Σταύρος Αντωνιάδης 8279
antoniadis@ee.auth.gr

Νίκος Χατζηβασιλειάδης 8268
Chantivn@ee.auth.gr

Φίλιππος Χρήστου 8276
fchristou@auth.gr

CUDA: NOISE RENDERING.

3Η ΑΝΑΦΟΡΑ ΕΡΓΑΣΙΑΣ ΣΤΑ ΠΑΡΑΛΛΗΛΑ ΚΑΙ ΔΙΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Η παρούσα εργασία αφορά την αποθορυβοποίηση εικόνων μέσω του αλγορίθμου Non-Loca Means, και την εφαρμογή αυτού σε cuda για την βελτίωση του χρόνου εκτέλεσης.

CUDA: NOISE RENDERING.

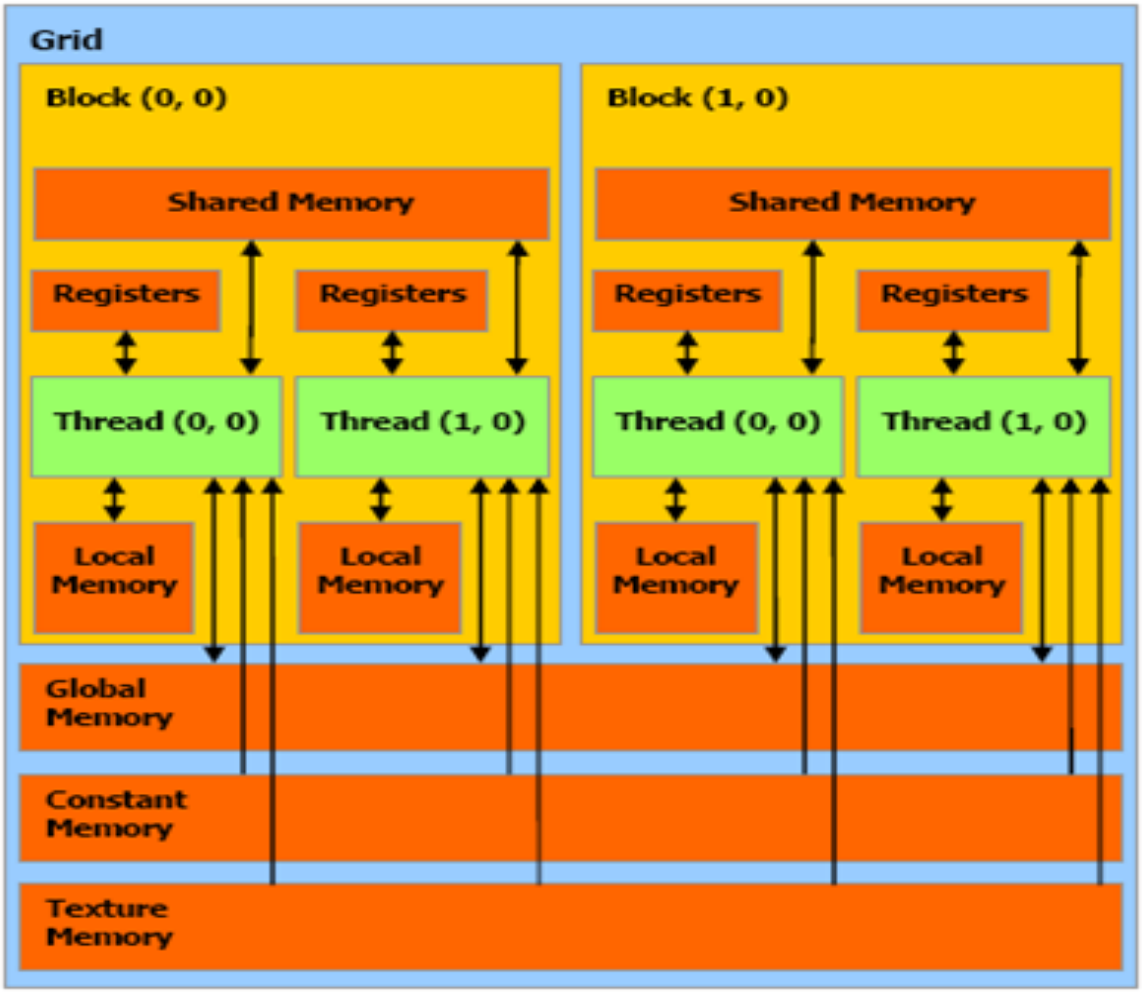
3Η ΑΝΑΦΟΡΑ ΕΡΓΑΣΊΑΣ ΣΤΑ ΠΑΡΑΛΛΗΛΑ ΚΑΙ ΔΙΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

Οι μετρήσεις, τα νούμερα και τα αποτελέσματα έχουν ληφθεί από το Διάδη.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΕΙΣΑΓΩΓΙΚΑ

Η Cuda είναι μια γλώσσα προγραμματισμού αρκετά διαδεδομένη και ενδέχεται να συνεχίσει να είναι για τα επόμενα χρόνια (και αυτό γιατί η Nvidia έχει διαβεβαιώσει τον τεχνολογικό κόσμο ότι θα υποστηρίζεται και από τις επόμενες γενιές των καρτών γραφικών που θα παράξει) . Η ιδιαιτερότητα της γλώσσας αυτής, είναι ότι η παραλληλοποίηση που πετυχαίνει δεν είναι στον CPU αλλά στην GPU. Η κάρτα γραφικών περιέχει εκατομμύρια threads τα οποία περιλαμβάνονται μέσα στα Blocks (χιλίαδες σε αριθμό ανάλογα με το μοντέλο της κάρτας γραφικών) του Grid της. Έτσι η cuda βελτιώνει θεαματικά το χρόνο για αλγορίθμους στους οποίους πρέπει να γίνεται η ίδια διαδικασία επανειλημμένα. Ένας τέτοιος αλγόριθμος είναι ο Non-Local Means, με τον οποίο μπορούμε να αποθορυβοποιήσουμε εικόνες.



Ο ΑΛΓΟΡΙΘΜΟΣ NON LOCAL MEANS

ΤΕΧΝΙΚΗ ΠΑΡΑΛΛΗΛΟΠΟΙΗΣΗΣ ΠΟΥ ΧΡΗΣΙΜΟΠΟΙΗΘΗΚΕ

Σκοπός της παραλληλοποίησης είναι κάθε thread της GPU να κάνει μια συγκεκριμμένη δουλειά. Αυτή είναι :

- Ένα cuda thread ανατίθεται σε κάθε pixel.
- Το κάθε Pixel έχει μια γειτονιά (3x3,5x5,7x7 pixels για τα διαφορετικά patch Size. Προσπαθήσαμε να περάσουμε την γειτονιά του κάθε pixel σε έναν πίνακα στον register του εκάστοτε thread, αλλά παρατηρήσαμε ότι όταν το patch Size ήταν μεγαλύτερο από 3x3 οι χρόνοι αντί να βελτιώνονται χειροτέρευαν και αυτό επειδή οι registers δεν μπορούν να αποθηκεύσουν μεγάλες δομές)
- Δουλειά του κάθε thread είναι να συγκρίνει όλους του γείτονές του με όλους τους γείτονες όλων των άλλων thread. Από αυτήν την σύγκριση θα προκύψει ένας βαθμωτός αριθμός.
- Η σύγκριση μεταξύ των γειτονιών των Pixel έγινε χρησιμοποιώντας την δεύτερη Νόρμα πολλαπλασιασμένη με έναν σταθερό αριθμό , ξεχωριστό για κάθε γείτονα, που προκύπτει από την Gaussian συνάρτηση.

Χρησιμοποιώντας την Shared Memory

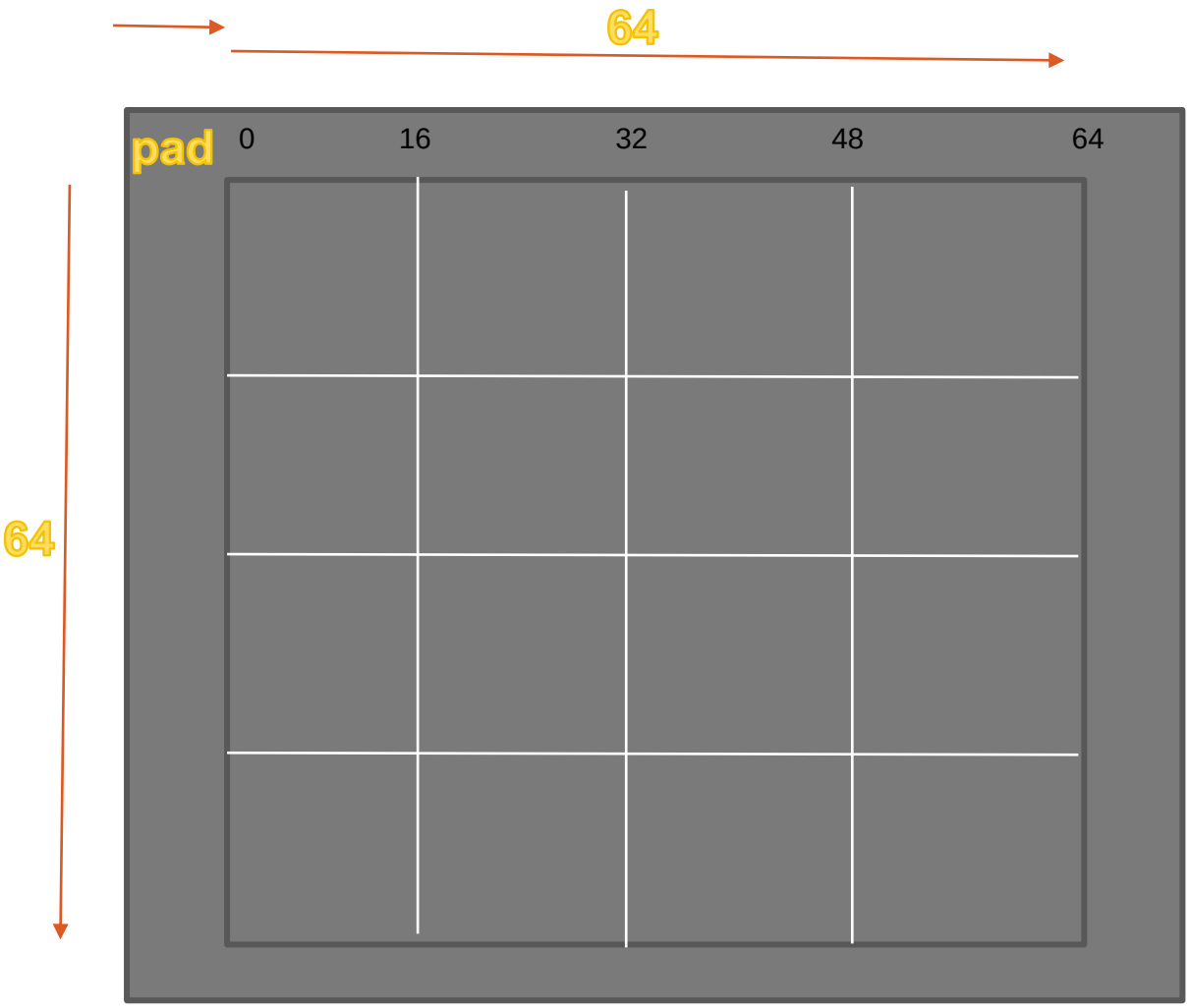
Η παραπάνω λογική συνεχίζει να υφίσταται και σε συνδυασμό με την χρησιμοποίηση της Shared Memory στοχεύουμε στην περαιτέρω βελτίωση των χρόνων.Η προσπέλαση των pixels από τα threads μέσω της Global memory (Constant στην περίπτωση μας, αλλά οι διαφορές δεν είναι πολύ μεγάλες) σε κάθε επανάληψη καθυστερεί αρκετά το πρόγραμμα μας. Για το λόγο αυτό φορτώνουμε τμήματα της εικόνας κάθε φορά στην Shared Memory, κάθε thread αναλαμβάνει να αντιγράψει ενα μέρος του τμήματος αυτού. Εκεί η προσπέλαση στοιχείων είναι κατά πολύ γρηγορότερη. Το κάθε thread προσπελάζει όλη την Shared Memory ταυτόχρονα με όλα τα άλλα. Εφόσον τελειώσουν τη σύγκριση τα threads , μετακινείται στην Shared Memory ένα άλλο τμήμα της Global memory (εδώ χρησιμοποιούμε επικάλυψη, δηλαδή το επόμενο τμήμα έρχεται και γράφεται στις θέσεις μνήμης του προηγούμενου). Η διαδικασία αυτή ολοκληρώνεται όταν διατρέξουμε όλη την Global Memory.

Η Shared Memory είναι πολλές φορές μικρότερη της Global (περίπου 49Kbytes) αλλά και πολλές φορές πιο γρήγορη. Η τεχνική που χρησιμοποιήθηκε παραπάνω είναι γνωστή ως <<Blocking>>.

Εφόσον τα στοιχεία της εικόνας είναι float me μέγεθος 4 bytes ,δεν τίθεται θέμα συγκρούσεων στην Shared Memory, αφού κάθε λέξη είναι 4 bytes και τα wraps των 32 thread μπορούν ταυτόχρονα να προσπελάζουν τις διάφορες λέξεις εφόσον χρησιμοποιούνται και τα 32 banks.

Το πρόβλημα που αντιμετωπίσαμε ήταν ότι ενώ είχαμε [32 32] threads δεν μπορούσαμε να περάσουμε ένα 32x32 υποτμήμα της εικόνας μιάς και θέλαμε κάθε pixel να έχει την γειτονία του. Για το λόγο αυτό σπάσαμε την εικόνα (Global Memory) σε ένα υποτμήμα 16x16 ώστε να έχουμε στην

διάθεση μας έναν επαρκή αριθμό threads για να περάσουμε και τους γείτονες των ακριανών Pixels (padding). Τα threads λοιπόν που χρησιμοποιήσαμε για να αντιμετωπίσουμε το πρόβλημα ήταν 16x16 συν έναν αριθμό threads που εξαρτόταν κάθε φορά από το μέγεθος της εικόνας και τον αριθμό του patch Size.



Αυτό που κάνουμε σχηματικά είναι (για την εικόνα 64x64) να βάζουμε στην shared memory κάθε φορά ένα από τα παραπάνω τετραγωνάκια **συν τους γείτονες των ακριανών**.

ΜΕΤΡΗΣΕΙΣ

Παρακάτω φαίνονται οι χρόνοι σε πίνακες.Οι διαφορές στον χρόνο ανάμεσα σε σειριακή υλοποίηση και με υλοποίηση cuda είναι εμφανής.

Σειριακά

		Μέγεθος Εικόνας		
		64x64	128x128	256x256
Μέγεθος Γειτονιάς	3x3	0,87	17,12	Size error
	5x5	0,91	17,50	Size error
	7x7	0,94	18,76	Size error

Με cuda:

		Μέγεθος Εικόνας		
		64x64	128x128	256x256
Μέγεθος Γειτονιάς	3x3	0.02970	0.2307	2.293
	5x5	0.0664	0.5253	5.263
	7x7	0.1204	0.9553	9.57

Εικόνες

Οι εικόνες βρίσκονται σε διαφορετικό αρχείο επειδή ήταν πολλές και η τοποθέτησή τους εδώ δεν θα διευκόλυνε τον αναγνώστη. Οι εικόνες βρίσκονται στον φάκελο pictures. Παρακάτω ακολουθεί ο σχολιασμός τους.

Σχολιασμός Εικόνων

Παρατηρούμε ότι η filtered εικόνα και στην περίπτωση του σειριακού κώδικα και στην περίπτωση του κώδικα με Cuda είναι ίδια.Συμπεραίνουμε λοιπόν ότι η υλοποίηση μας είναι σωστή. Γενικότερα παρατηρούμε ενα καλό αποτέλεσμα στην αποθορυβοποιημένη εικόνα με την μέθοδο της non local means.

Μας ζητήθηκε ακόμα να σχεδιάσουμε μία είσοδο, ώστε να ελέγξουμε την ορθότητα της υλοποίησης μας. Δημιουργήσαμε λοιπόν έναν πίνακα όπου στην μία μεριά θα έχει μόνο μηδενικά και στην άλλη μονο άσσους. Εισάγαμε σφάλμα σε αυτήν και τρέξαμε τον αλγόριθμο μας (Βλέπε στον φάκελο pictures). Παρατηρούμε ότι αφαιρέθηκαν όλα τα ανομοιόμορφα τμήματα που εισήγαγε το σφάλμα και η εικόνα πήρε την αρχική της μορφή όπως άλλωστε περιμέναμε.

ΣΥΜΠΕΡΑΣΜΑΤΙΚΑ

Ρίχνοντας μία προσεκτικότερη ματιά στον πίνακα των χρόνων βλέπουμε την τρομερή βελτίωση που πετυχαίνουμε με την υλοποίηση της non local means σε Cuda. Η επιτάχυνση φτάνει στην περίπτωση μας μέχρι και x74, γεγονός που μας υποδυκνύει το μεγαλείο και την ισχύ της GPU.