Κωνσταντίνος Χατζής

ΑΕΜ: 9256

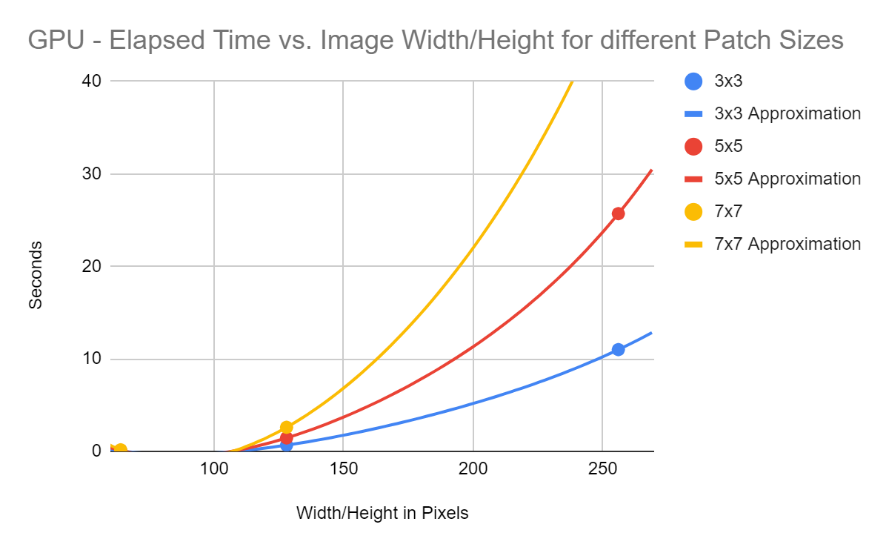
[github.com/kostascc/PDS-Project-3](https://github.com/kostascc/PDS-Project-3)

Στη Τρίτη εργασία ασχολούμαστε με το φίλτρο Non Local Means και την υλοποίησή του σε CUDA. Ακολουθεί η παρουσίαση του κώδικα, ορισμένες μετρήσεις και σχόλια / προτεινόμενες βελτιώσεις για την απόδοση του αλγορίθμου σε CPU και GPU, και εικόνες που αποθορυβοποιήθηκαν με το φίλτρο αυτό.

## Κώδικας

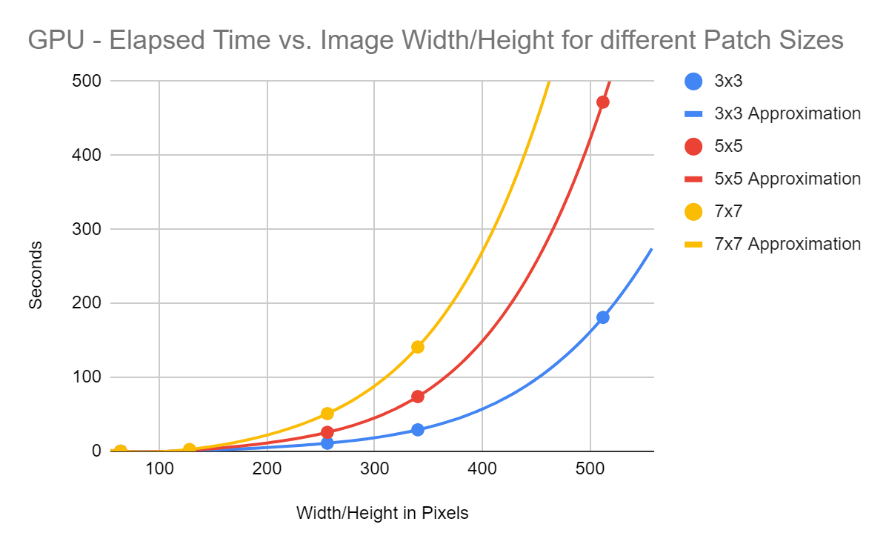
Για την παραγωγή του αλγορίθμου χρησιμοποιήθηκε το αρχείο Matlab που περιέχεται στην εκφώνηση της εργασίας, και με βάση αυτό παράχθηκε ο κώδικας για CPU με χρήση OpenMP (είναι αντίστοιχος με αυτόν της GPU που ακολουθεί). Στη συνέχεια έγινε μετατροπή του σε CUDA.

Συγκεκριμένα, γίνεται κλήση του Kernel για δισδιάστατο πλήθος Block, όσα και τα Patch που περιέχει η εικόνα (πχ 251x251 για εικόνα 256x256 και Patch Size 5x5). Κάθε ένα από τα Βlock αυτά περιέχει επίσης δισδιάστατο πλήθος Τhread (8x8, το διπλάσιο του πλήθους Warp που εκτελεί ταυτόχρονα ένας πολυεπεξεργαστής από τους 16 διαθέσιμους). Αυτά τα Thread μοιράζονται όλα τα Patch της εικόνας στα οποία πρέπει να γίνει αναζήτηση για βάρη, ανά το Block στο οποίο ανήκουν (δηλαδή [256-5]x[256-5] Patch, μείον ενός, αυτού για το οποίο γίνεται η αναζήτηση).

 Ένα Patch σε κάθε Block - το Patch για οποίο γίνεται η αναζήτηση - βρίσκεται πάντα στην *L2 Cache* απαράλλαχτο (για αποφυγή των Cache Miss στη Global Memory), και ένα ακόμα Patch άδειο. Στο δεύτερο γίνονται διαδοχικές προσθέσεις Pixel, ανάλογα της απόστασης που έχει το εκάστοτε Patch με το οποίο γίνεται σύγκριση. Στο τέλος, το σύνολο των βαρών που υπολογίστηκαν, διαιρείται από όλα τα Pixel του τελικού Patch, και αυτό εκχωρείται πίσω στην Global Memory για να εξαχθεί ως εικόνα.

\*Για τη βελτίωση της απόδοσης, έγινε χρήση συναρτήσεων της CUDA ( \_\_expf, \_\_fdiv\_rn κτλ.) που κάνουν ταχύτερους υπολογισμούς σε Floating Point μεγέθη μονής ακρίβειας, με το μειονέκτημα των μεγαλύτερων σφαλμάτων.

## Απόδοση

Από τα πρώτα δυο διαγράμματα φαίνεται ότι ο αλγόριθμος κοστίζει χρονικά αρκετά (σε αντίθεση με άλλα φίλτρα, όπως αυτό του kNN, ή σε σχέση με τις βελτιωμένες εκδόσεις του NLM που συζητώνται στο τέλος), καθώς σε μια μικρή εικόνα, στα 256x256 pixel, απαιτούνται περίπου 25 δευτερόλεπτα για την αποθορυβοποίησή της.

Οι δοκιμές που παρουσιάζονται, καθώς και η παραγωγή των εικόνων, έγιναν σε κάρτα γραφικών GTX 1650, η οποία παρέχει 4 GB Global Memory, και 1 MB Shared Memory (48KB ανά Thread Block). Ο κώδικας γράφτηκε με σκοπό την καλύτερη απόδοση στα χαρακτηριστικά της συγκεκριμένης GPU, και ενδέχεται να μη λειτουργεί σε παλαιότερες τεχνολογίες CUDA.

Παρατηρείται από τη διάρκεια της αποθορυβοποίησης, για Patches μεγέθους 7x7 έναντι των μικρότερων, ότι ένα σημαντικό μέρος της καθυστέρησης που επιβαρύνει τον αλγόριθμο είναι η χρήση της Global Memory χωρίς κάποια βελτίωση της τοπολογίας των Pixel (καθώς κάθε Patch κάνει τουλάχιστον τόσα Miss, όσο είναι το ύψος του). Αυτό έχει ως αποτέλεσμα, το Cache Miss Rate να είναι μεγάλο και ο αλγόριθμος να είναι αργός. Την ίδια στιγμή, και σε μικρότερα Patches υπάρχει μεγάλο Miss Rate (>1/5 για 5x5 Patch), εξού και ο αλγόριθμος δεν είναι γρήγορος για οποιοδήποτε μέγεθος Patch.

|  |
| --- |
|  |
| Πίνακας Pixel, όπου κόκκινο είναι ένα Patch. |
|  |
| Ένας από τα αντίγραφα των Pixel, όπου κόκκινο είναι το Patch που μεταλλάχθηκε σε διάνυσμα. |

|  |  |  |  |
| --- | --- | --- | --- |
| Αποθορυβοποίηση |  |  |  |
| Διαφορά (Residual) |  |  |  |
|  |  |  | 40 |

Μια πιθανή βελτίωση που θα μπορούσε να προταθεί για μικρές εικόνες, είναι η δημιουργία μετατοπισμένων αντιγράφων της εικόνας στη μνήμη, όσων και το πλάτος των Patch, και η μεταφορά των διαφορετικών σειρών των Patch στην ίδια σειρά, μετατρέποντάς τα έτσι σε 1D διανύσματα (δίνεται ένα σκιαγράφημα στα δεξιά). Τα αντίγραφα θα δώσουν ξανά 1D διανύσματα, τα οποία όμως ξεκινούν από αντίστοιχες θέσεις δεξιότερα κ.ο.κ.

Αξίζει να σημειωθεί ότι στη CPU η αποθορυβοποίηση εικόνας 128x128 διήρκησε 252 δευτερόλεπτα.

## Αποτελέσματα

|  |  |
| --- | --- |
|  |  |
| Εικόνες με επιπρόσθετο θόρυβο | |

Στις παρακάτω εικόνες φαίνεται πως αν και ο αλγόριθμος είναι αρκετά ‘βαρύς’, στην πράξη είναι υπέρ ικανός να αποθορυβοποιήσει μια εικόνα μικρού μεγέθους, σε λογικό χρονικό διάστημα.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Οι εικόνες που παρατίθενται παραπάνω έχουν παραχθεί για Patch μεγέθους 5x5. Παρόλο που στο έγγραφο δεν φαίνονται ιδιαίτερες διαφορές μεταξύ τους, με μια μεγέθυνση στα αρχεία που συμπεριλαμβάνονται στο Project μπορεί να δει κανείς περισσότερες λεπτομέρειες.

Στο πρώτο σύνολο εικόνων, επιλέχθηκε ένας τοίχος κτιρίου στον οποίο δεν υπάρχουν πολλές ομοιότητες μεταξύ διαφορετικών σημείων (όπως τα τούβλα, που σε κάθε Patch έχουν ελαφρώς διαφορετική γωνία), αλλά ταυτόχρονα φαίνονται αρκετές λεπτομέρειες. Ως αποτέλεσμα, το φίλτρο έχει καταφέρει να αφαιρέσει αρκετό θόρυβο από σημεία όπως το δάπεδο, το χρωματισμό των τούβλων και του παραθύρου, τα πολύ φωτεινά ή πολύ σκοτεινά σημεία. Την ίδια στιγμή όμως αφαίρεσε και πολύ λεπτομέρεια από τους σχηματισμούς των τούβλων, καθώς δεν υπήρχαν Patch αρκετά όμοια για να προσδιορίσουν τη λεπτομέρεια αυτή.

Η πληροφορία που ‘λείπει’ από τον καθαρισμό της εικόνας, φαίνεται στις εικόνες με τίτλο ‘Διαφορά’ (Residual), όπου έγινε αφαίρεση της αρχικής (καθαρής) εικόνας από την αποθορυβοποιημένη. Τα σημεία που παρουσιάζονται με μαύρο είναι αυτά που δεν υπέστησαν μεγάλη απώλεια λεπτομέρειας/πληροφορίας, ενώ με πιο έντονο χρώμα είναι αυτά που είχαν εντονότερη απώλεια από το φίλτρο. Στο τελευταίο σύνολο παρουσιάζονται οι αποθορυβοποιήσεις δυο εικόνων, για τις βέλτιστες τιμές ‘σ’.

## Βελτίωση

Αν και ο αλγόριθμος Non Local Means δίνει καλό αποτέλεσμα στην αποθορυβοποίηση, σε κάθε περίπτωση δεν μπορεί να χρησιμοποιηθεί για μεγάλες εικόνες, ή ακόμα και για καθαρισμό ζωντανής ροής εικόνων. Γι’ αυτό θα μπορούσαμε να βελτιωθεί το φίλτρο αυτό με:

* Σύγκριση μόνο του κεντρικού Pixel σε κάθε Patch, με τη σκέψη ότι το Patch αυτό θα έχει περίπου το ίδιο βάρος ακόμα και αν συγκριθεί ολόκληρο,
* Αναζήτηση μόνο σε κοντινά Patch, καθώς αυτά έχουν μεγαλύτερη πιθανότητα να είναι όμοια,
* Χρήση kNN-Search (με Vantage Point Tree για πολύ μεγάλες εικόνες) για ταχύτερη εύρεση των Patch με τη μεγαλύτερη ομοιότητα.