Εθνικό και Καποδιστριακό
ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

**ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ**

# Παράλληλα Συστήματα

MPI, MPI/OPENMP, CUDA

Convolution

Μούρης Δημήτριος ~ 1115201200114

Κωσταγιόλας Νικόλαος Στέφανος ~ 1115201100039

Αθήνα 2015

# Α) ΕΙΣΑΓΩΓΗ

## 1. Οδηγίες μεταγλώττισης και εκτέλεσης

Η μεταγλώττιση του προγράμματος γίνεται με τη χρήση της εντολής make. Για κάθε εργασία (MPI, MPI/OPENMP, CUDA) θα παραχθούν τα αντίστοιχα εκτελέσιμα αρχεία mpi_conv, mpi_omp_conv και cuda_conv.
Και για τα τρία εκτελέσιμα, οδηγίες εκτέλεσης δίνονται εκτελώντας το πρόγραμμα χωρίς παραμέτρους.
Ως απαραίτητες παράμετροι ορίζονται και με τη συγκερκιμένη σειρά, το όνομα της εικόνας που θα ανοίξει, το πλάτος, το ύψος, ο αριθμός επαναλήψεων που θα γίνει συνέλιξη στην εικόνα και τελευταία παράμετρος ένα εκ των rgb, grey που θα καθορίζεται από το αν η εικόνα είναι έγχρωμη ή ασπρόμαυρη αντίστοιχα.
Παράδειγμα εκτέλεσης:
mpirun -np 4 ./mpi_conv waterfall_grey_1920_2520.raw 1920 2520 50 grey

## 2. Παραδοχές

- ➔ Τρέχοντας το εκτελέσιμο, γίνεται έλεγχος για τα ορίσματα ότι έχουν δοθεί σωστά και αφού γίνουν οι απαραίτητοι έλεγχοι από τη master process, κάνει broadcast στις υπόλοιπες τις παραμέτρους εκτέλεσης.
- ➔ Ανάλογα με τον αριθμό των διεργασιών γίνεται διαμέριση του φόρτου ώστε η κάθε μια διαβάζει παράλληλα με τις άλλες το κομμάτι που της αναλογεί. Αντίστοιχα στο τέλος γίνεται και παράλληλο γράψιμο της καινούριας εικόνας.
- ➔ Με ελέγχους για το τύπο της εικόνας κάθε φορά εκτελείται άλλο κομμάτι κώδικα για να γίνεται σωστά η συνέλιξη.
- ➔ Ο πίνακας έχει δεσμευτεί μονοδιάστατος και όχι δισδιάστατος καθώς η κωδικοποίηση του προβλήματος αποδείχτηκε πιο “ασφαλής” και αποτελεσματική με τη χρήση αυτού του προτύπου. Επιπρόσθετα, η διαμέριση διευκολύνεται σημαντικά με τη χρήση του μονοδιάστατου μοντέλου.

# Β) ΣΧΕΔΙΑΣΜΟΣ

Το πρόβλημα της παραλληλοποίησης της συνέλιξης εικόνας απαιτεί προσεκτικό σχεδιασμό και κατανόηση του κάθε συστατικού του στοιχείου. Μια άμεση προσέγγιση, όπου προχωρούμε στο στάδιο της υλοποίησης χωρίς τον κατάλληλο σχεδιασμό του παράλληλου αλγόριθμου, οδηγεί στην ντετερμινιστική αποτυχία του εγχειρήματος σε σημαντικά θέματα όπως αυτό της κλιμάκωσης και του χρόνου εκτέλεσης. Συνεπώς, η συστηματική προσέγγιση του προβλήματος μας είναι απαραίτητη.

Σε αυτήν την κατεύθυνση υπάρχει η μεθοδολογία του Foster, η οποία ορίζει τέσσερα στάδια σχεδιασμού. Ονομαστικά είναι το στάδιο του partitioning, communication, agglomeration και mapping.

## 1. Partitioning

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. Σύμφωνα με την μεθοδολογία, στα πρώτα στάδια πρέπει να επιλέγουμε πιο επιθετικούς διαμερισμούς.

Σαν πρώτη επιλογή κάθε διεργασία είναι υπεύθυνη για ένα κομμάτι με υποδιαιρεμένες διαστάσεις της αρχικής εικόνας. Σαν να έχει δηλαδή, η κάθε διεργασία μια δικιά της ξεχωριστή εικόνα. Σε αυτή την απόφαση μας οδήγησε η ανάγκη για ιδιαίτερη έμφαση στην ισοκατανομή του φορτίου ανάμεσα στις διεργασίες, εφόσον η κάθε διεργασία θα έχει τη μνήμη της όπου θα περιέχεται το αντίστοιχο κομμάτι της εικόνας.

Με αυτό τον τρόπο:

- ➔ έγινε εφικτή η δημιουργία εργασιών που ξεπερνούν τον αριθμό των διαθέσιμων επεξεργαστών,
- ➔ αποφεύγεται η περιττή εκτέλεση υπολογισμών,
- ➔ οι εργασίες έχουν τον ίδιο φόρτο εργασίας,

- ➔ έχουμε αριθμό εργασιών που κλιμακώνουν με το μέγεθος του προβλήματος.
- ➔ και έπιτυγχανεται μικρή κατανάλωση μνήμης μέσω της υλοποίησης parallel IO αφού η κάθε διεργασία διαβάζει το κομμάτι της εικόνας που της αναλογεί. Ταυτόχρονα αυτό είναι πολύ πιο γρήγορο από άποψη χρόνου.

Πιο αναλυτικά, κατά την εκκίνηση του προγράμματος ο χρήστης επιλέγει το πλήθος των διεργασιών για το οποίο θα εκτελεστεί το πρόγραμμα. Βάσει ενός αλγορίθμου (ο οποίος περιγράφεται στη συνάρτηση divide_rows υπολογίζεται ο βέλτιστος τρόπος με τον οποίο θα κατανεμηθεί ο πίνακας ανάμεσα στις διεργασίες).

## 2. Communication

Όπως ήδη αναφέρθηκε οι διεργασίες μοιράζονται στο πλέγμα με τέτοιο τρόπο ώστε να είναι η μία δίπλα στην άλλη με αύξουσα σειρά. Αυτό διευκολύνει κάθε διεργασία στην ανεύρεση των γειτονικών διεργασιών της. Ιδιαίτερη περίπτωση αποτελούν οι διεργασίες που βρίσκονται στην περιφέρεια του πλέγματος. Σε αυτή την περίπτωση γειτονικές διεργασίες θεωρούνται αυτές που βρίσκονται στην κατάλληλη εκτός ορίων απέναντι πλευρά.

Προχωρώντας τώρα σε θέματα κώδικα σαν επιλογή έχουμε αποφασίσει να χρησιμοποιήσουμε non-blocking αποστολές και λήψεις μηνυμάτων μπορώντας με αυτόν τον τρόπο να κάνουμε 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 μια εργασία όσο περιμένει να έρθουν τα γειτονικά κελιά του πίνακα. Όταν ολοκληρωθεί η εργασία αυτή αν έχουν έρθει τα δεδομένα που περιμένει προχωράει στην εκτέλεση των απαιτούμενων νέων εργασιών αλλιώς εξακολουθεί να περιμένει την λήψη των απαιτούμενων δεδομένων.

Για την αποστολή και την λήψη αποφασίσαμε να γίνουν με την βοήθεια Datatypes (vector, contiguous) δεδομένων όπου ήταν δυνατό (up,right,down,left) για την αποφυγή διάφορων κινδύνων και την καλύτερη διαχείριση αυτών.

## 3. Abstract

Γενικότερα η υλοποίηση μας ακολούθησε το παρακάτω μοτίβο:

```
for loop
        ISend
        IRecv
        Inner Data Computations
        Wait(RRecv)
        Outer Data Computations
        Wait(RSend)
end for
```

# Γ) ΠΕΡΑΙΤΕΡΩ ΛΕΠΤΟΜΕΡΙΕΣ

## 1. MPI/OPENMP

Το OpenMP είναι μια Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface, API) το οποίο δημιουργήθηκε από κατασκευαστές υλικού και λογισμικού. Τα αρχικά αντιστοιχούν στη φράση "Open Specifications for Multi Processing"]. Το πρότυπο αυτό προσφέρει στον προγραμματιστή ένα σύνολο από οδηγίες, οι οποίες ενσωματώνονται στον κώδικα ενός προγράμματος, και έτσι μπορεί ο μεταγλωττιστής να επιτύχει παραλληλισμό στο πρόγραμμα αυτό, σύμφωνα πάντα με τις οδηγίες και παραμέτρους που έχει θέσει ο προγραμματιστής.

Στο κομμάτι του OpenMP σκοπός είναι κομμάτια που εκτελούνται τοπικά να παραλληλοποιηθούν με τη χρήση thread μέσω του OpenMP ώστε να βελτιώνεται η ήδη υλοποιημένη εκδοχή του MPI. Συγκεκριμένα στην άσκηση αυτή το σημείο στο οποίο κρίναμε πως αυτή η υλοποίηση θα βοηθούσε από άποψη χρόνου είναι το σημείο στο οποίο διατρέχουμε με επανάληψη τον τοπικό πίνακα κάθε διεργασίας.

Επίσης υπήρχε η δυνατότητα παραλληλοποίησης με χρήση του OpenMP σε άλλο σημείο της υλοποίησης. Συγκεκριμένα στην γενική επανάληψη για τον αριθμό των φορών που θα γίνει η συνέλιξη. Αυτό βέβαια δημιουργούσε πρόβλημα στην ανταλλαγή δεδομένων ανάμεσα στις διεργασίες, αφού μετά υπήρχαν δύο λύσεις. Να σπάσουν τα send/receive και να γίνουν από νήματα ή να τα κάνει μόνο το master thread. Η δεύτερη επιλογή είχε προφανώς bottleneck, η πρώτη δεν ήταν τοσο εύκολη και συγκριτικά με τους υπολογισμούς στα inner data τα send/receive δεν ήταν τόσο πολλά οπότε δεν θα υπήρχε το επιθυμητό speed up. Αυτοί οι δύο λόγοι μας ώθησαν να κάνουμε παράλληλο το κομμάτι υπολογισμού των inner data.

## 2. CUDA

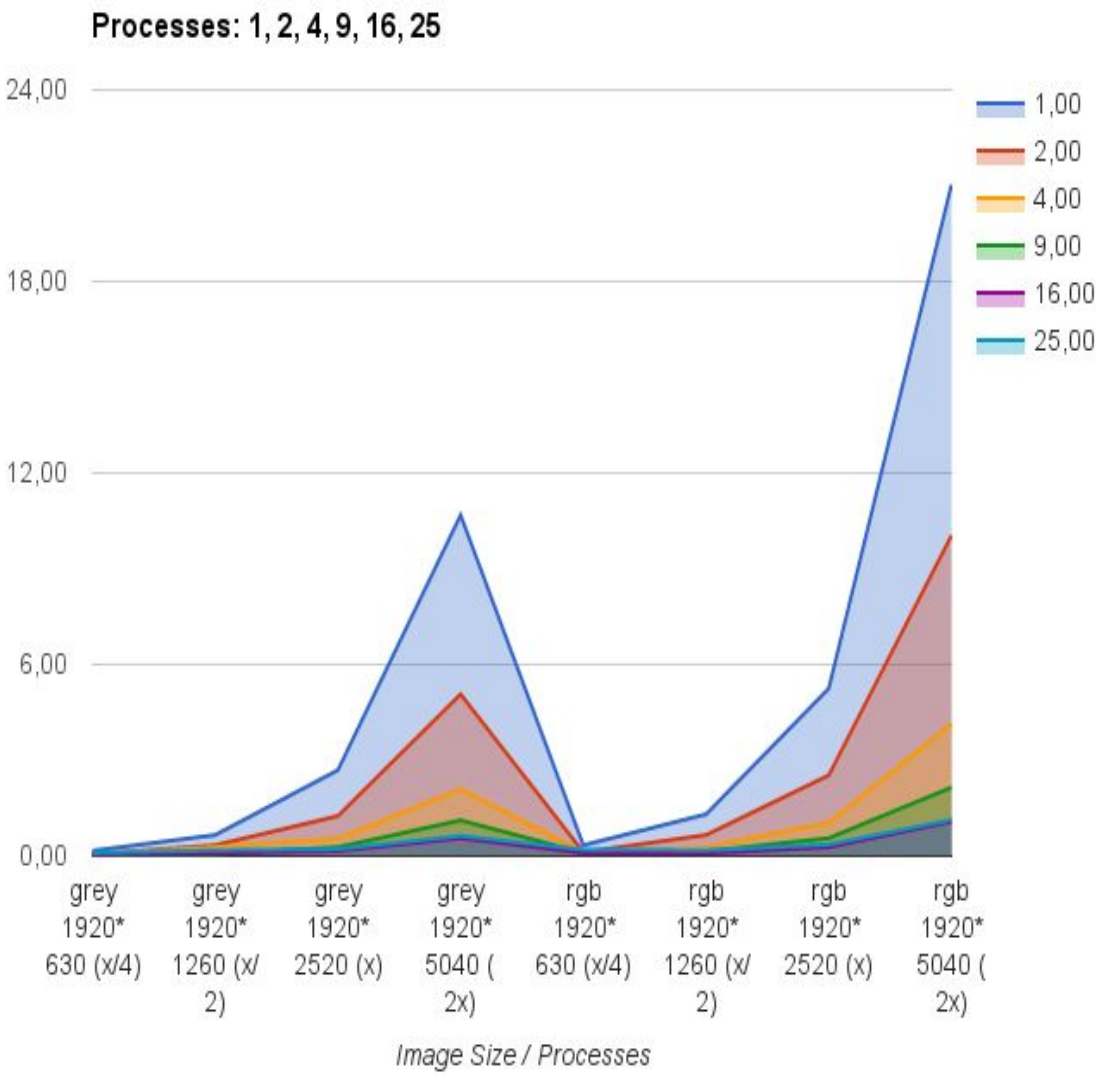
Το πρόγραμμα σε cuda βασίστηκε στις προηγούμενες υλοποιήσεις (mpi) και έγιναν οι απαραίτητες μετατροπές. Αρχικά δεσμεύσεις χώρου στη gpu, αντιγραφές πινάκων (host -> device & device -> host), δημιουργία και κλήση της συνάρτησης __global__ και τέλος τα απαραίτητα free. Δεν υπάρχει κάποια αξιοσημείωτη τεχνική λεπτομέρεια αφού ο κώδικας πάνω κάτω είναι ίδιος, παρά μόνο ότι το “τεστάρισμα” και οι μετρήσεις χρόνου έγιναν σε προσωπικό υπολογιστή με Linux και εγκατεστημένο Cuda. Η κάρτα γραφικών στην οποία μετρήθηκαν οι χρόνοι είναι η ASUS GEFORCE GTX 760 OC 2GB.

# Δ) ΜΕΤΡΗΣΕΙΣ ΑΠΟΔΟΣΗΣ ΚΑΙ ΕΠΙΤΑΧΥΝΣΗΣ

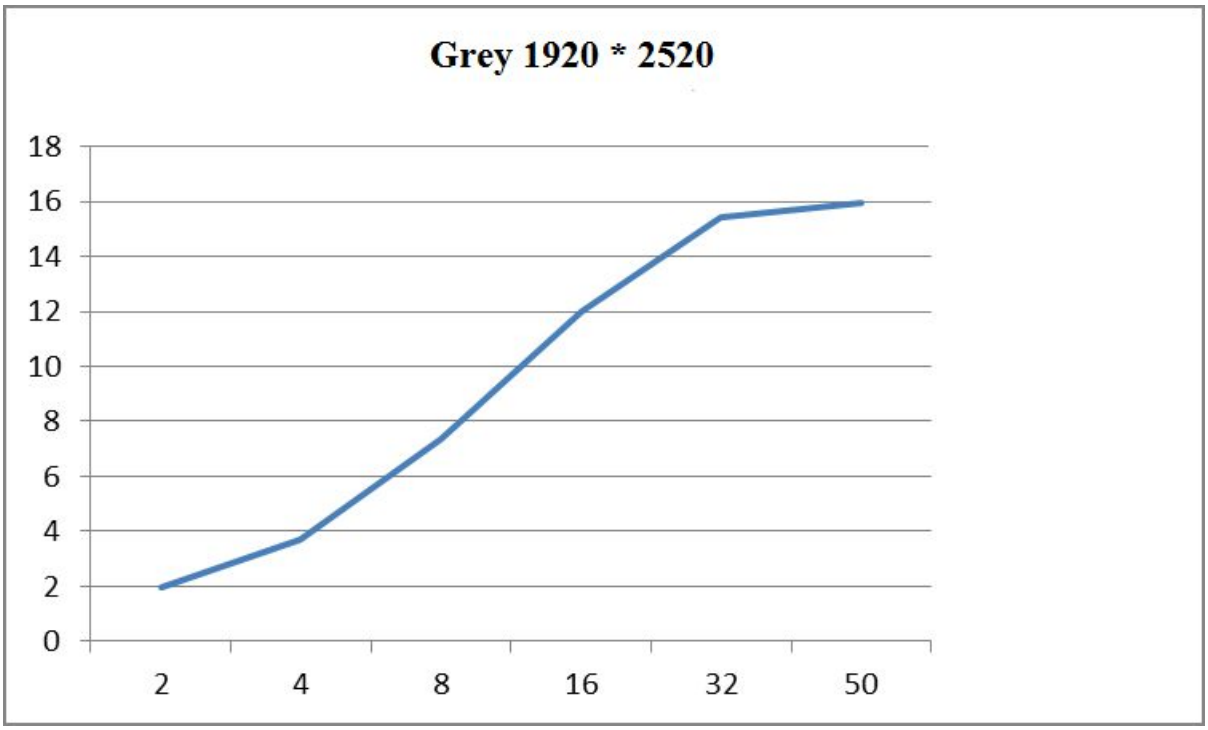
## 1. MPI

| MPI | 20 Iterations | | | | | |
|---|---|---|---|---|---|---|
| Image Size / Processes | 1,00 | 2,00 | 4,00 | 9,00 | 16,00 | 25,00 |
| grey 1920*630 (x/4) | 0,17 | 0,06 | 0,06 | 0,05 | 0,05 | 0,10 |
| grey 1920*1260 (x/2) | 0,66 | 0,34 | 0,28 | 0,07 | 0,10 | 0,18 |
| grey 1920*2520 (x) | 2,69 | 1,26 | 0,55 | 0,28 | 0,16 | 0,22 |
| grey 1920*5040 (2x) | 10,68 | 5,07 | 2,10 | 1,12 | 0,56 | 0,63 |
| rgb 1920*630 (x/4) | 0,34 | 0,12 | 0,12 | 0,10 | 0,10 | 0,20 |
| rgb 1920*1260 (x/2) | 1,31 | 0,66 | 0,28 | 0,15 | 0,09 | 0,18 |
| rgb 1920*2520 (x) | 5,25 | 2,53 | 1,05 | 0,56 | 0,28 | 0,36 |
| rgb 1920*5040 (2x) | 21,03 | 10,04 | 4,15 | 2,15 | 1,08 | 1,13 |

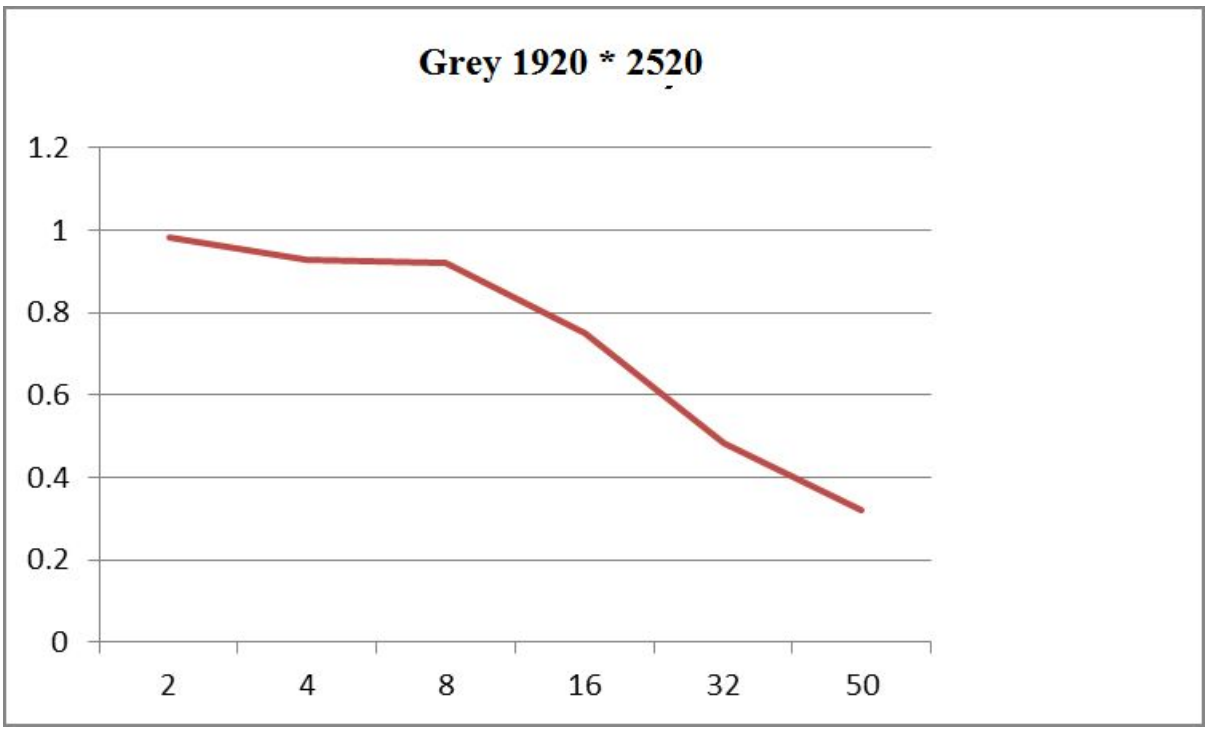
**Δ.1.1 Μετρήσεις χρόνου για MPI**



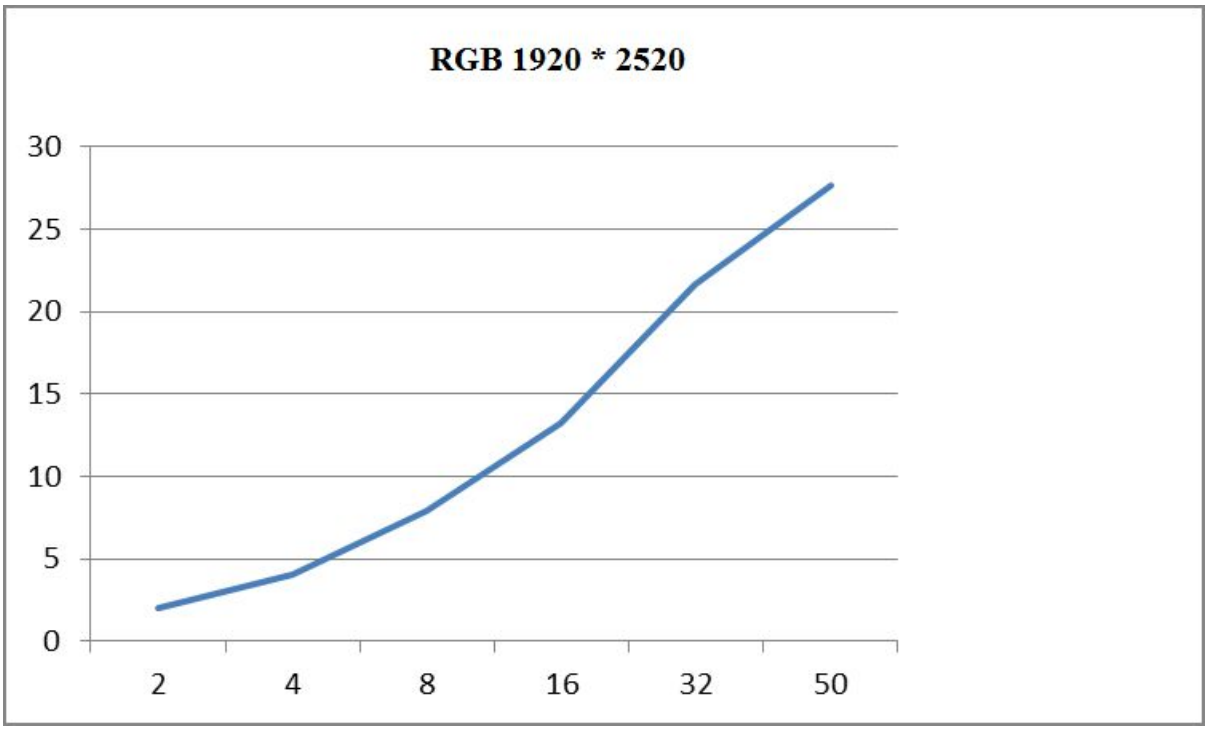

**Δ.1.2 Σχηματική σύγκριση μετρήσεων MPI ως προς αριθμό διεργασιών**




**Δ.1.3 Speed Up = Tserial / Tparallel**




**Δ.1.4 Efficiency = Speed Up / Proc**




**Δ.1.5 Speed Up = Tserial / Tparallel**

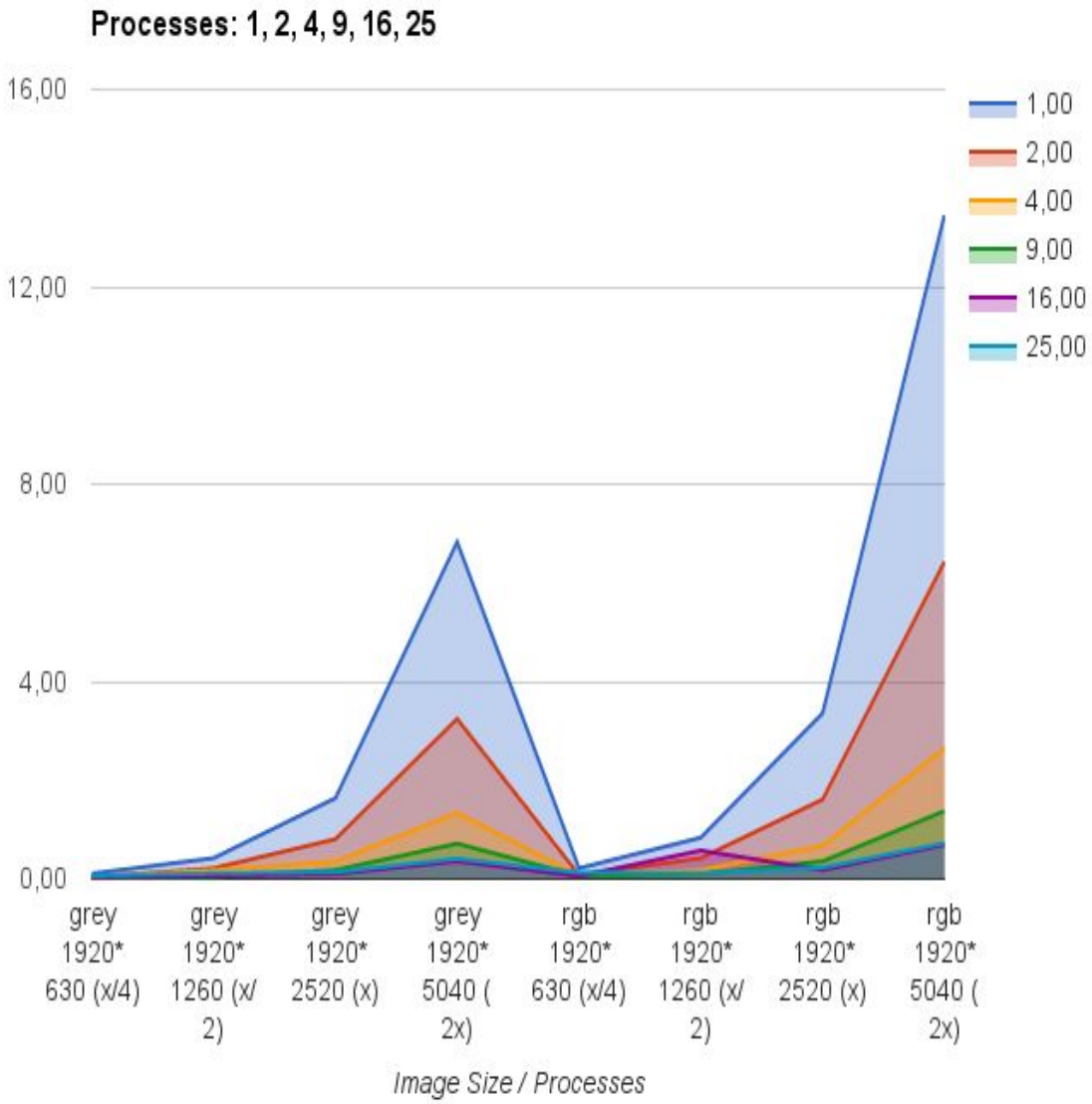

**Δ.1.6 Efficiency = Speed Up / Proc**

## 2. MPI/OMP

| MPI & OpenMP | 20 Iterations | | | | | |
|---|---|---|---|---|---|---|
| Image Size / Processes | 1,00 | 2,00 | 4,00 | 9,00 | 16,00 | 25,00 |
| grey 1920*630 (x/4) | 0,11 | 0,04 | 0,04 | 0,03 | 0,03 | 0,06 |
| grey 1920*1260 (x/2) | 0,42 | 0,22 | 0,18 | 0,04 | 0,06 | 0,12 |
| grey 1920*2520 (x) | 1,64 | 0,81 | 0,35 | 0,18 | 0,10 | 0,14 |
| grey 1920*5040 (2x) | 6,83 | 3,24 | 1,34 | 0,72 | 0,36 | 0,41 |
| rgb 1920*630 (x/4) | 0,22 | 0,08 | 0,07 | 0,06 | 0,06 | 0,13 |
| rgb 1920*1260 (x/2) | 0,84 | 0,42 | 0,18 | 0,10 | 0,58 | 0,12 |
| rgb 1920*2520 (x) | 3,36 | 1,61 | 0,67 | 0,36 | 0,18 | 0,23 |
| rgb 1920*5040 (2x) | 13,45 | 6,43 | 2,66 | 1,38 | 0,69 | 0,72 |

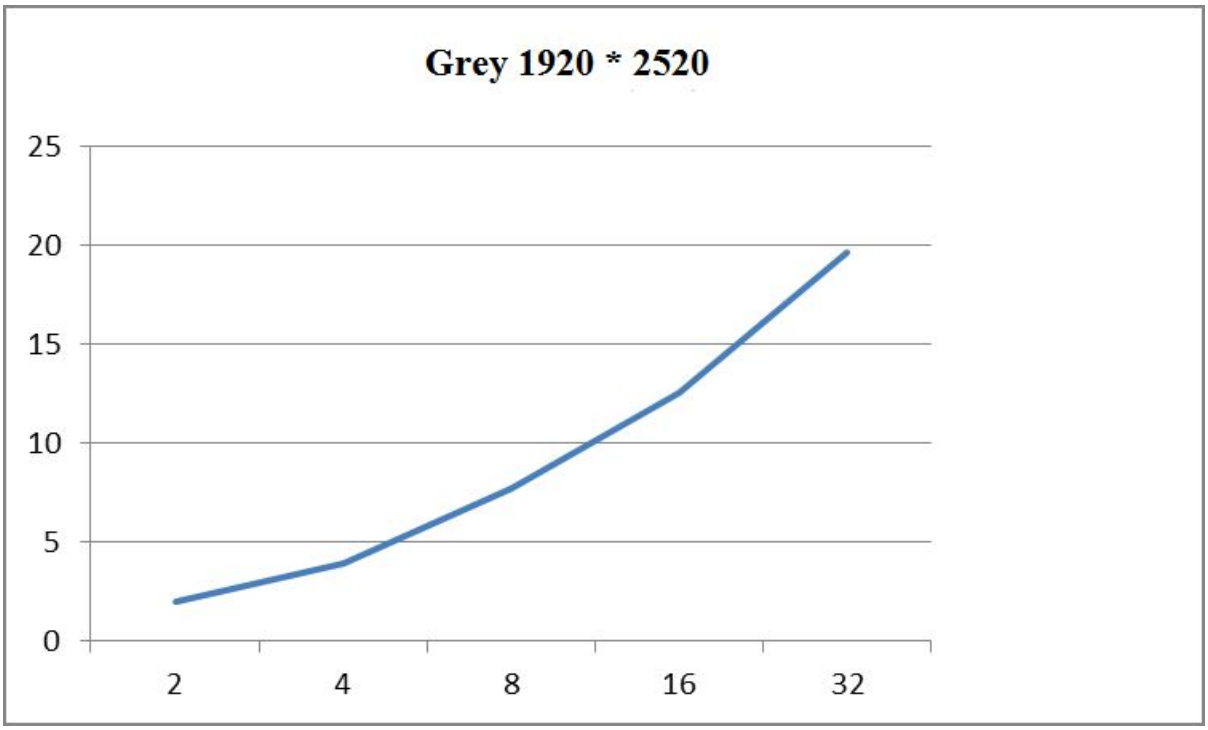
**Δ.2.1 Μετρήσεις χρόνου για MPI + OpenMP**

Φαίνεται ότι σε λίγα δεδομένα αρκούν λίγες διεργασίες γιατί όσο μεγαλώνει ο αριθμός τους, αυξάνει το κόστος κατασκευής τους.

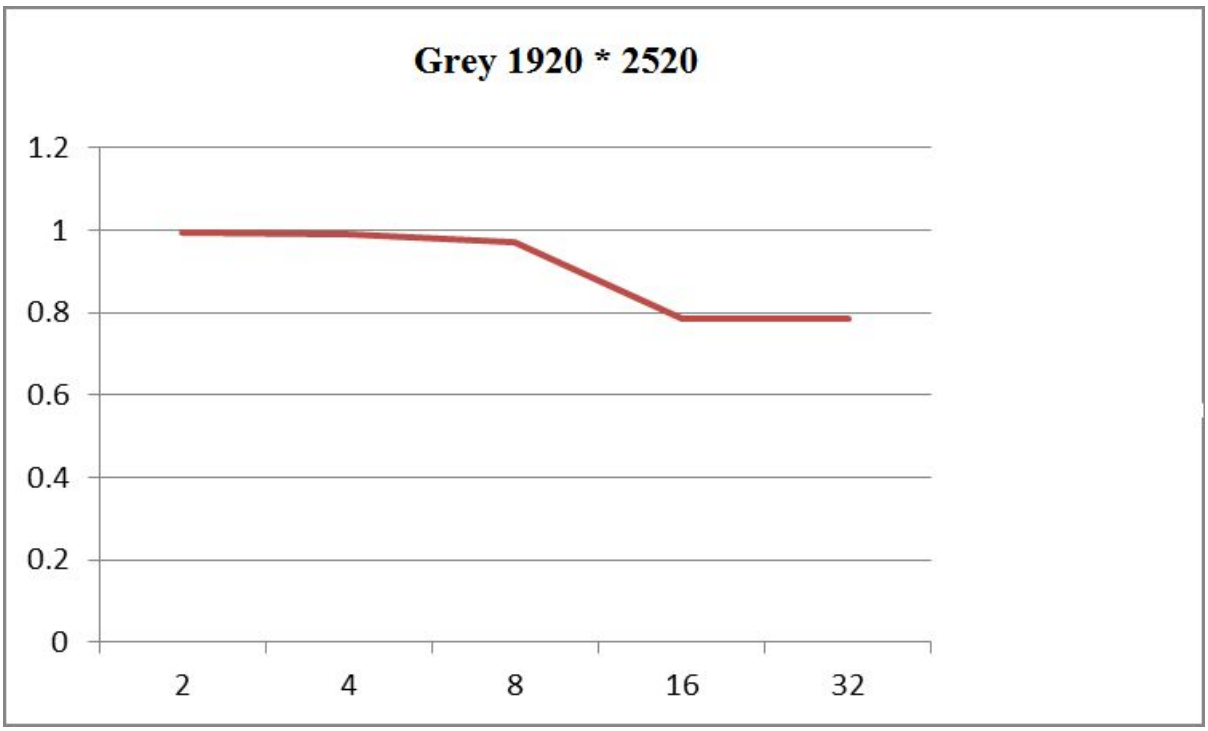



**Δ.2.2 Σχηματική σύγκριση μετρήσεων MPI + OpenMP ως προς αριθμό διεργασιών**

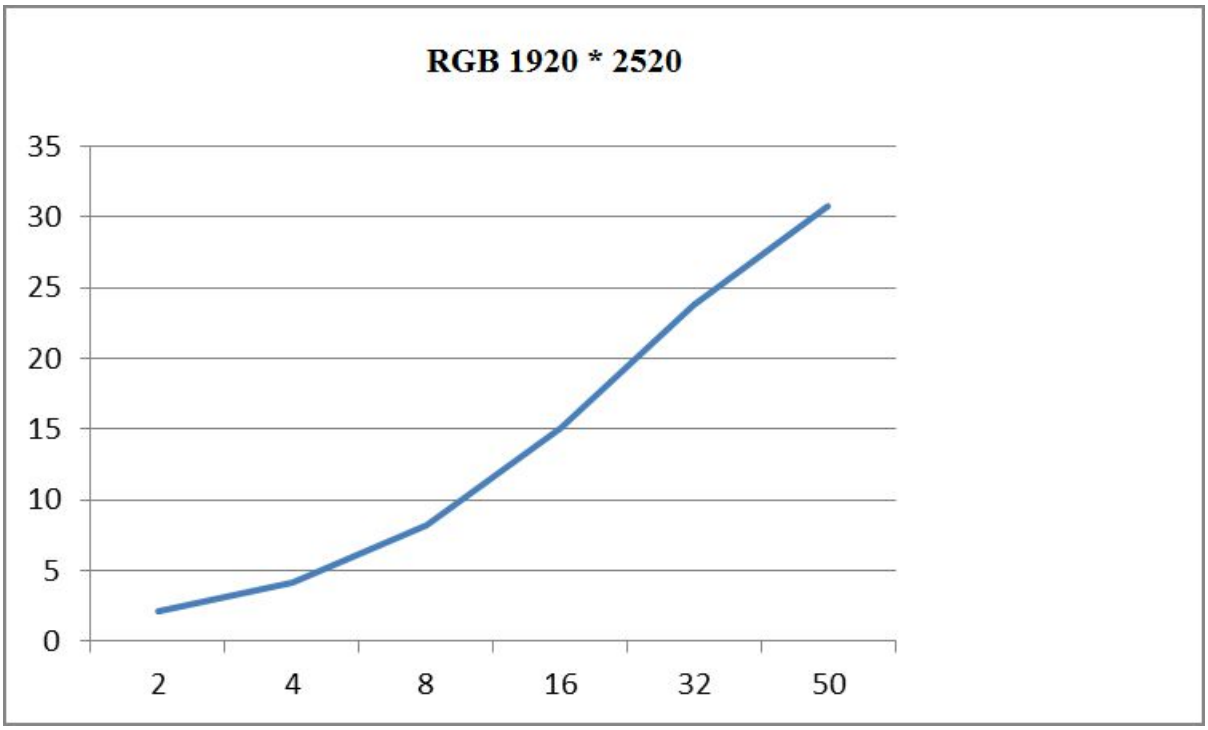
Όμοια κι εδώ με το MPI, φαίνεται ότι σε λίγα δεδομένα αρκούν λίγες διεργασίες γιατί όσο μεγαλώνει ο αριθμός τους, αυξάνει το κόστος κατασκευής τους.



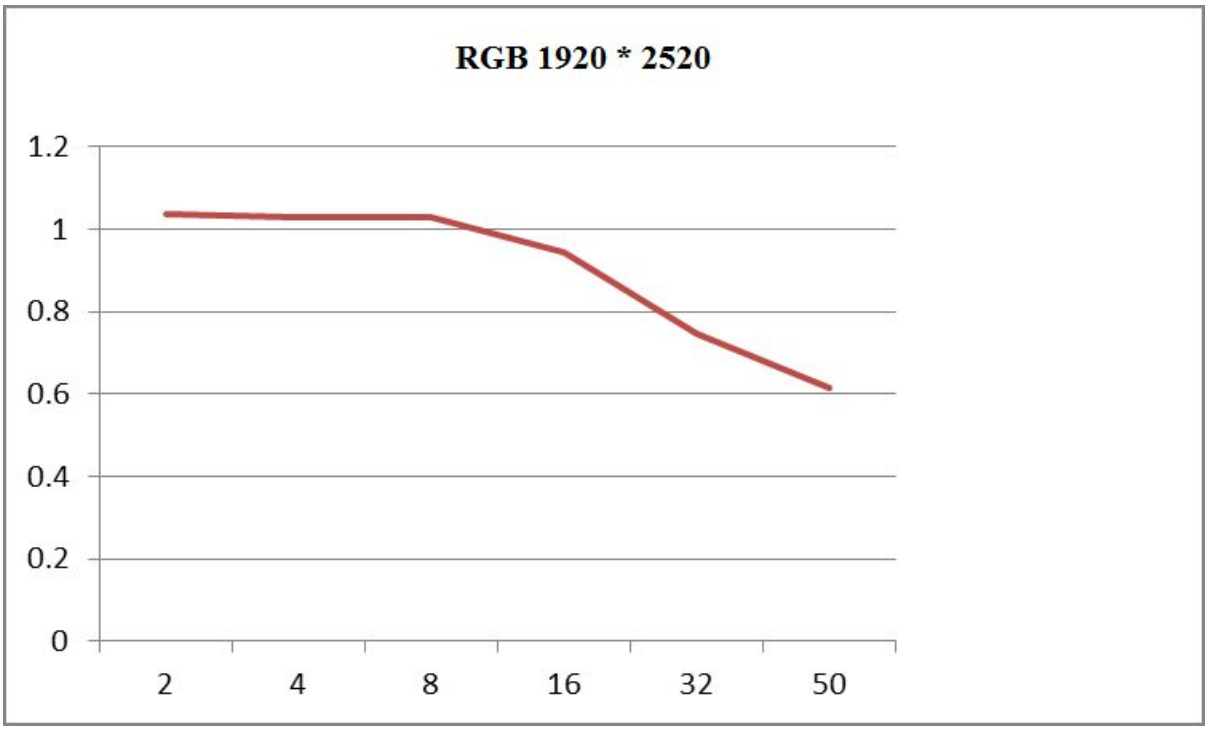

**Δ.2.3 Speed Up = Tserial / Tparallel**




**Δ.2.3 Efficiency = Speed Up / Proc**




**Δ.2.4 Speed Up = Tserial / Tparallel**

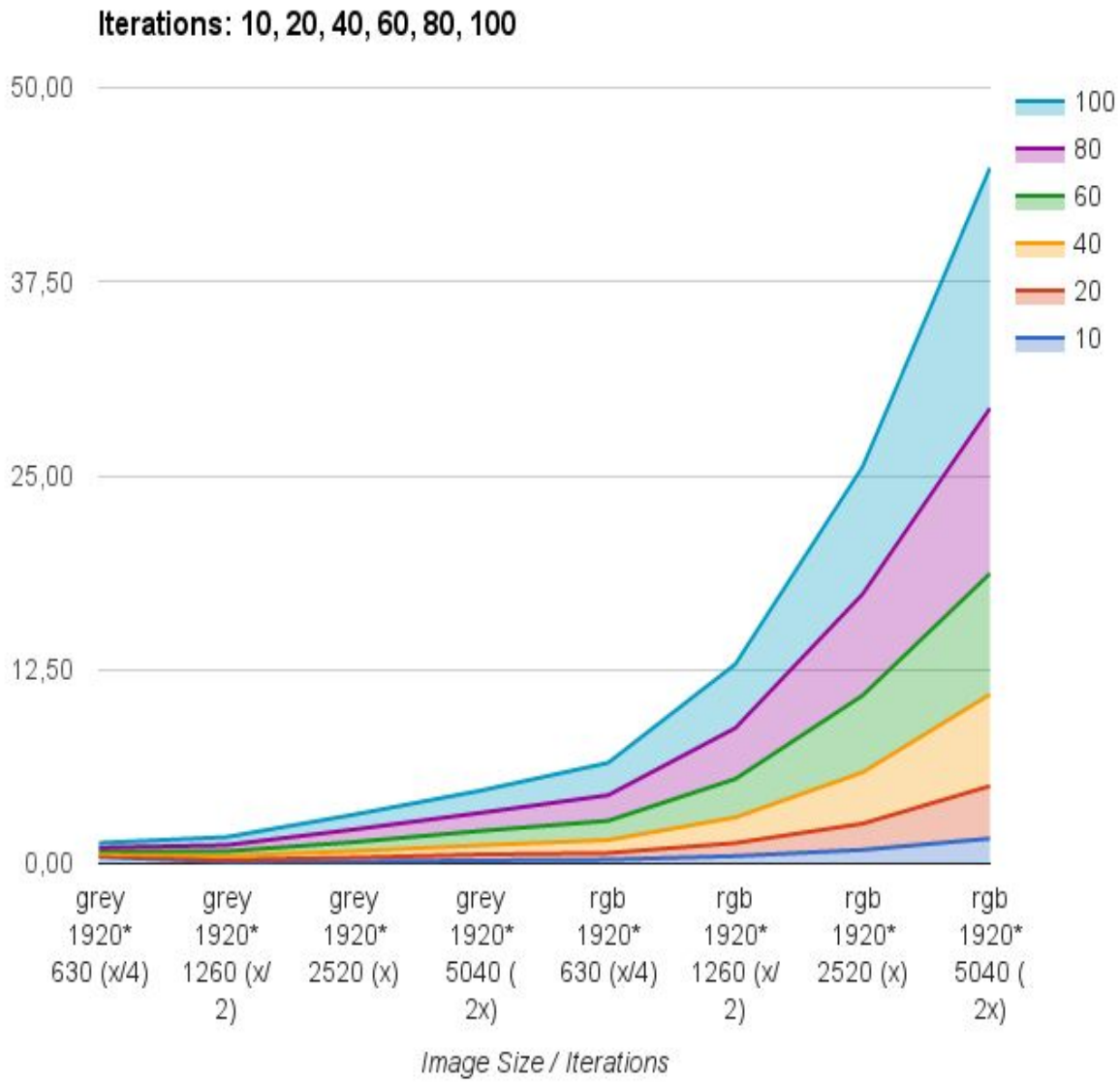



**Δ.2.5 Efficiency = Speed Up / Proc**

# 3. CUDA

| CUDA | GEFORCE | GTX | 760 | OC | 2GB | |
|---|---|---|---|---|---|---|
| Image Size / Iterations | 10 | 20 | 40 | 60 | 80 | 100 |
| grey 1920*630 (x/4) | 0,44 | 0,07 | 0,12 | 0,19 | 0,22 | 0,30 |
| grey 1920*1260 (x/2) | 0,10 | 0,15 | 0,24 | 0,30 | 0,40 | 0,52 |
| grey 1920*2520 (x) | 0,14 | 0,25 | 0,42 | 0,58 | 0,81 | 0,96 |
| grey 1920*5040 (2x) | 0,20 | 0,39 | 0,59 | 0,92 | 1,17 | 1,44 |
| rgb 1920*630 (x/4) | 0,25 | 0,43 | 0,83 | 1,24 | 1,65 | 2,09 |
| rgb 1920*1260 (x/2) | 0,49 | 0,84 | 1,65 | 2,48 | 3,28 | 4,12 |
| rgb 1920*2520 (x) | 0,89 | 1,69 | 3,31 | 4,95 | 6,54 | 8,21 |
| rgb 1920*5040 (2x) | 1,61 | 3,38 | 5,90 | 7,79 | 10,66 | 15,49 |

**Δ.3.1 Μετρήσεις χρόνου για CUDA**



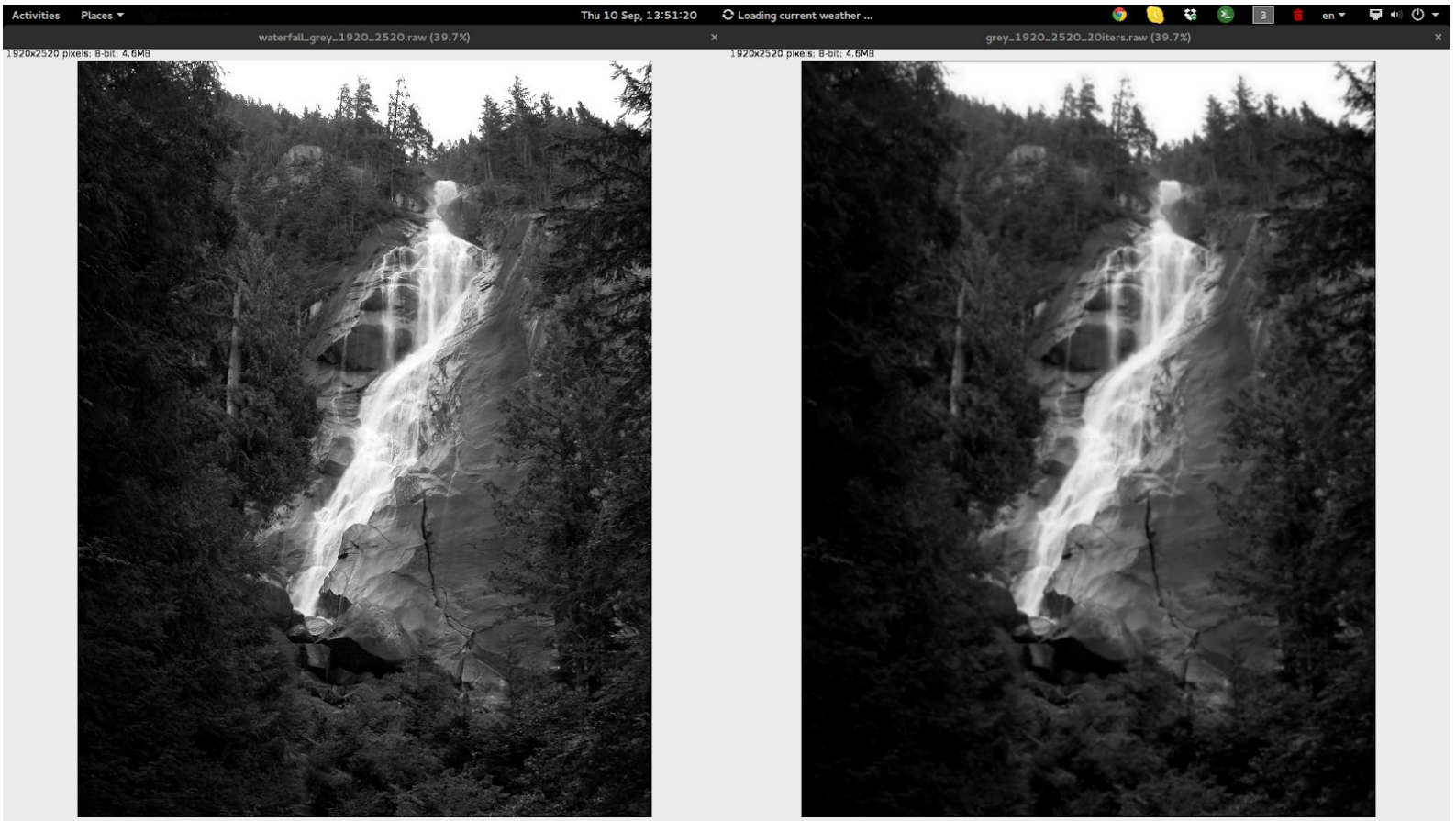

**Δ.2.2 Σχηματική σύγκριση μετρήσεων CUDA ως προς αριθμό επαναλήψεων**

Οι χρόνοι αυξάνονται για πιο μεγάλα μεγέθη, ενώ σε χαμηλά πληρώνουμε το κόστος δημιουργίας του thread μιας και το πρόγραμμα μπορεί να δουλέψει πιο βέλτιστα με λιγότερα.

# Ε) ΑΠΟΤΕΛΕΣΜΑΤΑ ΣΕ ΕΙΚΟΝΕΣ

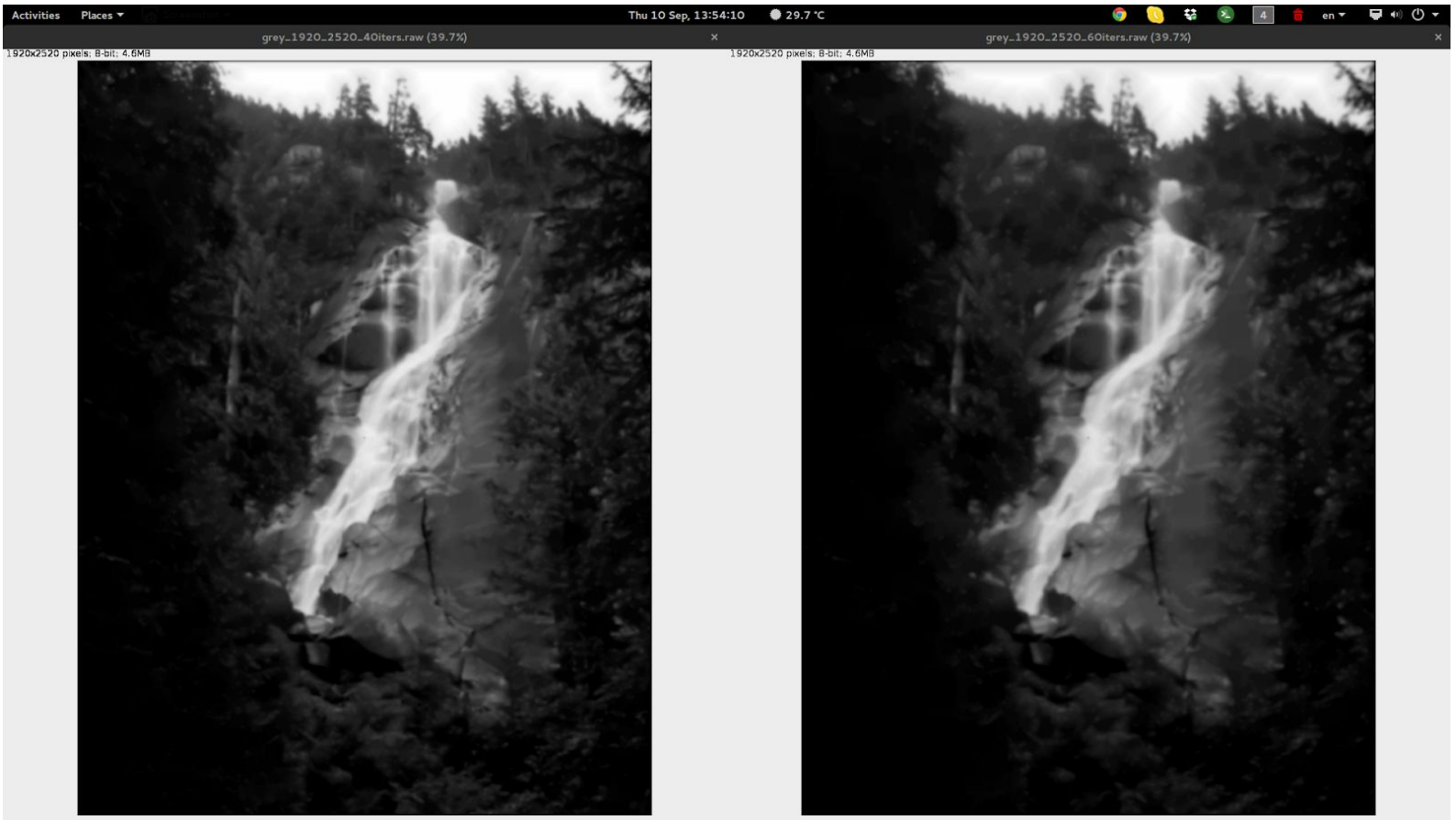
## 1. Grey

Αριστερά: 0 επαναλήψεις (αρχική εικόνα) Δεξιά: 20 επαναλήψεις



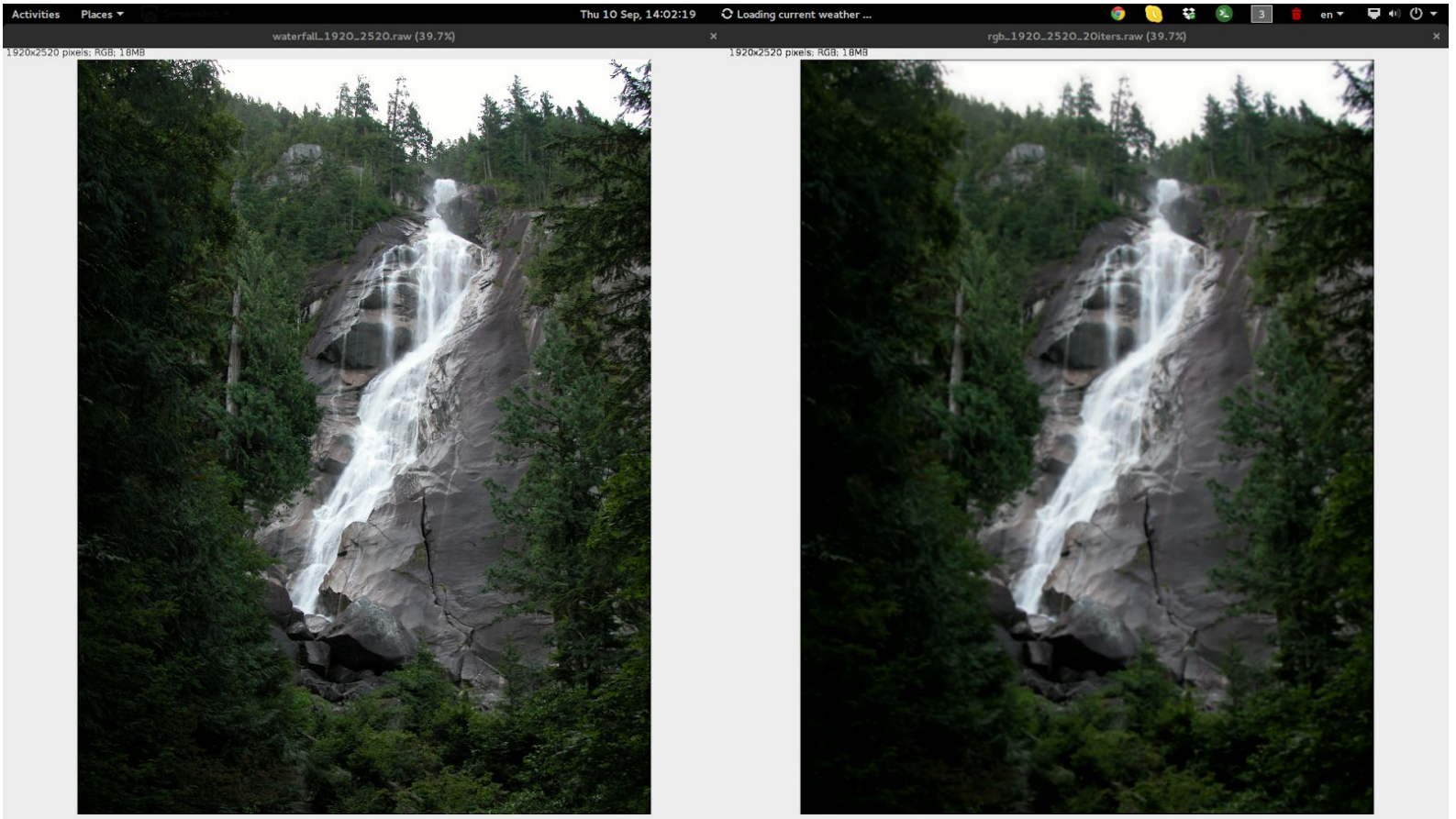
Αριστερά: 40 επαναλήψεις

Δεξιά: 60 επαναλήψεις



## 2. RGB

Αριστερά: 0 επαναλήψεις (αρχική εικόνα) Δεξιά: 20 επαναλήψεις



Αριστερά: 40 επαναλήψεις

Δεξιά: 60 επαναλήψεις