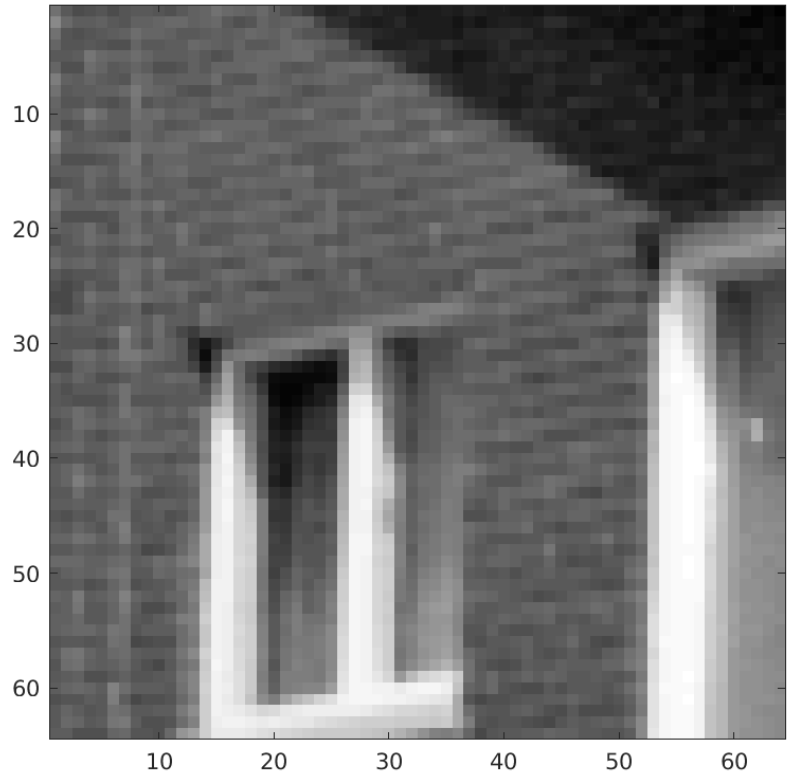
Παράλληλα και Διανεμημένα Συστήματα

Εργασία 3

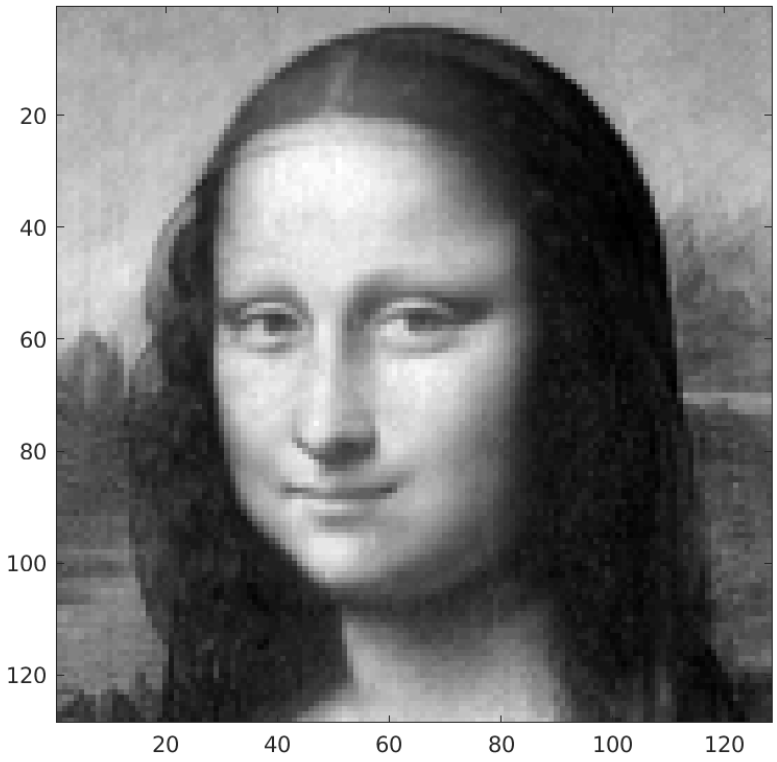
1) Εισαγωγικά

Σε αυτήν την εργασία του μαθήματος, κληθήκαμε να εκμεταλλευτούμε τις μεγάλες δυνατότητες των GPU που αφορούν την παράλληλη επεξεργασία δεδομένων. Πιο συγκεκριμένα, μας δόθηκε κάποιος κώδικας σε MATLAB και σε CUDA, πάνω στον οποίο έπρεπε να κάνουμε κάποιες αλλαγές για να μπορέσουμε να υλοποιήσουμε τον αλγόριθμο που μας ζητήθηκε. Ο αλγόριθμος που έπρεπε να υλοποιήσουμε ήταν ο non-local-means. Σκοπός αυτού του αλγορίθμου ήταν να έχουμε στο τέλος της επεξεργασίας κάποιας εικόνας, να προκύψει η αρχική πάνω στην οποία έγινε η επεξεργασία. «Διαβάζοντας» λίγο καλύτερα τα μαθηματικά που μας δόθηκαν, μπορούμε να βγάλουμε κάποια χρήσιμα συμπεράσματα για να γίνει η υλοποίηση. Πιο συγκεκριμένα, στον πρώτο τύπο γίνεται σαφές ότι **f**(**y**) θα είναι η εικόνα το **xεΩ** θα μου δείχνει το pixel. Όσον αφορά τα βάρη w(i, j), το εκθετικό θα μου δείχνει το πόσο θα μοιάζουν τα patches που γειτονεύουν και το σ^2 θα είναι το denoising strength. Όλα αυτά δείχνουν ότι ανάλογα με το μέγεθος της εικόνας θα παίρνω πιο μεγάλους ή πιο μικρούς χρόνους εκτέλεσης, αλλά και ανάλογα με την επιλογή του denoising strength θα βελτιώνεται ή θα χειροτερεύει η απόδοση και η ποιότητα του αλγορίθμου. Σε γενικές γραμμές, θα μπορούσαμε να πούμε ότι ο Non-Local-Means είναι σχετικά «χαζός» αλγόριθμος καθώς γίνεται προσπέλαση ολόκληρης της εικόνας χωρίς κάποια στόχευση σε συγκεκριμένα pixel. Η επεξεργασία της αφορά έναν Gaussian θόρυβο και οι εικόνες που χρησιμοποιήθηκαν ήταν 3 διαφορετικές ανάλυσης 64x64, 128x128 και 256x256. Σε κάθε εικόνα, έπρεπε να πάρουμε και περιοχές ανάλυσης 3x3, 5x5, 7x7 αντίστοιχα. Επιπλέον αναφέρω ότι χρειάστηκε να κάνω μερικές αλλαγές και στον κώδικα του MATLAB pipeline_non_local_means.m για να μπορέσω να περάσω κάποια ορίσματα στην GPU και να εκτελεστεί ο kernel. Όλες οι δοκιμές που έγιναν υπάρχουν στο GitHub μέσα στο φάκελο matlab ως αποτελέσματα της «θορυβοποίησης»-«αποθορυβοποίησης». Οι εικόνες που χρησιμοποιήθηκαν είναι οι εξής :

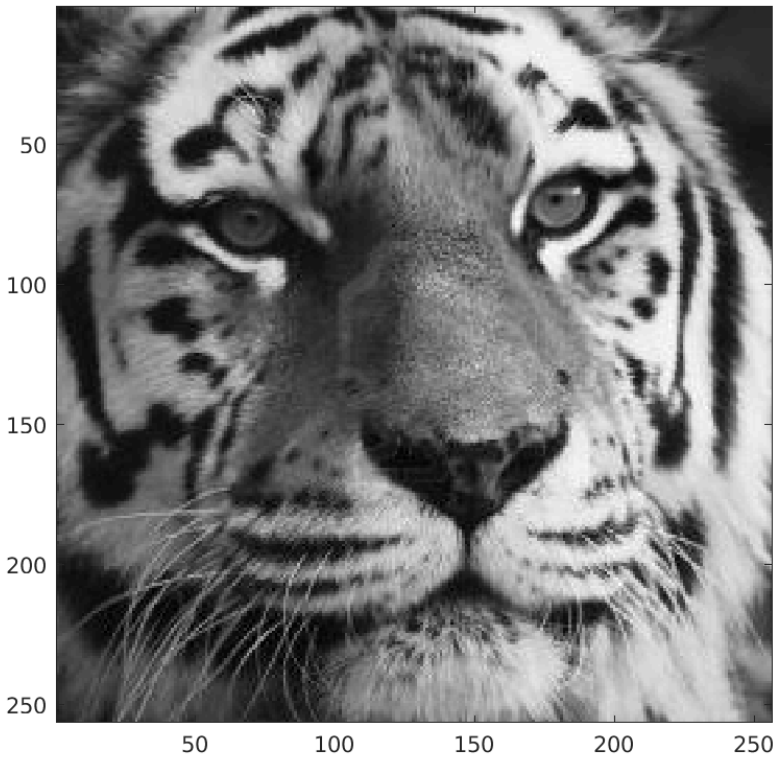
i) House, 64x64



ii) Monalisa, 128x128



iii) Tiger, 256x256



2) Global Memory

Σε αυτό το κομμάτι της εργασίας, χρησιμοποίησα μόνο την global memory της GPU παίρνοντας ολόκληρη την εικόνα για να την επεξεργαστώ. Εδώ πέρα θα μπορούσαμε να πούμε ότι χάνω πολύ χρόνο λόγο της προσπέλασης ολόκληρης της εικόνας, ενώ θα μπορούσα να γλυτώσω κάποιες προσπελάσεις στην εικόνα θα πρέπει να εκμεταλλευτώ τη shared memory, ανάλογα φυσικά με το μέγεθος των πλαισίων στα οποία χωρίζω την κάθε εικόνα (επόμενη υλοποίηση). Ο σκοπός αυτής της υλοποίησης ήταν να υπάρχει ως μέτρο σύγκρισης με την επόμενη υλοποίηση, η οποία όπως θα δούμε είχε μεγάλη βελτίωση από άποψη ταχύτητας. Η λογική του αλγορίθμου είναι να γίνεται χρήση ενός block στην GPU και 16x16 νημάτων στα οποία γίνεται επεξεργασία όλων των pixels. Με άλλο αριθμό threads δεν πήρα καλύτερα αποτελέσματα, συνεπώς επιλέχτηκε αυτός ο αριθμός threads. Στο worst-case scenario, **7x7** ως επιλογή περιοχής παίρνω τους παρακάτω χρόνους για την υλοποίηση με την Global Memory. Τέλος αναφέρω πως ο θόρυβος εξαρτάται αρκετά από τα filtSigma και patchSigma που διαλέγουμε. Θεωρώ πως έχω κάνει τις βέλτιστες επιλογές για αυτές τις εικόνες.

Photo_Dim (px*px)	64x64	128x128	256x256
Total Time (s)	1.885	5.181	59.340

3) Shared Memory

Σε αυτήν την υλοποίηση χρησιμοποιούνται 16x16 blocks και 16x16 threads για την περαιτέρω παραλληλοποίηση και βελτίωση του speedup της επεξεργασίας της εικόνας. Ανάλογα με το μέγεθος της εικόνας που χρησιμοποιούμε, το κάθε block θα επεξεργάζεται μικρότερο ή μεγαλύτερο αριθμό pixels και πιο συγκεκριμένα αν nxn η εικόνα (στην εργασία χρησιμοποιήσαμε συμμετρικές εικόνες, όπως ζητήθηκε), τότε θα έχουμε n/16 x n/16 = p pixels. Λαμβάνοντας ακόμη υπόψιν ότι κάθε block θα μπορεί να χρησιμοποιεί max 16x16=256 threads, σημαίνει αντίστοιχα ότι για τις 3 περιπτώσεις θα έχουμε :

i) 64x64 → 4x4 = 16 pixels/block → 256/16 = 16 threads/pixel
ii) 128x128 → 8x8 = 64 pixels/block → 256/64 = 4 threads/pixel
iii) 256x256 → 16x16 = 256 pixels/block → 256/256 = 1 thread/pixel

Και άρα τα αποτελέσματα που πήρα (πάλι για 7x7):

Photo_Dim (px*px)	64x64	128x128	256x256
Total Time (s)	1.618	1.987	6.916
Speedup	1.165	2.607	8.58

Παρατηρώ ότι για μεγαλύτερες εικόνες θα έχω μεγαλύτερο speedup. Αναφορικά με το **max error** σε ένα pixel της φιλτραρισμένης εικόνας σε σχέση με την αρχική, έχω τα εξής αποτελέσματα ενδεικτικά, για 7x7 (προφανώς για διαφορετικά μεγέθη περιοχών θα έχουμε διαφορετικά αποτελέσματα) :

Photo_Dim (px*px)	64x64	128x128	256x256
Max Error	0.1194	0.1137	0.1637

Η **μέση τιμή** του σφάλματος ενδεικτικά παρατίθεται παρακάτω :

Photo_Dim (px*px)	64x64	128x128	256x256
Mean	0.000044	0.0013	0.000006