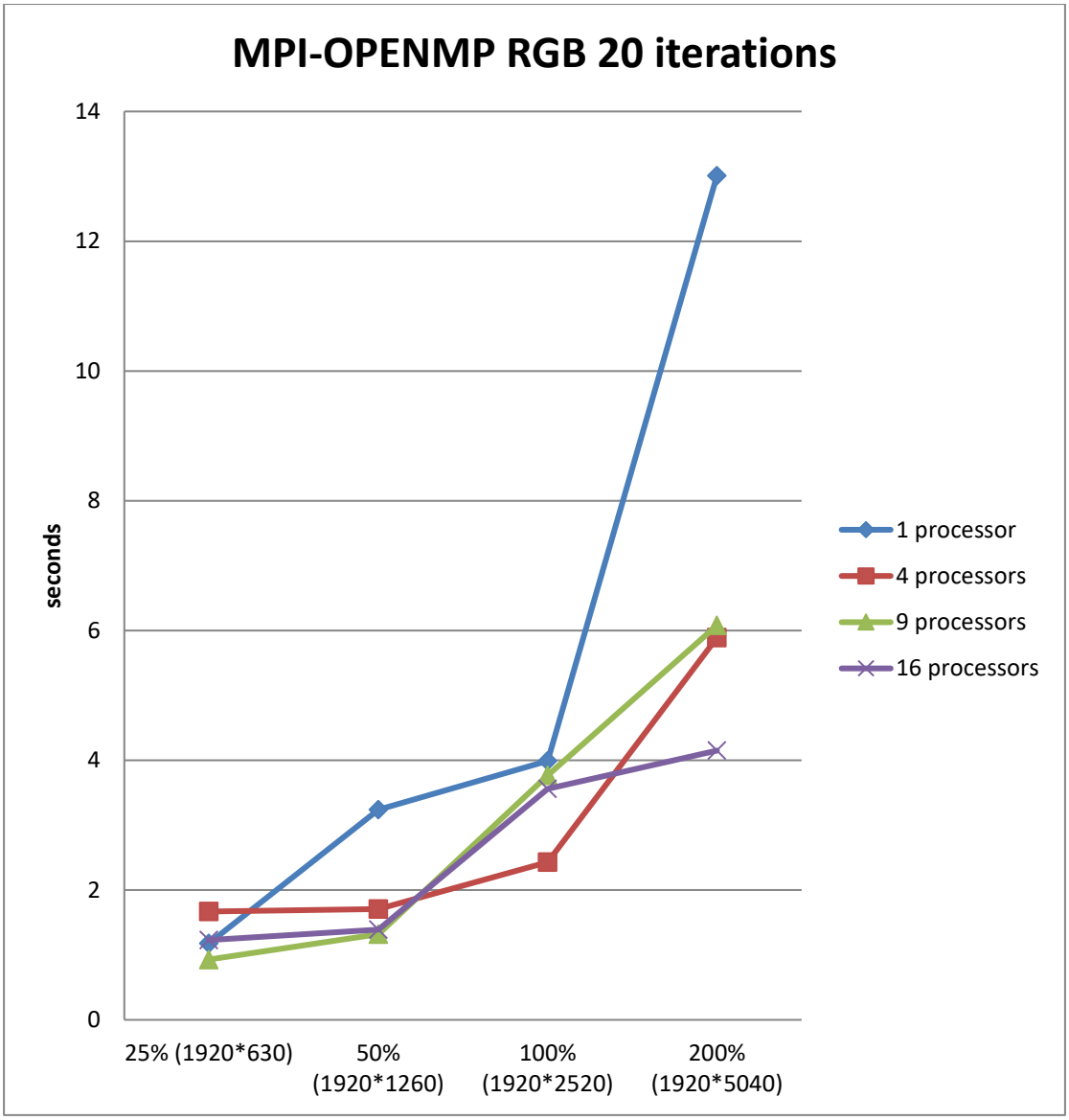
MPI-RGB 40 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	6.14	1.66	1.5	1.42
50% (1920*1260)	12.3	3.35	2.96	2.82
100% (1920*2520)	24.58	6.7	7.2	6.5
200% (1920*5040)	49.35	13.43	11.41	11.21



MPI_OPENMP-GREY 20 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	0.58	0.5	0.34	0.45
50% (1920*1260)	1.06	0.57	0.51	0.41
100% (1920*2520)	1.16	0.82	1.06	0.96
200% (1920*5040)	4.21	2.04	1.98	1.49



MPI_OPENMP-RGB 20 iterations				
Image Size / Processors	**1**	**4**	**9**	**16**
25% (1920*630)	1.18	1.67	0.93	1.23
50% (1920*1260)	3.24	1.71	1.32	1.39
100% (1920*2520)	3.99	2.43	3.77	3.56
200% (1920*5040)	13.01	5.89	6.08	4.15



7. Συμπεράσματα

Παρατηρούμε μία αναλογική αύξηση του χρόνου όσο αυξάνουμε το μέγεθος της εικόνας, όμως η μείωση όσο αυξάνουμε το πλήθος των επεξεργαστών δεν είναι αναλογική, γιατί αυξάνεται και ο χρόνος επικοινωνίας.
Φαίνεται ότι σε λίγα δεδομένα αρκούν λίγες διεργασίες γιατί όσο μεγαλώνει ο αριθμός τους, αυξάνει το κόστος κατασκευής τους.
Επίσης, βλέπουμε ότι οι χρόνοι της επεξεργασίας RGB εικόνας είναι 3 φορές μεγαλύτεροι από την αντίστοιχη ασπρόμαυροι, καθώς έχουμε τα τριπλάσια δεδομένα (red, green και blue bytes για κάθε pixel).

Βλέπουμε επίσης ότι με τη χρήση του allReduce παρατηρείται αύξηση του χρόνου λόγω παραπάνω επεξεργασίας και παραπάνω επικοινωνίας. Έτσι, ενώ θεωρητικά θα μπορούσαμε να γλιτώσουμε χρόνο αν δεν είχαμε αλλαγές στην εικόνα, στο παρόν πρόβλημα αυτό δεν ισχύει, δεν διακόπτεται η επανάληψη και δεν υπάρχει μείωση στον χρόνο.

Στα διαγράμματα hybrid (MPI + OpenMP) παρατηρούμε μία μείωση χρόνου, λόγω του ότι η επεξεργασία χωρίζεται σε threads.

Με το τελευταίο διάγραμμα βλέπουμε ότι οι χρόνοι αυξάνονται όσο αυξάνεται το πλήθος των επαναλήψεων, κι έτσι γίνεται πιο εμφανές το πλεονέκτημα της παραλληλίας.

Ευχαριστούμε πολύ!!