Παράλληλα και κατανεμημένα συστήματα
Εργασία 3

Μπεκιάρης Θεοφάνης ΑΕΜ:8200

Στόχος Εργασίας

Εκτέλεση του αλγόριθμου Non Local Mean με χρήση της Cuda για αποθορυβοποίηση εικόνων.

Παρατηρήσεις σχετικά με τον αλγόριθμο

-Ο αλγόριθμος απαιτεί την σωστή επιλογή μαθηματικής πράξης για το συμβολισμό ||f(i)-f(j)|| που περιγράφει το κατά πόσο είναι όμοιες οι γειτονιές 2 pixels.
-Ο αλγόριθμος για τον υπολογισμό της νέας τιμής ενός pixel απαιτεί πρώτα τον υπολογισμό της τιμής Z(i) που αντιστοιχεί σε ένα pixel και ύστερα το άθροισμα όλων των τιμών w(i,j)*f(j),κάτι το οποίο καθιστά στην παραλληλοποίηση του αλγόριθμου περιπλοκότητα και επιπλέον καθυστερήσεις.

Κώδικες που περιέχονται στης εργασία

Για την διεκπεραίωση της εργασίας έχουν κατασκευαστεί:
-Ένα σειριακό πρόγραμμα το οποίο έγραψα και εκτελεί την υλοποίηση του αλγόριθμου Non Local Mean.
-Ένα πρόγραμμα που περιέχει παραλληλοποιημένη έκδοση του

προηγούμενου προγράμματος με χρήση της Cuda και το οποίο διαχειρίζεται την εικόνα σαν να αποτελείται απο πολλές μικρές υποεικόνες -Ένα δεύτερο πρόγραμμα που περιέχει παραλληλοποιημένη έκδοση του προγράμματος με χρήση της Cuda το οποίο όμως λαμβάνει υπόψην του όλα τα pixel της εικόνας για την νέα τιμή των αποθυβοποιημένων pixel

Οι παραπάνω κώδικες που έχουν υλοποιηθεί σε γλώσσα C χρησιμοποιούνται σε συνδιασμό με το πρόγραμμα MATLAB.Εχώ χρησιμοποιήσει τα αρχικά Scripts MATLAB που δόθηκαν μαζί με την εκφώνηση της εργασίας,τα οποία με τις απαραίτητες αλλαγές καλούν τις υλοποιήσεις του αλγόριθμου Non Local Mean.

Σειριακό Πρόγραμμα

Συγκεκριμένες λεπτομέριες για τον τρόπο υλοποίησης του αλγορίθμου δίνονται με σχόλια μέσα στο πρόγραμμα.Ένα σημείο που θα επιλέξω να αναλύσω είναι το κομμάτι για τον υπολογισμό της διαφοράς ||f(i)-f(j)||.Για την συγκεκριμένη πράξη έχω επιλέξει να εκτελέσω την καρτεσιανή διαφορά των δυο pixel με βάρη.Συγκεκριμένα την ερμηνεύω ως το άθροισμα των τετράγωνων της διαφοράς (xi-xj) των τιμών των pixel γειτόνων πολλαπλασιαζόμενη με ένα βάρος wi,το βάρος που έχω επιλέξει είναι η απόσταση των pixel-γειτόνων απο το κεντρικό pixel το οποίο επεξεργαζόμαστε.

$$\left|\left| f(i) - f(j) \right|\right| = \sum \left\{ (xi - xj)^2 \cdot w_k \right\} / \Sigma(w_k)$$

Το πρόγραμμα υλοποιεί την πράξη αυτή μέσω της συνάρτησης gaussianDistance().

Η συνάρτηση normFactor() υπολογίζει τις τιμές Zi των pixel.

Η συνάρτηση weightingFunct() υπολογίζει τα βάρη w(i,j).

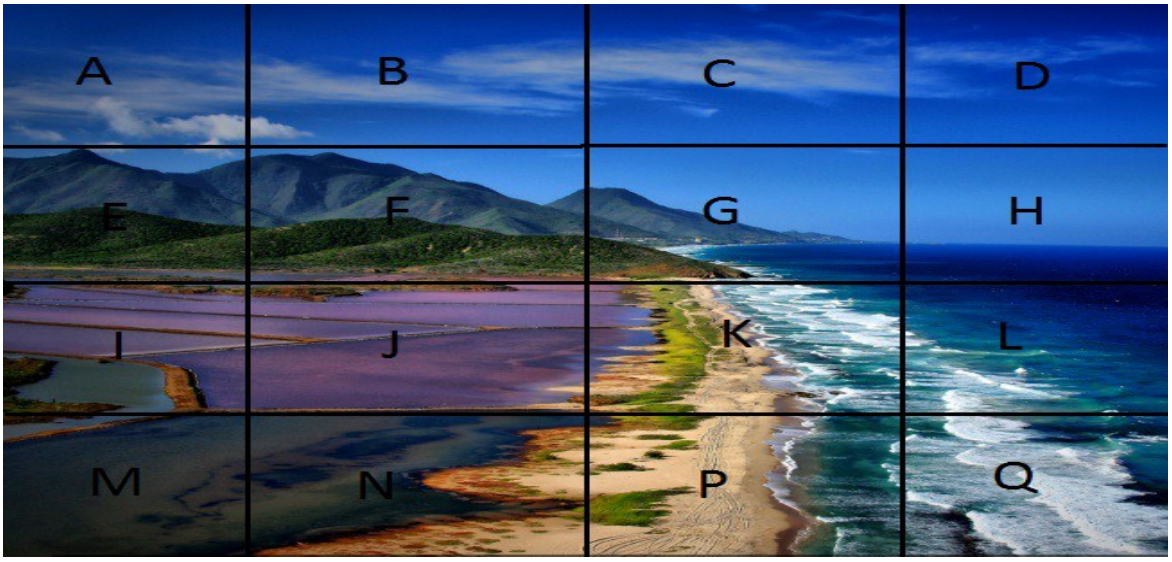
Τέλος η συνάρτηση nonLocalMeans() χρησιμοποιήει τα βάρη w(i,j) για να υπολογίσει το άθροισμα

$$\hat{f}(\mathbf{x}) = \sum_{\mathbf{y}\in\Omega} w(\mathbf{x}.\mathbf{y}) f(\mathbf{y}), \qquad \forall \mathbf{x} \in \Omega,$$

αρα και την νέα τιμή του pixel στο οποίο κάνουμε αποθορυβοποίηση.

Πρώτο παραλληλοποιημένο πρόγραμμα σε Cuda

Η λογική με την οποία παραλληλοποίησα το αρχικό σειριακό πρόγραμμα ήταν η εξής:
Θεωρούμε οτι η αρχική εικόνα χωρίζεται σε μικρότερα κομμάτια όπως φαίνεται για παράδειγμα στην ακόλουθη εικόνα.

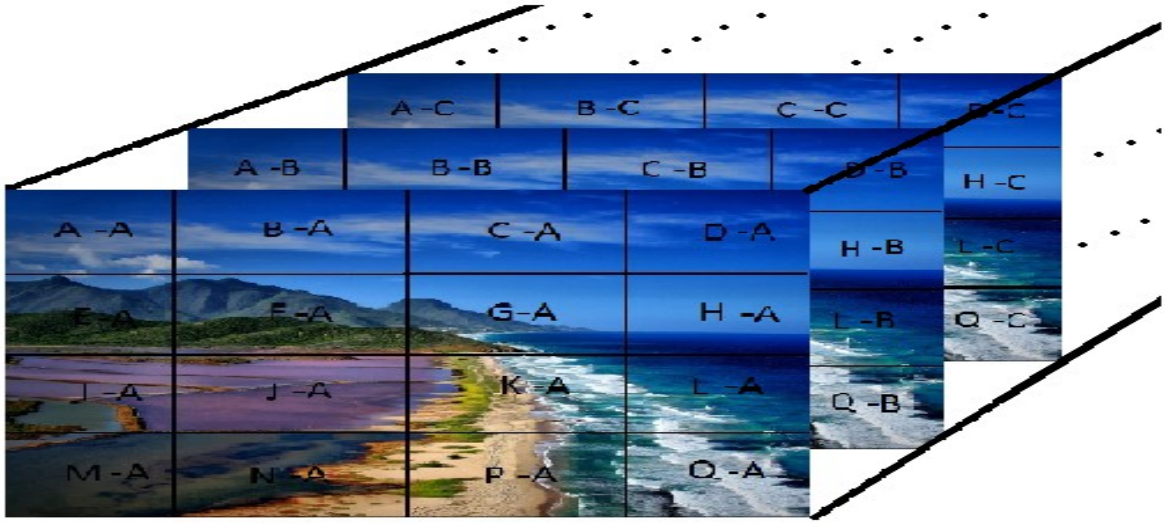


Μέσα απο το Sript του MATLAB οταν θα καλέσουμε το πρόγραμμα kernel το οποίο υλοποιεί τον NonLocalMeans αλγόριθμο,δημιουργούμε ενα grid το οποίο θα περιέχει έναν αριθμό απο blocks το καθένα απο τα οποία θα αναλάβει να διαχειριστεί και να εκτελέσει τον αλγόριθμο σε ένα απο τα κομμάτια στα οποία έχουμε χωρίσει την αρχική εικόνα.Τα blocks είναι δύο διαστάσεων και αποτελούνται απο NxN threads.Ο αριθμός των κομματιών στα οποία θα χωρίσουμε την εικόνα καθώς και των blocks εξαρτάται απο τον αριθμό των διαστάσεων που θα ορίσουμε να περιέχει το κάθε block.Ο αριθμός που έχω ορίσει ειναι το 16 (δηλαδή blocks 16x16 αρα 256 threads/block) καθώς διαιρεί ομοιόμορφα την εικόνα δεδομένου των διαστάσεων εικόνων που πρέπει να χρησιμοποιήσουμε και δέν χρειάζεται να εισάγουμε στον κώδικα επιπλέον συνθήκες ελέγχου για το αν έχουμε βγεί εκτός ορίων,είναι αρκετά μεγάλος για να προκύψουν καλά αποτελέσματα αποθορυβοποίησης και δεν ξεπερνάει τα όρια των threads που μπορούμε να χρησιμοποιήσουμε ανα block.Το κάθε thread αναλαμβάνει να υπολογίσει την νέα τιμή ενός απο τα pixel της εικόνας εκτελώντας τον αλγόριθμο μέσα στο block που ανήκει.Για παράδειγμα το block με συντεταγμένες (0,0) δηλαδή blockIdx.x=0 και blockIdx.y=0 φορτώνει απο την global στην shared μνήμη τις τιμές των pixel που αντιστοιχούν στο χώρο Α και έπειτα εκτελεί τον αλγόριθμο λαμάνοντας υπόψη μόνο τα pixel της περιοχής Α.Ομοίως η ίδια λογική ακολουθείται

και για όλες τις υπόλοιπες περιοχές B,C,D,..... κλπ.Με την παραπάνω υλοποίηση υπολογίζονται ταυτόχρονα όλες οι περιοχές τις εικόνας και η χρήση της shared memory βοηθάει ακόμα περαιτέρω τον χρόνο εκτέλεσης καθώς κάθε thread των block εκτελεί μόνο μία προσπέλαση στην global μνήμη μεταφέροντας μία τιμή ενός pixel στην shared μνήμη η οποία μπορεί να χρησιμοποιήθεί και απο τα υπόλοιπα thread του block στο οποίο ανήκει.

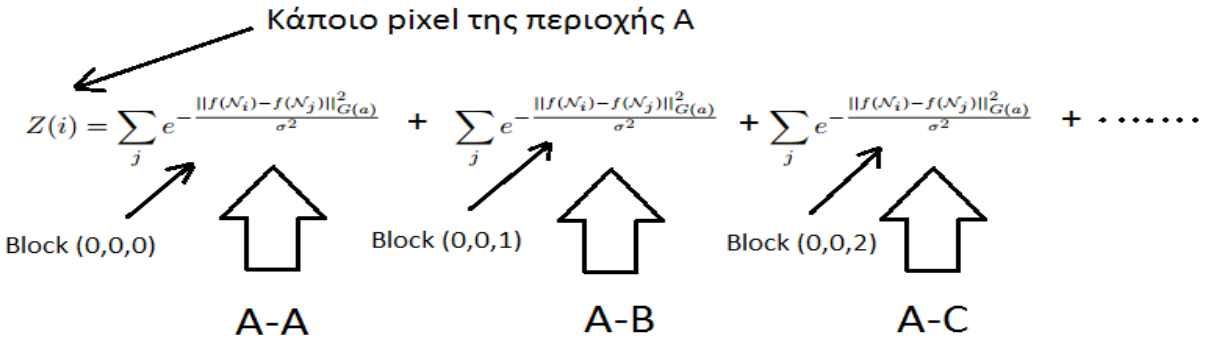
Δεύτερο παραλληλοποιημένο πρόγραμμα σε Cuda

Η προηγούνεμη υλοποίηση αν και είναι αρκετά αποδοτική και σε χρόνο και σε ποιότητα της αποθορυβοποίησης των εικόνων διαχειρίζεται την αρχική εικόνα σαν σύνολο υπο-εικόνων αλλοιώνονται τον αλγόριθμο NonLocalMeans ο οποίος υποθέτει οτι για την νέα τιμή ενος pixel πρέπει να λάβουμε υπόψην όλα τα pixel της εικόνας.Επιπλέον με την προηγούμενη υλοποίηση παρατηρούμε ότι για πολύ μεγάλες εικόνες εμφανίζονται κάποιες γραμμμές στην αποθορυβοποιημένη εικόνα.Για αυτόν τον λόγο και επειδή δεν ήξερα κατα πόσο σωστή είναι η παραπάνω λογική παραλληλοποίησης δημιούργησα και μία επιπλέον υλοποίηση η οποία συνυπολογίζει όλα τα pixel της εικόνας για να υπολογίσει την νέα τιμή ενός pixel.
Κατα την εκτέλεση του προγράμματος kernel το πρόγραμμα μας δημιουργεί ενα 3D Grid απο blocks.Οι 2 διαστάσεις του Grid διαμορφώνονται με τον ίδιο τρόπο όπως είδαμε και πρίν,δημαδή κάθε block θα υπολογίσει την νεά τιμή των pixel που του ανήκουν με την διαφορά όμως πως τα blocks πλέον_δεν θα λαμβάνουν υπόψην τους μόνο την περιοχή που τους ανήκει αλλά ανάλογα με το βάθος που βρίσκονται μέσα στην τρίτη διάσταση θα φορτώνουν μία άλλη περιοχή της εικόνας και θα την συγκρίνουν με αυτήν που τους ανήκει.Το μέγεθος της τρίτης διάστασης Z είναι ίσο με το πλήθος όλων των δυνατών περιπτώσεων σύγκρισης μεταξύ περιοχών και αυτός ο αριθμός είναι ίσος με το γινόμενο των άλλων δύο διαστάσεων X,Y.Για παράδειγμα για εικόνα 256x256 δημιουργείται Grid μεγέθους (16,16,256). Η παρακάτω εικόνα θα βοηθήσει περισσότερο στην κατανόηση της παραπάνω λογικής.



Ας πάρουμε για παράδειγμα τα blocks με συντεταγμένες (0,0),το κάθε ένα απο τα block φορτώνει στην shared μνήνη που του ανιστοιχεί τις τιμές των pixel της περιοχής Α,θα φορτώσει όμως επιπλέον και έναν άλλο πίνακα στην shared μνήμη με τιμές απο κάποια άλλη περιοχή της εικόνας ανάλογα το βάθος στο οποίο βρίσκεται στην τρίτη διάσταση Z (blockIdx.z) και θα συγκρίνει πλέον τις νέες τιμές της περιοχής Α με την αντίστοιχη(΄΄δεύτερη΄΄) περιοχή που φόρτωσε στην shared μνήμη.Δηλαδή στην ουσία πλεόν όλα τα block με συντεταγμένες blockIdx.x=0 και blockIdx.y=0 θα είναι υπεύθυνα για τον υπολογισμό των τιμών της περιοχής Α .Για παράδειγμα το block με συντεταγμένες (0,0,0) θα φορτώσει το τμήμα Α και θα το συγκρίνει με τον ίδιο το Α.Το block με συντεταγμένες (0,0,1) θα φορτώσει το τμήμα Α και Β και θα συγκρίνει το Α με το Β.Ομοίως το block με συντεταγμένες (0,0,2) θα φορτώσει το τμήμα Α και C και θα συγκρίνει το Α με το C και ούτο καθεξής.Οταν μιλάμε για σύκριση και υπολογισμούς του ενός σε σχέση με το άλλο στην ουσία αναφερόμαστε στον υπολογισμό των μερικών αθροισμάτων για τον υπολογισμό του Zi και για τα μερικά αθροίσματα w(i,j)*f(j).Δηλαδή ένα thread ενος block για ένα pixel i θα υπολογίσει την τιμή Zi υπολογίζοντας τα μερικά αθροίσματα που προκύπτουν απο την σύκριση των 2 περιοχών που έχουν φορτωθεί στη μνημη shared.Στο τέλος κάθε σύγκρισης,κάθε block,για παράδειγμα με συντεταγμένες (0,0) δηλαδή blockIdx.x=0 και blockIdx.y=0 της περιοχής Α θα έχει υπολογίσει ένα απο τα μερικά αθροίσματα που αντιστοιχούν στον τύπο για το υπολογίζμό του Zi και έχουν να κάνουν με τα pixel μεταξύ των δύο περιοχών,τα οποία θα πρέπει να προστεθούν όλα μάζι για να προκύψει η τελική τιμή του Zi για το pixel i κάτι το οποίο επιτυγχάνεται με την εντολή atomicAdd.Με την ίδια

λογίκη υπολογίζονται τα Zi των pixel και για τις υπόλοιπες περιοχές.Όταν τελικά υπολογιστούν οι τιμές Zi τότε τα block εκτελούν με την ίδια λογική και τους υπολογισμούς για τα μερικά αθροίσματα w(i,j)*f(j).Στην παρακάτω εικόνα γίνεται προσπάθεια για την απεικόνιση της παραπάνω λογικής για τα pixel της περιοχής A τα οποία τα διαχειρίζονται τα blocks με συντεταγμένες blockIdx.x=0 και blockIdx.y=0.



Παρατήρη για μια λεπτομέρια στην υλοποίηση του κώδικα

Ο αλγορίθμος απαιτεί πρώτα τον συνολικό υπολογισμό της τιμής Zi ενός pixel και αυτό δεδομένου της παραπάνω υλοποίησης δημιουργεί πρόβλημα που οφείλεται στην αδυναμία για συγχρονισμό μεταξύ των block.Οταν ένα block για παράδειγμα γράψει τα μερικά αθροίσματα της τιμής Zi ή των αθροισμάτων w(i,j)*f(j) θα πρέπει να περιμένει τα άλλα block να τελειώσουν τις εγραφές στην τιμή Zi ώστε να χρησιμοποιηθεί η συνολική τιμή της μεταβλητής Zi.Για να πετύχουμε λοιπόν τον παραπάνω συγχρονισμό μέσα απο το Sript Matlab υπολογίζουμε πρώτα όλες τις τιμές Zi,έπειτα μεταφέρουμε τον έλεγχο στην cpu (για τον συγχρονισμό των blocks) και στην συνέχεια ξανά καλούμε την gpu να χρησιμοποιήσει τις τιμές Zi ώστε να υπολογίσει εν τέλη τις νέες τιμές των pixel.Για να μην δημιουργήσουμε δύο συναρτήσεις kernel το παραπάνω επιτυγχάνεται με την κλίση του ίδιου προγράμματος kernel μόνο που ανάλογα με το τι απο τα δύο θέλουμε,δηλαδή υπολογισμό του Zi ή των w(i,j)*f(j),εισάγουμε μια επιπλέον παράμετρο flag που θα καθορίσει την επιλογή μας.Έτσι για flag=0 το πρόγραμμα kernel υπολογίζει τα Zi ενώ για flag=1 υπολογίζει τα w(i,j)*f(j).

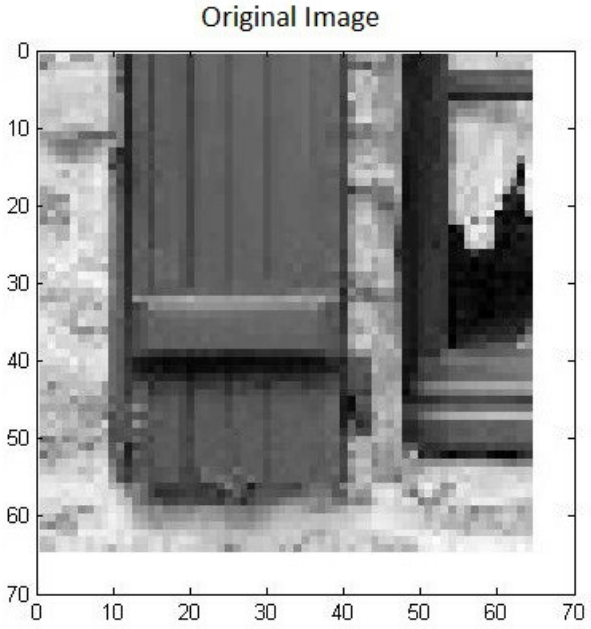
Σχεδίαση και περιγραφή τεχνητής εισόδου, για έλεγχο ορθότητας.

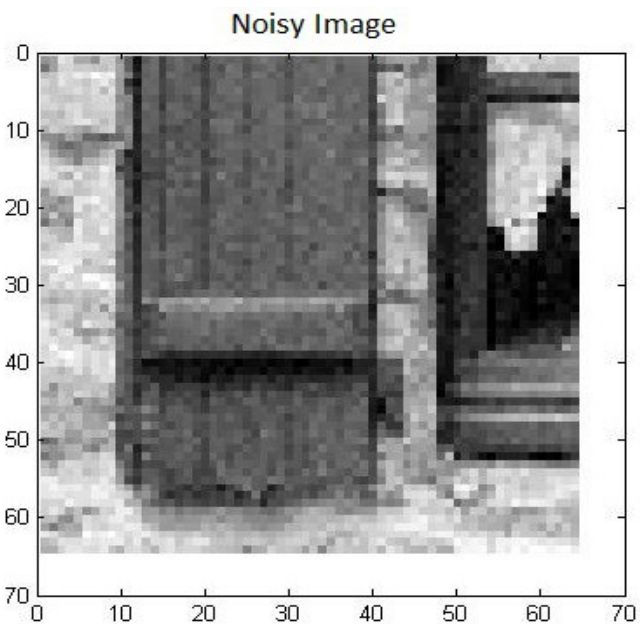
---Το σειριακό πρόγραμμα καλείται απο το script του matlab ως MEX file,το script εκτελεί αυτόματα την εντολή mex και επομένως δεν χρειάζεται να εκτελέσουμε κάποια λειτουργία περαιτέρω.
---Τα παραλληλοποίημένα προγράμματα σε cuda χρειάζονται πρίν την εκτέλεση του script να μεταγλωττιστούν με τον compiler nvcc της nvidia και γιαυτό τον σκοπό πρέπει πρώτα να εκτελεστεί το Makefile που βρίσκεται στο φάκελο cuda.
Το script ειναι μία συνάρτηση που παίρνει ως ορίσματα :
-Το μέγεθος των patch με τιμές 3,5,7,9,... κλπ
-Την επιθυμητή τιμή σ
-Το όνομα της εικόνας που θέλουμε να αποθορυβοποίησουμε και βρίσκεται στον φάκελο data.Το όνομα το εισάγουμε ως string δηλαδή ειναι της μορφής 'ονομαεικονας'.
Επομένως για να μπορέσουμε να τρέξουμε τα πρόγραμματα και να ελέγξουμε την ορθότητα τους καλούμε το script-συνάρτηση με όνομα pipeline_nonLocalMeans.m με τα αντίστοιχα ορίσματα.Ένα παράδειγμα ειναι της μορφής : pipeline_nonLocalMeans(3,0.04,'window64')
Τα προγράμματα κατά την εκτέλεση του αποθηκεύει τις εικόνες που προκύπτουν στον φάκελο images,και επομένως σε περίπτωση που η εκτέλεση του κώδικα γίνεται στο Διάδη μπορούν να κατέβουν και να απεικονιστούν στο Matlab σε κάποιο προσωπικό υπολογιστή.

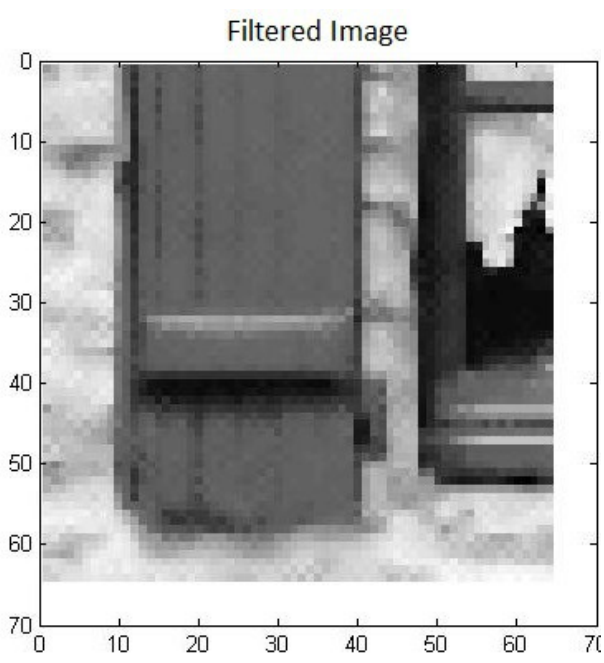
Αποτελέσματα εικόνων απο το σειριακό πρόγραμμα

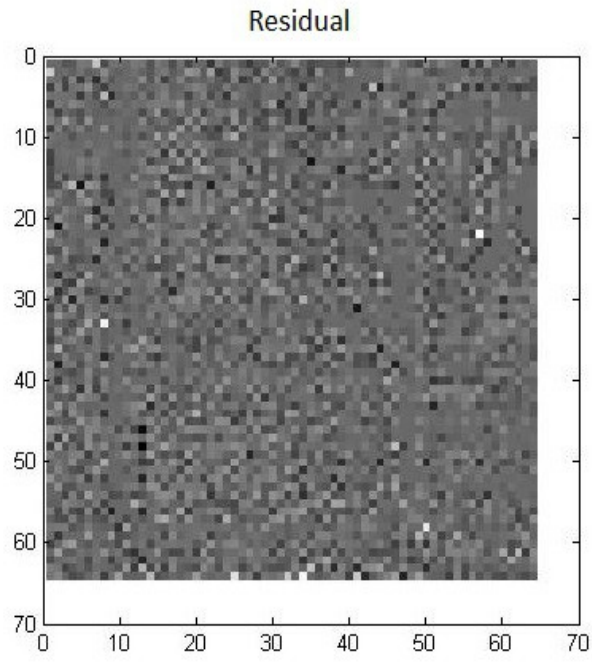
Τα παρακάτω αποτελέσματα προκύπτουν απο το σειριακό πρόγραμμα που έγραψα και όχι απο το προτεινόμενο.

Για εικόνα 64X64 pixel,patchSize=5 και sigma=0.04

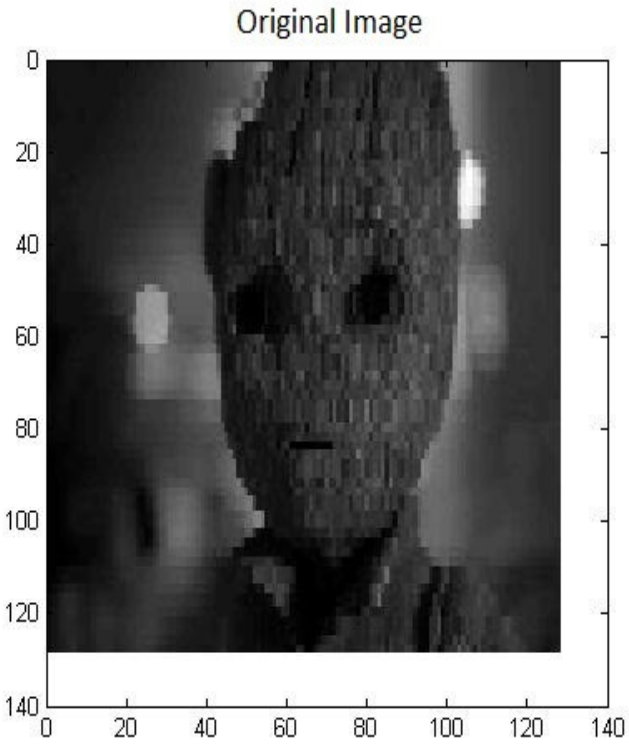


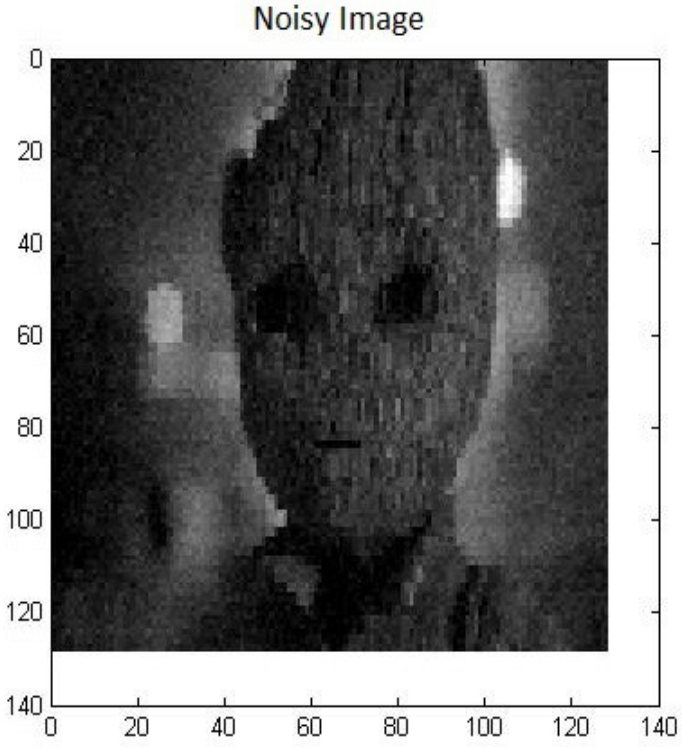


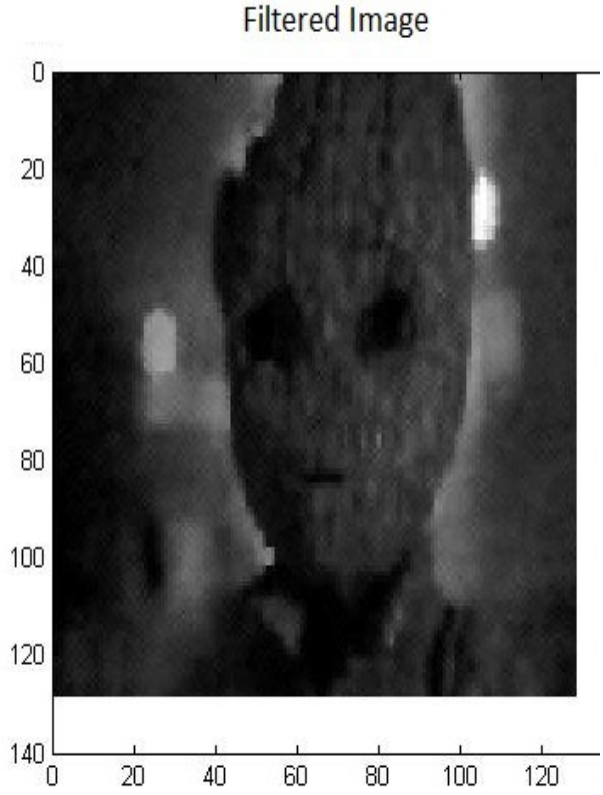


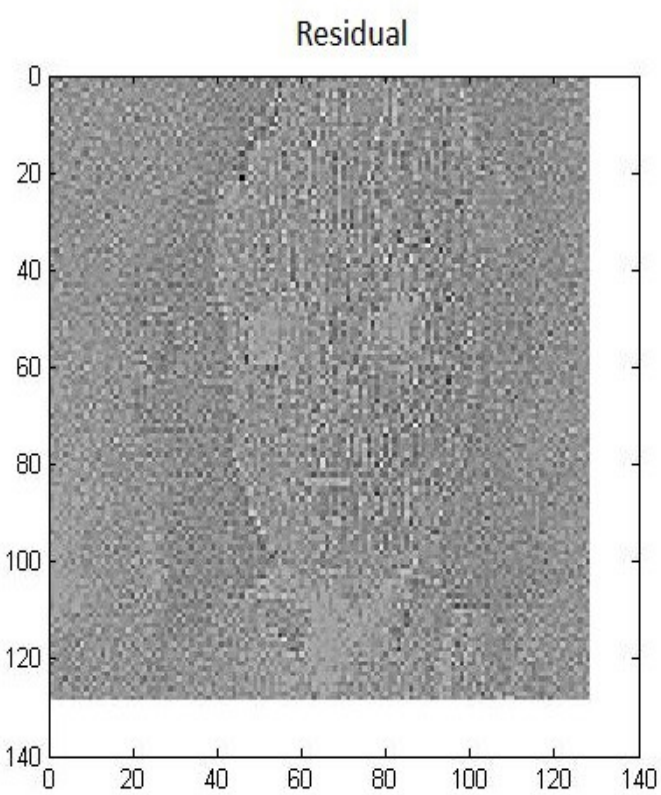


Για εικόνα 128X128 pixel,patchSize=5 και sigma=0.04



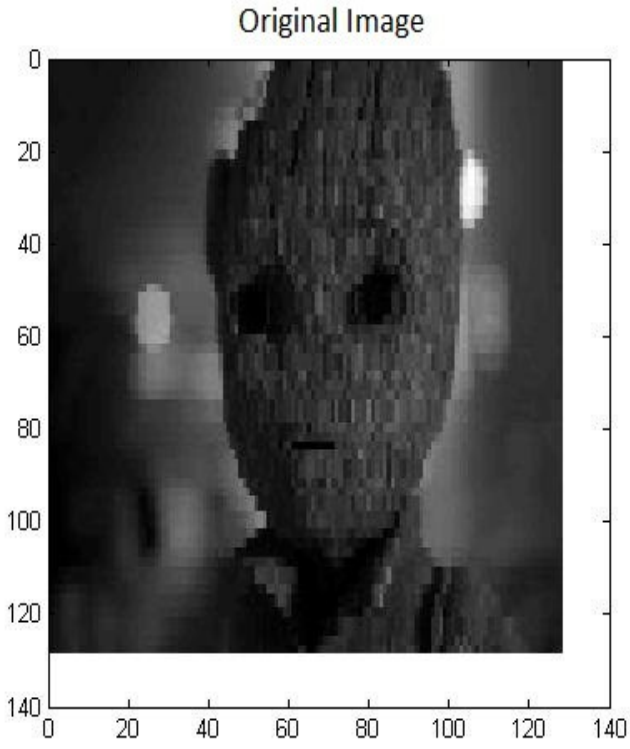


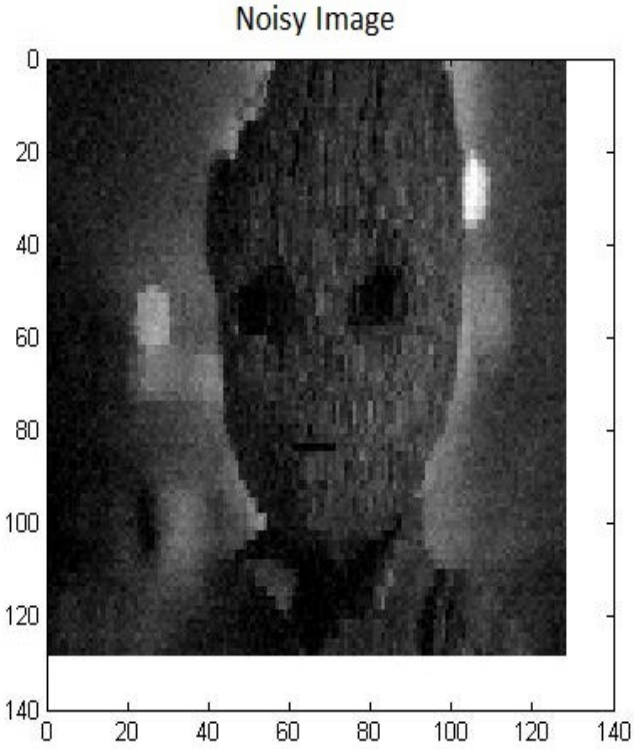


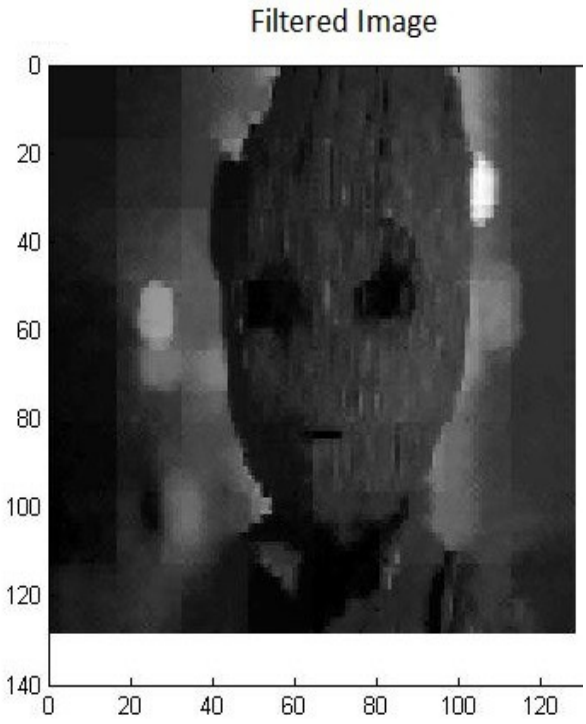


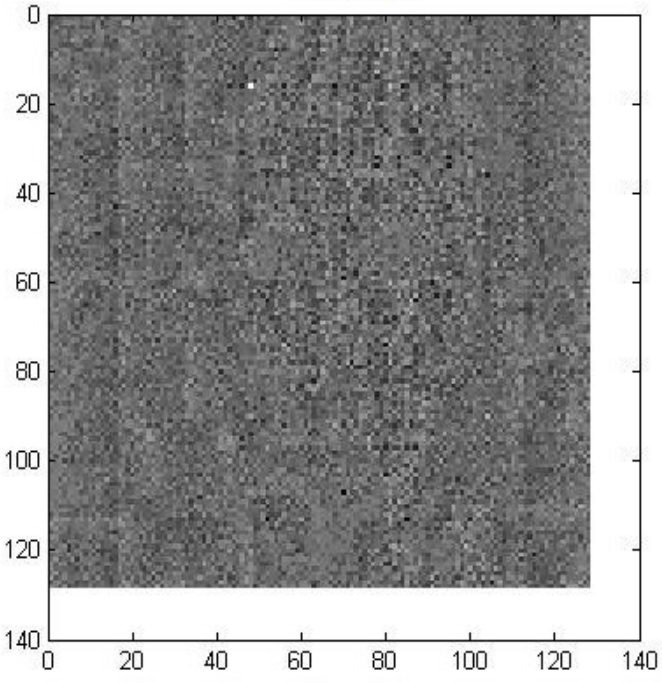
<u>Πρώτο Παραλληλοποιημένο Πρόγραμμα</u>

Για εικόνα 128X128 pixel,patchSize=5 και sigma=0.04

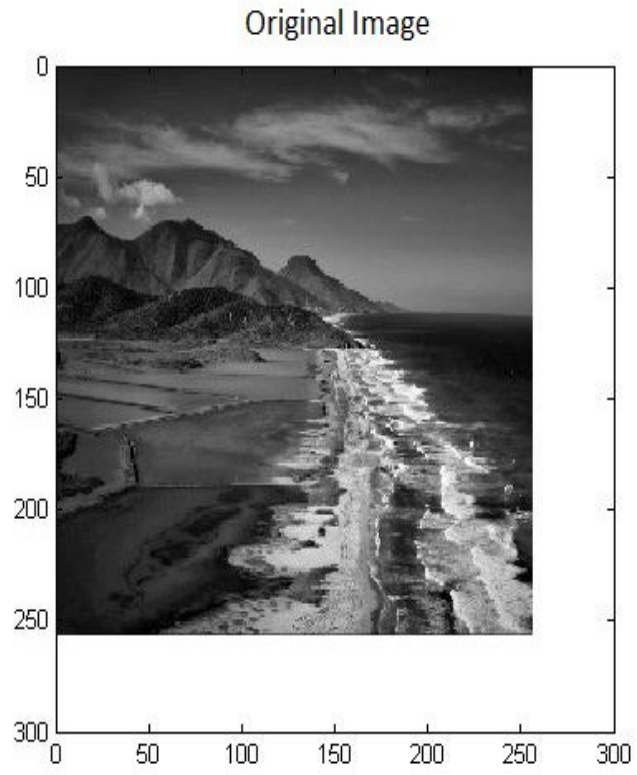


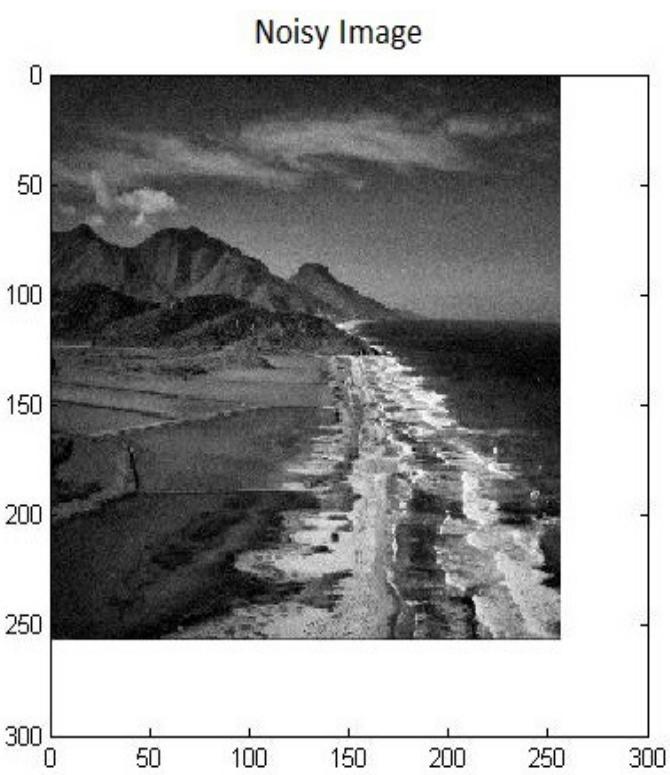


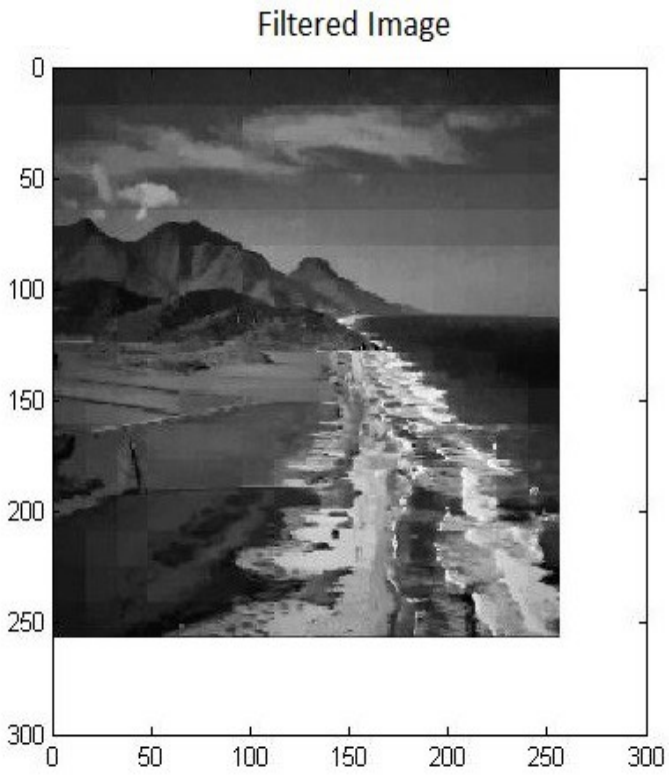


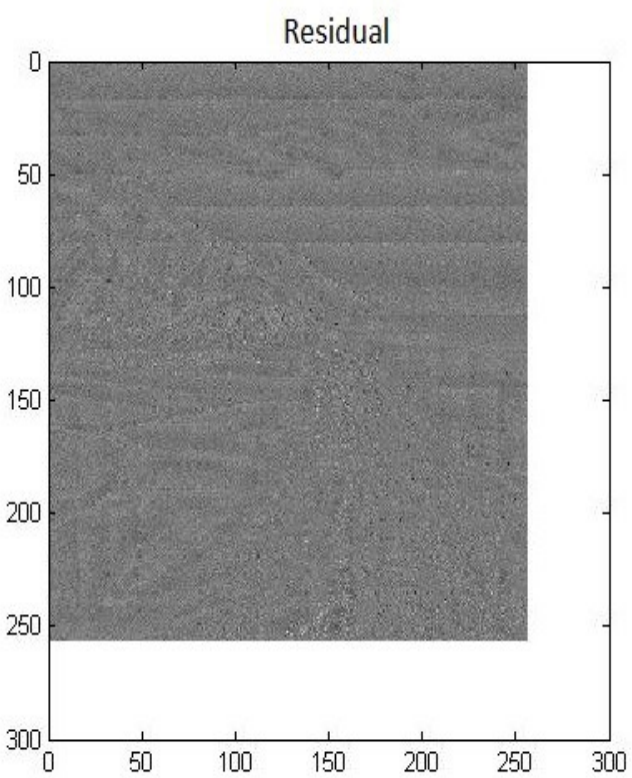


Για εικόνα 256X256 pixel,patchSize=5 και sigma=0.04



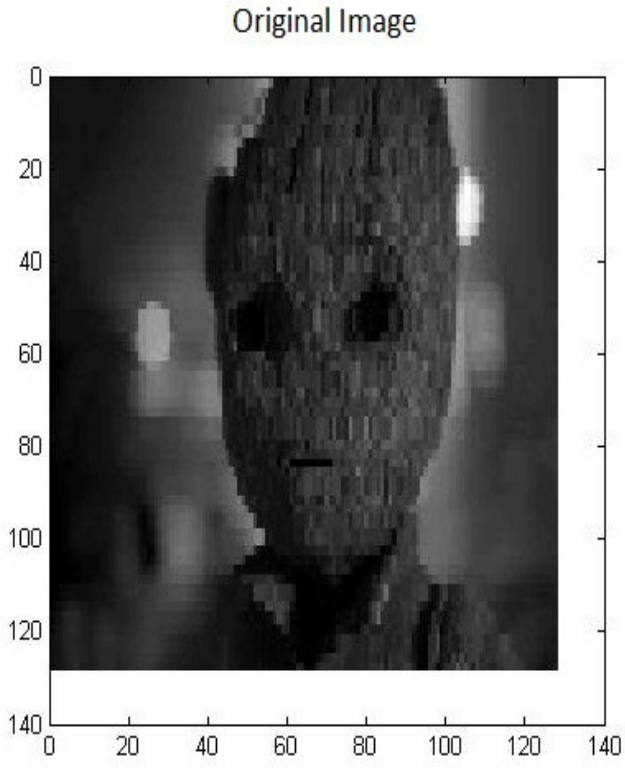


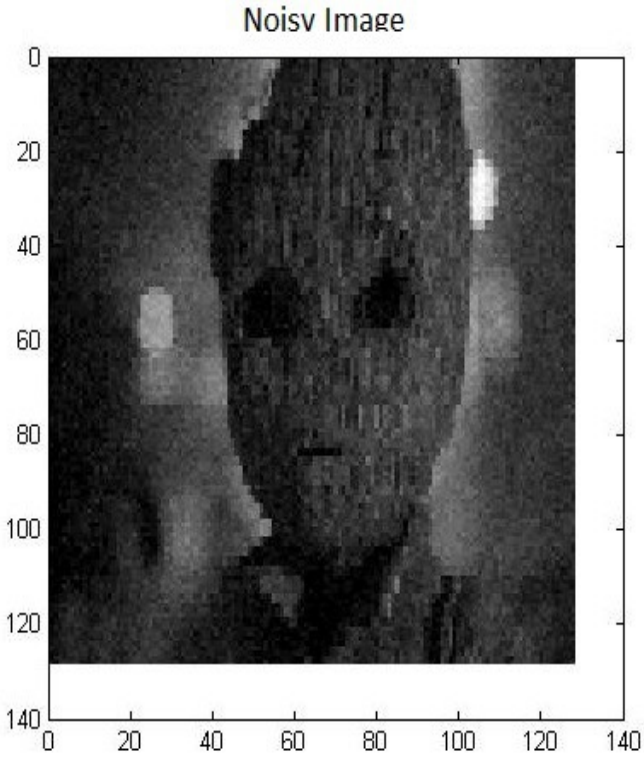


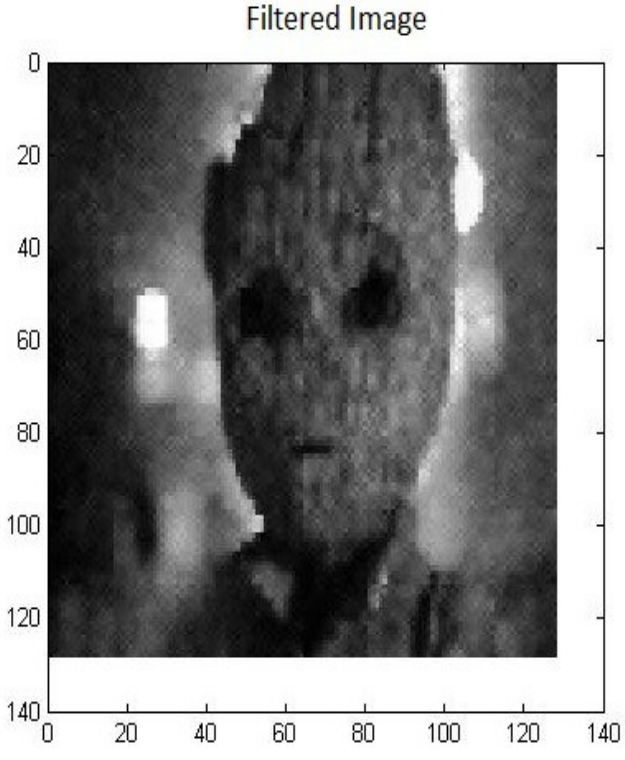


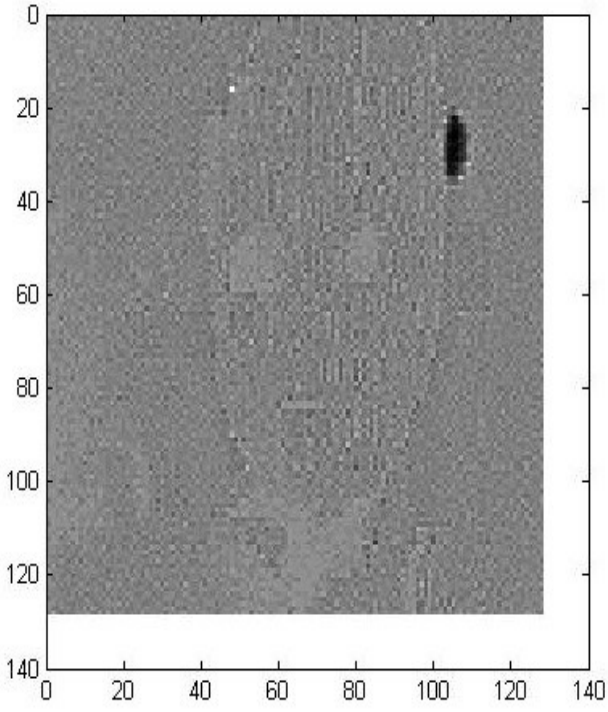
<u>Δεύτερο Παραλληλοποιημένο Πρόγραμμα</u>

Για εικόνα 128X128 pixel,patchSize=5 και sigma=0.04

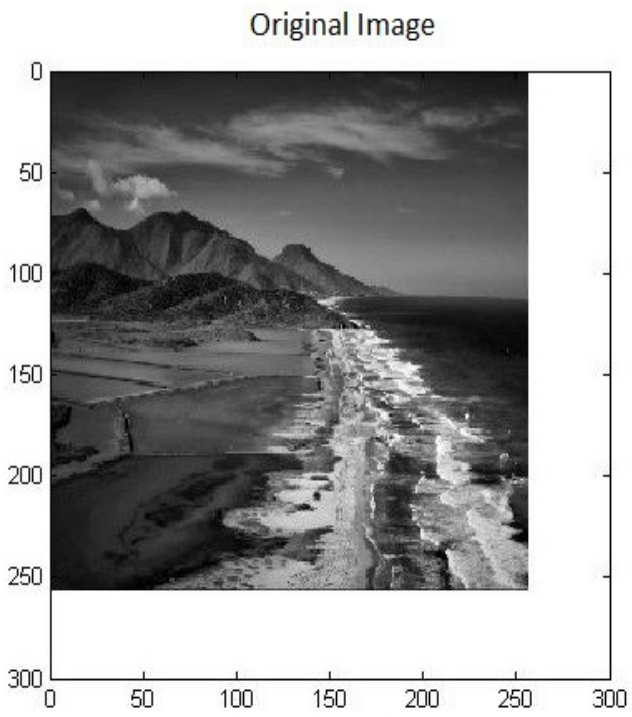


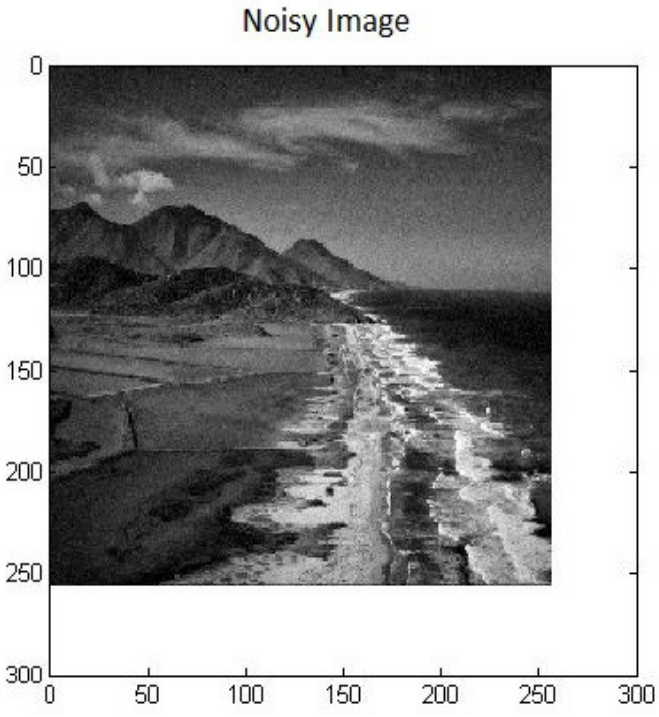


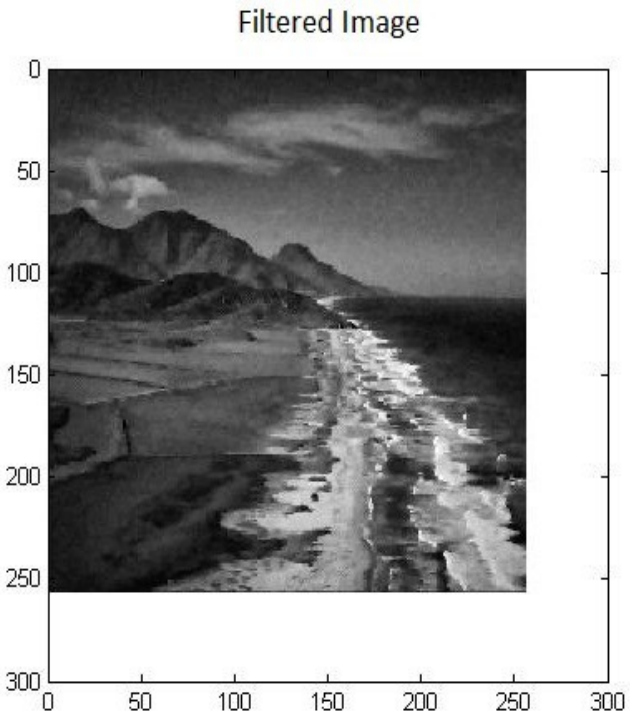


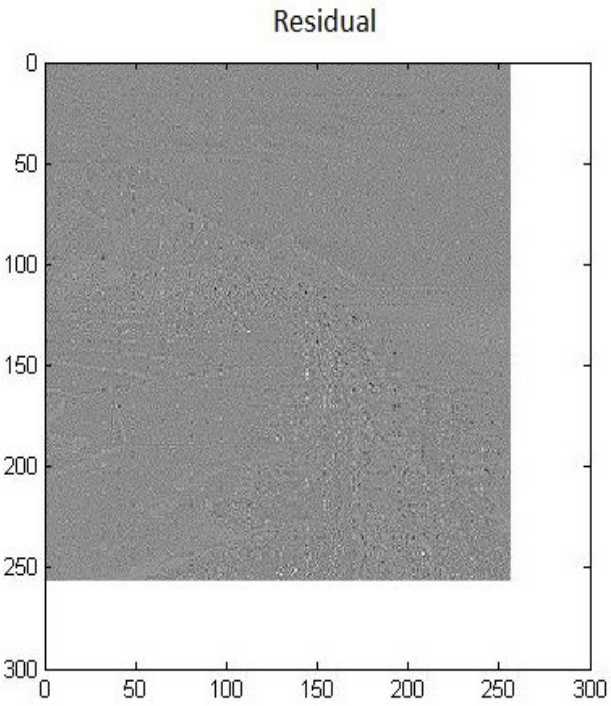


Για εικόνα 256X256 pixel,patchSize=5 και sigma=0.02









Οι χρόνοι εκτέλεσης των προγραμμάτων

Εικόνα 64x64	Σειριακό	Πρώτο Παραλληλοποιημενο	Δεύτερο Παραλληλοποιημένο
3x3 patch	5.97 sec	0.0044 sec	0.031 sec
5x5 patch	11.53 sec	0.0098 sec	0.074 sec
7x7 patch	20.17 sec	0.017sec	0.137sec

Εικόνα 128x128	Σειριακό	Πρώτο Παραλληλοποιημενο	Δεύτερο Παραλληλοποιημένο
3x3 patch	95 sec	0.011 sec	0.45 sec
5x5 patch	183 sec	0.024 sec	1.12 sec
7x7 patch	311 sec	0.044 sec	2.08 sec

Εικόνα 256x256	Σειριακό	Πρώτο Παραλληλοποιημενο	Δεύτερο Παραλληλοποιημένο
3x3 patch	– sec	0.033 sec	7.67 sec
5x5 patch	– sec	0.079 sec	18.21 sec
7x7 patch	– sec	0.14 sec	33.52 sec

Σχόλια και παρατηρήσεις

Στους παραπάνω πίνακες φαίνονται τα αποτελέσματα απο την εκτέλεση των προγραμμάτων για μεγέθοι εικόνων 64x64,128x128 και 256x256 και μεγέθοι των patches 3x3 5x5 και 7x7.Αρχικά αυτό που παρατηρούμε είναι η πολύ μεγάλη άυξηση στον χρόνο εκτέλεσης του σειριακού προγράμματος καθώς το μεγέθος εικόνας και patch αυξάνονται.Για μέγεθος εικόνων 256x256 οι χρόνοι ήταν αρκετά μεγάλη για να μετρηθούν,παρόλα αυτά η σύγκριση στις αποδώσεις των προγραμμάτων είναι εμφανής.Για την αποφυγή κάποιου μπερδέματος θα υπενθυμίσω ότι οι χρόνοι προκύπτουν απο το σειριακό πρόγραμμα που δημιούργησα εγώ και όχι απο το προτεινόμενο που δόθηκε μαζί με την εκφώνηση της εργασία.Οι χρόνοι και απο τα 2 παραλληλοποιημένα προγράμματα είναι με πολύ μεγάλη διαφορά καλύτεροι απο το σειριακό πρόγραμμα.Το πρώτο πρόγραμμα το οποίο διαχειρίζεται την εικόνα σαν σύνολο υπο-εικόνων

βλέπουμε ότι έχει πολύ μικρούς χρόνους εκτέλεσης αφού στην ουσία όσο και να αυξάνουμε τον αριθμό των pixel δημιουργούνται επιπλέον block τα οποία καλύπτουν τους υπολογιμούς χωρίς να επηρεάζονται απο το μέγεθος της εικόνας εφόσον κάθε block θα εκτελέσει τον αλγόριθμο σε μία περιοχή 16x16=256 pixels και κάθε thread του μπλόκ θα υπολογίσει την τιμή ενός απο τα pixel του block.Στην ουσία αυτο που γίνεται είναι ότι για κάθε επιπλέον pixel που προκύπτει απο μία μεγαλύτερη εικόνα παρέχουμε επιπλέον και περισσότερους υλικούς πόρους (επεξεργαστές) που αναλαμβάνουν να καλύψουν τους υπολόγισμούς,φυσικά υπάρχει ενα όριο για τους πόρους που μπορούν χρησιμοποιηθούν ατομικά απο ένα νήμα.Οι χρόνοι παρόλα αυτα έχουν μία μικρή (σχετικά) αύξηση επειδή όσο περισσότερα νήματα-block δημιουργούμε γίνεται πιο δύσκολος ο συχγρονισμός τους και επιπλέον υπάρχουν καθυστερήσεις απο την προσπάθεια για ταυτόχρονη προσπέλαση της global μνήμης απο πολλά νήματα. Το δέυτερο παραλληλοποιημένο πρόγραμμα έχει πολύ καλύτερους χρόνους απο το σειριακό πρόγραμμα,όπως βλέπουμε όμως είναι αρκετά μεγαλύτεροι σε σχέση με το προηγούμενο παραλληλοποιημένο πρόγραμμα.Η διαφορά αυτή οφείλεται στο ότι το δεύτερο πρόγραμμα αφού λαμβάνει υποψη όλες τις τιμές των pixel δημιουργεί πολύ περισσότερα thread-block τα οποία δεν είναι δυνατόν να είναι όλα ενεργά ταυτόχρονα,δημιουργούντε επιπλέον καθυστερήσεις απο τον τεράστιο αριθμό νημάτων που εκτελούν ταυτόχρονες προσπελάσεις στην μνήμη και τέλος έχουμε καθυστερήσεις και απο τον συχγρονισμό μεταξύ των block με την εντολή atomicadd() για την ατομική ανανέωση κοινών θέσεων μνήμης.Τα αποτελέσματα της αποθορυβοποίησης των εικόνων απο τα προγράμματα και στις δύο πριπτώσεις είναι πολύ καλά,με την παρατήρηση ότι στην περίπτωση του πρώτου παραλληλοποιημένου λόγο του μικρού δείγματος pixel για κάθε block βλέπουμε ότι στα όρια των block δημιουργούνται στην εικόνα κάποιες γραμμές.Αυτό αποφεύγεται στην δέυτερη υλοποίηση αφού πλέον λαμβάνουμε υπόψη όλα τα pixel της εικόνας.Τέλος βλέπουμε απο τις εικόνες Residual ότι η αποθορυβοποίηση δεν καταστρέφει την πληροφορία της αρχικής εικόνας αφού στις εικόνες Residual φαίνεται ο θόρυβος που έχει αφαιρεθεί απο την αρχική εικόνα ενώ η πληροφορία της αρχικής εικόνας φαίνεται αμυδρά εώς καθόλου.

Τελος