Παράλληλα και διανεμημένα Συστήματα – 3η Εργασία

Στο τρίτο παραδοτέο του μαθήματος “Παράλληλα και Διανεμημένα Συστήματα” παρουσιάζεται η υλοποίηση του αλγορίθμου αποθορυβοποίησης εικόνας **Non Local Means** [1] σε περιβάλλον CUDA. Τα αρχεία του πρότζεκτ βρίσκονται εδώ κι εδώ.

Non Local Means – Αλγόριθμος

Ο αλγόριθμος Non Local Means, σε αντίθεση με τους περισσότερους αλγόριθμους αποθορυβοποίησης που είναι διαδεδομένοι, για την απόρριψη του θορύβου σε ένα συγκεκριμένο pixel της εικόνας, δε λαμβάνει υπόψη μόνο τις κοντινές σε αυτό τιμές, αλλά εφαρμόζεται ένας σταθμισμένος μέσος όρος όλων των pixel της εικόνας, το οποίο περιγράφεται από την παρακάτω εξίσωση (1):

$$\hat{f}(\mathbf{y}) = \sum_{\mathbf{x}\epsilon\Omega} w(\mathbf{x},\mathbf{y}) \cdot f(\mathbf{x}) \qquad x, y\epsilon\Omega \subseteq \mathbb{N}^2$$

όπου Ω είναι το πεδίο ορισμού της εικόνας (εδώ μελετήθηκαν διδιάστατες grayscale εικόνες-Οι τιμές των pixel της εικόνας ανήκουν στο [0,1]) .Το εκάστοτε w(x,y) επηρεάζεται από το βαθμό ομοιότητας της γειτονιάς του pixel **x** με τη αντίστοιχη γειτονιά του pixel **y.** Ο βαθμός ομοιότητας ποσοτικοποιείται με τον υπολογισμό της element-wise απόστασης των αντίστοιχων σημείων της κάθε γειτονιάς, αφού πρώτα έχει εφαρμοστεί σε κάθε γειτονιά ένας Gaussian πυρήνας ίδιων διαστάσεων με κέντρο το σημείο ενδιαφέροντος (x,y αντίστοιχα), όπως περιγράφει η παρακάτω σχέση:

$$w(i,j) = \frac{1}{Z(i)} e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}}$$

Όπως φαίνεται, εκτελείται και μια κανονικοποίηση στα εν λόγω βάρη, διαιρώντας με το άθροισμα όλων των βαρών για ένα συγκεκριμένο pixel (όρος Z(i)), ώστε το pixel εξόδου να έχει μια τιμή στο [0,1], κάτι που αποτυπώνεται από τον παρακάτω τύπο:

$$Z(i) = \sum_j e^{-\frac{||f(\mathcal{N}_i) - f(\mathcal{N}_j)||^2_{G(a)}}{\sigma^2}}$$

V0 Σειριακή έκδοση

Αρχικά, υλοποιήθηκε μια έκδοση που εκτελεί τον NLMeans αποκλειστικά σε CPU. Πιο συγκεκριμένα, λαμβάνεται η εικόνα εισόδου, σε αυτήν προστίθεται AWGN (Additive White Gaussian Noise, Mean=0) με τυπική απόκλιση που ορίζεται στατικά στο αρχείο parameters.hpp (περισσότερες πληροφορίες στο appendix εκτέλεσης του κώδικα), στη συνέχεια γίνεται padding στα άκρα της εικόνας (υλοποιήθηκε και συνάρτηση που εκτελεί zero-padding, αλλά και το padding-mirroring που εφαρμόζεται στο αρχείο nonlocalmeans.m και δίνει καλύτερα αποτελέσματα) και κατόπιν καλείται η κύρια συνάρτηση nonlocalmeans() που επιστρέφει την αποθορυβοποιημένη εκδοχή της εικόνας. Από αυτήν αφαιρούνται τα padded σημεία στα άκρα της εικόνας και τέλος αποθηκεύονται αυτή η εικόνα και η εικόνα με το θόρυβο σε ξεχωριστά αρχεία. Σχετικά με την κύρια συνάρτηση nonlocalmeansCPU(), εκτελείται ένας τετραπλός βρόχος επανάληψης (για κάθε σημείο 2 διαστάσεων με κάθε σημείο στην εικόνα), όπου κάθε φορά βρίσκεται το βάρος w(x,y) της εξίσωσης (1), μέσω της συνάρτηση patchFilt(), υπολογίζοντας τις σταθμισμένες με Gaussian Kernel αποστάσεις των γειτονιών των δύο pixel (οι διαστάσεις των γειτονιών [PATCHxPATCH] αποτελούν παράμετρο του αλγορίθμου και εισάγουν ένα trade-off μεταξύ ποιότητας αποθορυβοποίησης και χρονικής πολυπλοκότητας).

V0 Multithreaded έκδοση

Επίσης, καθαρά για λόγους πληρότητας των αποτελεσμάτων, υλοποιήθηκε και μια απλή multithreaded έκδοση, οι οποία χρησιμοποιεί τη βιβλιοθήκη OpenMP που μελετήθηκε στα πλαίσια της πρώτης εργασίας και επιβάλλει την εκτέλεση του 4πλού βρόχου επανάληψης σε πολλούς επεξεργαστές ενός υπολογιστή για τη βελτίωση του χρόνου, σπάζοντας ουσιστικά το grid της εικόνας σε επιμέρους disjoint blocks, καθένα από τα οποία εκτελείται ταυτόχρονα σε διαφορετικό πυρήνα της CPU. Το αντίστοιχο αρχείο είναι το nlmeans_omp.cpp και απαιτεί την προσθήκη του flag -fopenmp κατά τη μεταγλώττιση.

V1 CUDA Global Memory έκδοση

Στη συνέχεια, έγινε προσπάθεια υλοποίησης αντίστοιχης έκδοσης με τη V0 η οποία να τρέχει σε GPU της NVIDIA μέσω του API της CUDA, αρχικά αγνοώντας την ύπαρξη όλων των άλλων (γρηγορότερων) τύπων μνήμης πέραν της Global που διαθέτει μια οποιαδήποτε GPU. Για το σκοπό αυτό, υλοποιήθηκε ο kernel που βρίσκεται στο αρχείο ./nlmeans_V1.cu. Για εικόνες μεγέθους που μελετάμε στα πλαίσια αυτής της εργασίας (από 64x64 έως 256x256 pixel), αποδείχθηκε ικανοποιητική η απλουστευμένη

θεώρηση ότι κάθε pixel της εικόνας γίνεται map σε ένα thread ενός block, ενώ, επίσης, σε μικρές εικόνες κάθε γραμμή της εικόνας αντιστοιχίζεται σε ένα block, οπότε ο πυρήνας γίνεται spawn με το grid <<N,N>>, για τετράγωνες εικόνες NxN.

V2 CUDA Shared Memory έκδοση

Όπως προκύπτει από τη δομή του προβλήματος, παρατηρείται μια επικάλυψη μεταξύ γειτονιών γειτονικών pixels, η οποία μάλιστα αυξάνεται όσο αυξάνουμε το μέγεθος του παράθυρου των patches. Αυτή η παρατήρηση οδήγησε στην υλοποίηση της έκδοσης V2, στην οποία γίνεται προσπάθεια να χρησιμοποιηθούν οι γρηγορότεροι τύποι μνημών της κάρτας γραφικών για τη βελτίωση του χρόνου εκτέλεσης του προβλήματος. Πιο αναλυτικά, το grid της εικόνας χωρίστηκε νοητά σε μη επικαλυπτόμενα τετράγωνα TILES (ασφαλώς με μέγεθος μεγαλύτερο από αυτό των patches) και έγινε mapping του κάθε TILE σε ένα block της GPU, ενώ, επίσης, κάθε block της GPU γίνεται spawn με νήματα ίσα με το πλήθος ενός, επίσης τετράγωνου, υπερ-παραθύρου BLOCKS*BLOCKS που περιέχουν τα TILES, κάτι που είναι απαραίτητο όπως θα φανεί στη συνέχεια. Πιο παραστατικά, το παραπάνω mapping απεικονίζεται στο παρακάτω σχήμα (για αποφυγή της σύγχησης, παντού γίνεται χρήση κεφαλαίων για τα υπερ-παράθυρα BLOCKS και μικρών για τα blocks που αποτελούν κομμάτια ενός grid της GPU).

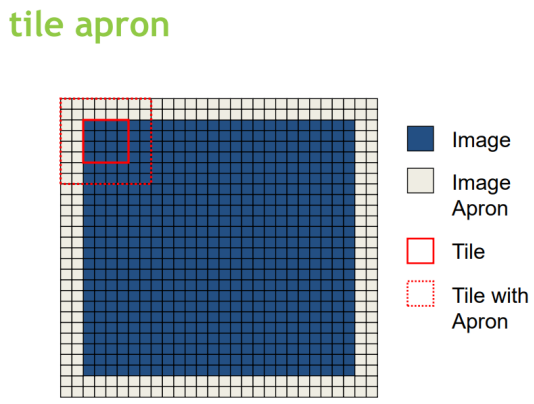


Figure 1: Partitioning εικόνας ώστε να γίνει mapped στα blocks της GPU. Πηγή : σελ.9

Στη συνέχεια αυτό που γίνεται είναι ότι κάθε block φορτώνει το BLOCK του στη shared memory, ενώ, επίσης, δεσμεύει άλλο ένα ίδιου μεγέθους BLOCK στη shared memory, το οποίο χρησιμοποιείται ώστε να φορτώνεται και κάθε άλλο BLOCK της εικόνας για να υπολογιστούν τα βάρη του εκάστοτε BLOCK (ουσιαστικά του εσωτερικού TILE). Το κέρδος από αυτή τη θεώρηση είναι η μείωση των προσβάσεων στη Global Memory σε σχέση με το V1, αφού έχοντας δυο TILES στην κοινή μνήμη μπορούμε να υπολογίσουμε τα βάρη μεταξύ δύο οποιωνδήποτε pixels εκατέρωθεν χωρίς να χρειάζεται να φορτώνουμε πολλές φορές pixel overlapping παραθύρων ξανά και ξανά. Φυσικά, υπάρχουν και κάποιοι περιορισμοί του υλικού που εισάγουν όπως θα παρατηρηθεί και στη συνέχεια κάποια trade-offs στην απόδοση αυτής της έκδοσης, όπως για παράδειγμα ο περιορισμός της ίδιας της Shared μνήμης (που περιορίζει τον αριθμό των blocks που μπορούν να εκτελούνται ταυτόχρονα και επηρεάζεται από το πλήθος των SM που έχει μια κάρτα) και το μέγεθος των BLOCKS (μεγαλύτερα BLOCKS μειώνουν την επικάλυψη των reads, αλλά το μέγεθος περιορίζεται από το μέγιστο επιτρεπτό πλήθος threads ανα block, όπως και από έμμεσους λόγους – πολύ μεγάλα BLOCKS οδηγούν σε μείωση της παραλληλίας, ενώ, επίσης, και μεγάλο πλήθος από threads οδηγεί αναπόφευκτα και στην πιο αργή εκτέλεση του block, αφού κάθε στιγμή μόνο warps των 32 threads εκτελούνται παράλληλα). Κατά τα άλλα, χρησιμοποιείται το ίδιο pipeline με τη σειριακή έκδοση (read εικόνας- θορυβοποίηση – padding - εκτέλεση της nonlocalmeans() -unpad – write εικόνας στο δίσκο), ενώ και η device συνάρτηση patchFilt() είναι ίδια με αυτή της V1 (απλώς τώρα τα patches που λαμβάνει είναι αποθηκευμένα στην shared memory, ενώ ο kernel βρίσκεται στην constant memory). Τα BLOCKS της εικόνας σαρώνονται σύμφωνα με την πορεία που φαίνεται στην παρακάτω εικόνα (από τα σκοτεινότερα προς τα φωτεινότερα).

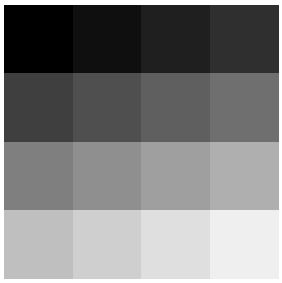


Figure 2: Πορεία σάρωσης των TILES

Εαν δε μπορούν να υπολογιστούν όλα τα BLOCKS ταυτόχρονα (λόγω περιορισμένης shared memory) αλλά μόνο

τα BLOCK/K = SMALL_BLOCK, τότε υπολογίζονται τα πρώτα SMALL_BLOCK και στη συνέχεια το κάθε block της gpu κάνει άλμα στο αντίστοιχο BLOCK του στο επόμενο SMALL_BLOCK μέχρις ότου φτάσει πέραν των ορίων της εικόνας, οπότε και τερματίζει.

Αποτελέσματα – Σχολιασμός

Οι εκδόσεις που υλοποιήθηκαν και παρουσιάζονται παραπάνω εκτελέστηκαν τοπικά, σε λάπτοπ με επεξεργαστή Intel i7-4720HQ και κάρτα Nvidia GeForce GTX960M, αλλά και στη συστοιχία του ΑΠΘ, η οποία διαθέτει κάρτες Tesla P100 (η V0 εκτελείται στο batch partition, ενώ οι αντίστοιχες εκδόσεις της CUDA στο gpu partition).

Figure 3: Benchmark PNG εικόνες για τις μετρήσεις χρόνων

Μετρήσεις Χρόνων Εκτέλεσης

I) Τοπικά

Μέγεθος Παραθύρου 3x3

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	400 ms	122 ms	6.8 ms	4 ms
128x128	6370 ms	2338 ms	58 ms	50 ms
256x256	107031 ms	41740 ms	730 ms	724 ms

Μέγεθος Παραθύρου 5x5

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	715 ms	297 ms	78 ms	15 ms
128x128	13340 ms	5720 ms	1142 ms	152 ms
256x256	195302 ms	77219 ms	17631 ms	2464 ms

Μέγεθος Παραθύρου 7x7

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	1190 ms	413 ms	142 ms	21 ms
128x128	20812 ms	8108 ms	2215 ms	251 ms
256x256	306203 ms	105540 ms	Timed out*	3610 ms

* >30 sec

II) Συστοιχία

Μέγεθος Παραθύρου 3x3

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	584 ms	234 ms	9.4 ms	14.4 ms
128x128	8986 ms	2607 ms	51 ms	82 ms
256x256	204597 ms	56102 ms	522 ms	488 ms

Μέγεθος Παραθύρου 5x5

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	990 ms	458 ms	50 ms	35 ms
128x128	15884 ms	4460 ms	465 ms	207 ms
256x256	253229 ms	67879 ms	5812 ms	912 ms

Μέγεθος Παραθύρου 7x7

Pixels/ Versions	Sequential	OpenMP	CUDA Global	CUDA Shared
64x64	1483 ms	592 ms	84 ms	60 ms
128x128	23597 ms	6345 ms	659 ms	274 ms
256x256	283025 ms	100118 ms	11075 ms	1418 ms

Όπως είναι φανερό από τους παραπάνω χρόνους, επιτυγχάνεται μια θεαματική επιτάχυνση από τις εκδόσεις που εκτελούνται στη CPU στις CUDA υλοποιήσεις που κινείται στις τάξεις του x20 μέχρι x100+, ανάλογα και με το μέγεθος του παραθύρου, αλλά και το μέγεθος της εικόνας, κάτι που ξεπερνάει κατά πολύ τις επιδόσεις που μπορούν να επιτευχθούν με CPU multithreading (x8~x16 σε δυνατό PC στην ιδανική περίπτωση). Όσον αφορά τη σύγκριση μεταξύ

των GPU εκδόσεων σχετικά με το χρόνο εκτέλεσης του πυρήνα, παρατηρούμε μια σαφή επιτάχυνση στην Shared Memory έκδοση, η οποία φαίνεται να μεγαλώνει με την αύξηση του μεγέθους της εικόνας και (σαφώς) με την αύξηση του παραθύρου των γειτονιών, αφού, όπως εξηγήθηκε και παραπάνω παρουσιάζεται μεγαλύτερο overlapping σε διπλανές γειτονιές, κάτι που εκμεταλλεύται η V2 και μειώνει σημαντικά τις αναγνώσεις από την αργή off-chip Global Memory. Κάτι αξιοσημείωτο είναι ότι στην αρκετά δυνατών χαρακτηριστικών κάρτα Tesla P100 της συστοιχίας, για μέγεθος παραθύρου 3x3 (που υπάρχει το ελάχιστο overlapping) η Global έκδοση είναι κατάτι ταχύτερη για τις μικρές εικόνες, κάτι που μπορεί να αποδοθεί στο μεγάλο Memory Bandwidth της συγκεκριμένης κάρτας. Μια βελτίωση που δοκιμάστηκε εδώ είναι να χρησιμοποιηθεί μια υβριδική έκδοση V1.5 που αποτελεί μια αντιγραφή της V1, στην οποία ο Gaussian Kernel αποθηκεύεται στη shared memory. Αυτό φάνηκε να αποδίδει καλύτερα αποτελέσματα στις 2 περιπτώσεις που υστερεί η V2 στη συστοιχία, καθώς οι χρόνοι μειώθηκαν κατά 60%. (αρχείο nlmeans_V2new.cu)

Παρατηρούμε, επίσης, και κάποια αποτελέσματα αποθορυβοποίησης εικόνων. Για τα πειράματα χρησιμοποιήθηκαν οι τέσσερις εικόνες του σχήματος (η εικόνα που δόθηκε στα πλαίσια της εκφώνησης από το house.mat 64x64, δυο εικόνες 128x128 η καθεμία και μία μεγέθους 256x256 pixels). Σε κάθε εικόνα εισάγεται τεχνητά λευκός Gaussian θόρυβος μηδενικής μέσης τιμής και τυπικής απόκλισης που ορίζεται στο αρχείο parameters.hpp ως NOISE_STD. Για τους συντελεστές patchSigma και filterSigma που χρησιμοποιούνται στο φιλτράρισμα, κρατήθηκαν οι προκαθορισμένες τιμές (5/3 και 0.02 αντίστοιχα), καθώς δίνουν ικανοποιητικά αποτελέσματα για μικρές και μεσαίες στάθμες προσθετικού θορύβου, όπως δείχνουν και τα αποτελέσματα.

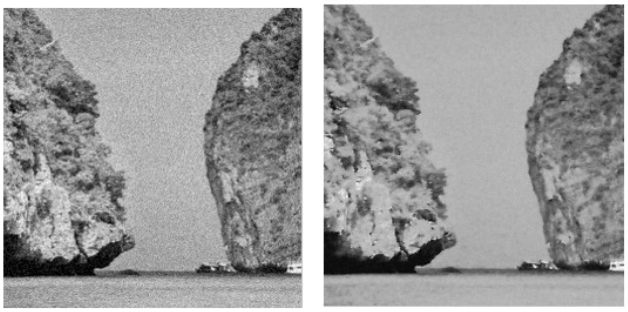
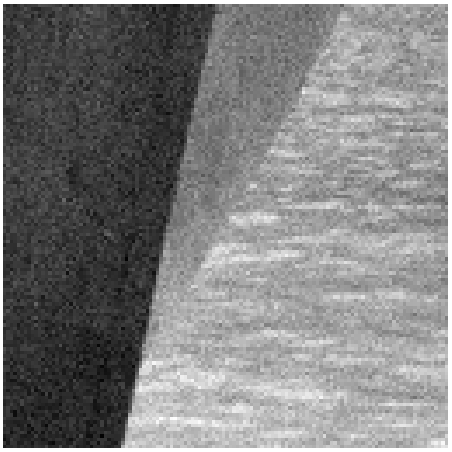


Figure 4: 256-beach.PNG Πριν-Μετά την αποθορυβοποίηση (Window: [5x5], NOISE_STD:0.45)



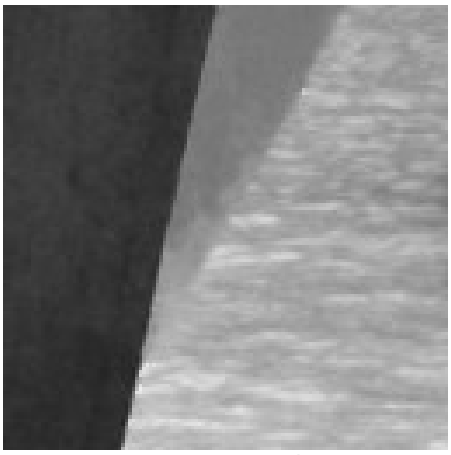


Figure 5: 128-boat.png Πριν-Μετά την αποθορυβοποίηση (Window:[7x7], NOISE_STD:0.03)

Επικύρωση αποτελεσμάτων – Σχόλια

Για την επικύρωση των αποτελεσμάτων ελέχθηκε η ομοιότητα που εμφανίζουν τα αποτελέσματα που προκύπτουν από τις εκδόσεις V0,V1,V2 με το αρχείο pipeline_non_local_means.m της εκφώνησης (παρόλο που ίσως δεν έχει πολύ νόημα μια τέτοια σύγκριση, καθώς όλες οι πράξεις στις εκδόσεις είναι μεταξύ floats σε αντίθεση με την περίπτωση της MATLAB όπου χρησιμοποιούνται doubles). Το μέσο απόλυτο σφάλμα (Mean Absolute Error – pixelwise) ήταν ~ 2*10^(-4) για τις εικόνες εισόδου. Οι png εικόνες εξόδου των εκδόσεων μεταξύ τους μπορούν να συγκριθούν είτε με τη βοήθεια του αρχείου differences.c, είτε με τη χρήση αυτής της σελίδας (περισσότερες πληροφορίες στο αρχείο md της σελίδας στο Github).

vtmoscho@ece.auth.gr

Appendix – Εκτέλεση Κώδικα

Για την επιτυχή εκτέλεση του κώδικα είναι απαραίτητη η ύπαρξη στο σύστημα της βιβλιοθήκης libpng της C*. Σε περίπτωση cross-platform προβλημάτων συμβατότητας, δίνεται η δυνατότητα στον αναγνώστη να εκτελέσει τον κώδικα μέσω της συστοιχίας του Α.Π.Θ, όπου επίσης δοκιμάστηκαν οι εκδόσεις και τρέχουν επιτυχώς. Στο φάκελο του πρότζεκτ, μάλιστα, παρατίθενται και 2 scripts (‘job.sh’ και ‘seq_job.sh’) τα οποία μπορεί να κάνει submit κάποιος στο σύστημα και να παρατηρήσει τη συμπεριφορά των CPU και CUDA εκδόσεων. Επίσης, να σημειωθεί ότι για τον κώδικα χρησιμοποιούνται κάποιες παράμετροι, οι οποίες ορίζονται από το χρήστη (π.χ. R=PATCH/2, NOISE_STD, patchSigma, filterSigma) και βρίσκονται στο αρχείο “./headers/parameters.hpp”. Οι τιμές είναι προκαθορισμένες, αλλά μπορεί ο αναγνώστης εάν επιθυμεί να αλλάξει κάποιες από αυτές στατικά για να δει τη συμπεριφορά του αλγορίθμου. Περισσότερες πληροφορίες για το πώς μπορεί κάποιος να τρέξει τις εκδόσεις βρίσκονται στο Markdown αρχείο του πρότζεκτ στο GitHub. Τέλος, προστέθηκε και μια παράμετρος B (επίσης στο parameters.hpp) η οποία ρυθμίζει την μεταβλητή MAX_BLOCKS (μέσω της σχέσης MAX_BLOCKS = B*(6000/(BLOCKS*BLOCKS)) που περιορίζει το μέγιστο αριθμό από blocks που μπορούν να δημιουργηθούν στη V2 και η τιμή της που δίνει τους βέλτιστους χρόνους αλλάζει ανάλογα με την κάρτα γραφικών (πιο δυνατές κάρτες απαιτούν μεγαλύτερο B για βέλτιστες επιδόσεις). Τοπικά η τιμή που έδωσε τα καλύτερα αποτελέσματα ήταν B=2, ενώ στην Tesla P100 της συστοιχίας χρειάστηκε να οριστεί B>=5, ενώ να σημειωθεί ότι για τη βέλτιστη εκμετάλλευση των επιδόσεων της GPU πρέπει να οριστεί και το αντίστοιχο compilation flag στο Compute Capability που διαθέτει η κάρτα (τοπικά χρησιμοποιήθηκε -arch=sm_50 λόγω CC 5.0, ενώ στη συστοιχία -arch=sm_60 λόγω CC 6.0).

*(παράδειγμα χρήσης της libpng βρίσκεται εδώ)

vtmoscho@ece.auth.gr