Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών
Α.Π.Θ.

29/1/2017

Παράλληλα και Κατανεμημένα Συστήματα Υπολογιστών

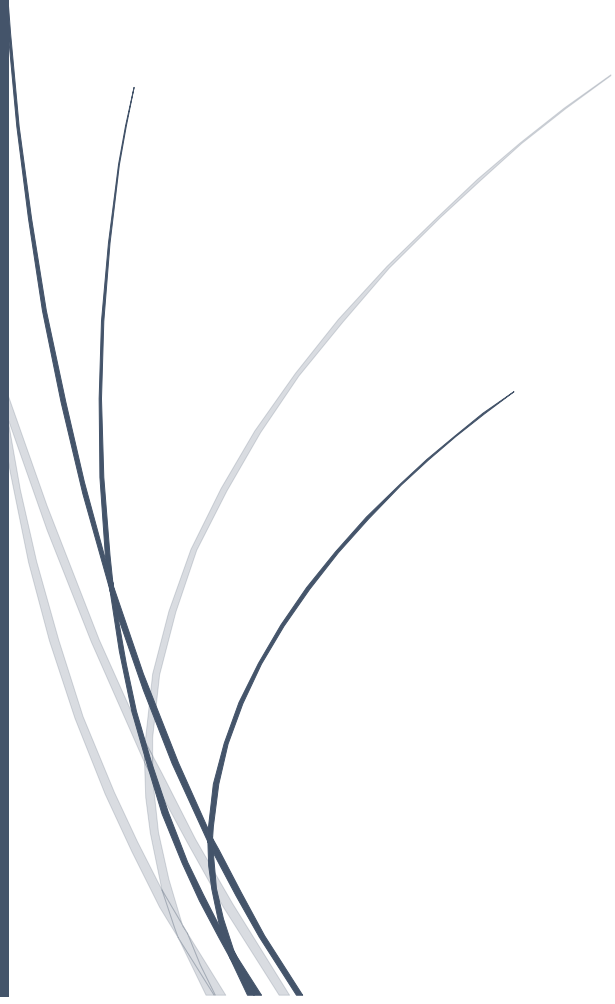
Εργασία 3

Διαμαντή Μαρία 8133

mfdiamanti@ece.auth.gr

Ντζιώνη Δήμητρα 8209

dntzioni@ece.auth.gr



Πίνακας περιεχομένων

1. Εισαγωγή

Η τρίτη εργασία στο μάθημα των Παράλληλων και Κατανεμημένων Συστημάτων Υπολογιστών αφορά τη συγγραφή ενός CUDA kernel, το οποίο υλοποιεί το γνωστό αλγόριθμο αποθορυβοποίησης εικόνας Non Local Means. Το CUDA kernel αυτό, καλείται μέσα από δοσμένο κώδικα γραμμένο σε MATLAB, στον οποίον μάλιστα γίνεται και η αρχική εισαγωγή των δεδομένων, όπως παραδείγματος χάριν της εικόνας. Επιπλέον, δίνεται έτοιμη και η υλοποίηση του αλγορίθμου Non Local Means γραμμένη σε MATLAB. Συνεπώς, καλούμαστε να συγκρίνουμε τη δική μας υλοποίηση με τη δοσμένη ως προς την ποιότητα αποθορυβοποίησης, αλλά και ως προς το χρόνο εκτέλεσης, καθώς ο δοσμένος αλγόριθμος Non Local Means σε MATLAB δεν κάνει χρήση των δυνατοτήτων της GPU. Ο αλγόριθμος βασίζεται στην εύρεση παρόμοιων γειτονιών σε όλη την εικόνα και στον υπολογισμό της αποθορυβοποιημένης τιμής καθενός pixel αυτής. Η εργασία αυτή έχει υλοποιηθεί και εκτελεσθεί για μεγέθη εικόνων 64 * 64, 128 * 128, 256 * 256 και μεγέθη γειτονιών 3 * 3, 5 * 5, 7 * 7.

2. Κώδικας MATLAB

Ξεκινώντας από την ανάλυση του κώδικα σε MATLAB, αυτός δέχεται μια εικόνα ως είσοδο, το μέγεθος της γειτονιάς των pixels που επιθυμούμε και κάποιες ακόμη παραμέτρους απαραίτητες για την υλοποίηση του αλγορίθμου Non Local Means, όπως το filtSigma και patchSigma. Η εισαγόμενη εικόνα αρχικά δέχεται normalizing και έπειτα της εφαρμόζεται ένας θόρυβος, τον οποίο καλούμαστε φυσικά να αποθορυβοποιήσουμε. Για να μπορούν τα ακριανά pixels της εικόνας να ανήκουν σε μια γειτονιά και να μπορέσουν να αποθορυβοποιηθούν ορθά, γίνεται padding και προστίθενται σε όλες τις πλευρές της εικόνας τόσοι γείτονες όσοι απαιτούνται, δηλαδή (patchSize-1)/2. Οι προστιθέμενοι αυτοί γείτονες παίρνουν ως τιμές αυτές των συμμετρικών τους pixels ως προς το ακριανό pixel στο οποίο είναι γείτονες. Επιπλέον, επειδή ζητείται η εικόνα να λαμβάνει τιμές float προτού περαστεί στο CUDA kernel, βασική προϋπόθεση είναι η μετατροπή του περιεχομένου της από double σε float μέσω της συνάρτησης single() του MATLAB. Μια τελευταία αναγκαιότητα προτού προχωρήσουμε στην αποθορυβοποίηση της εικόνας είναι η δημιουργία ενός φίλτρου Gauss, το οποίο θα χρησιμοποιηθεί στον αλγόριθμο αυτό. Έπειτα από την αποθορυβοποίηση της εικόνας, προχωρούμε στην περικοπή της ώστε να επανέλθει στο αρχικό της μέγεθος μέσω της συνάρτησης imcrop(). Τέλος, μέσω της συνάρτησης psnr() βρίσκουμε το λόγο ομοιότητας της φιλτραρισμένης εικόνας με την αρχική σε dB.

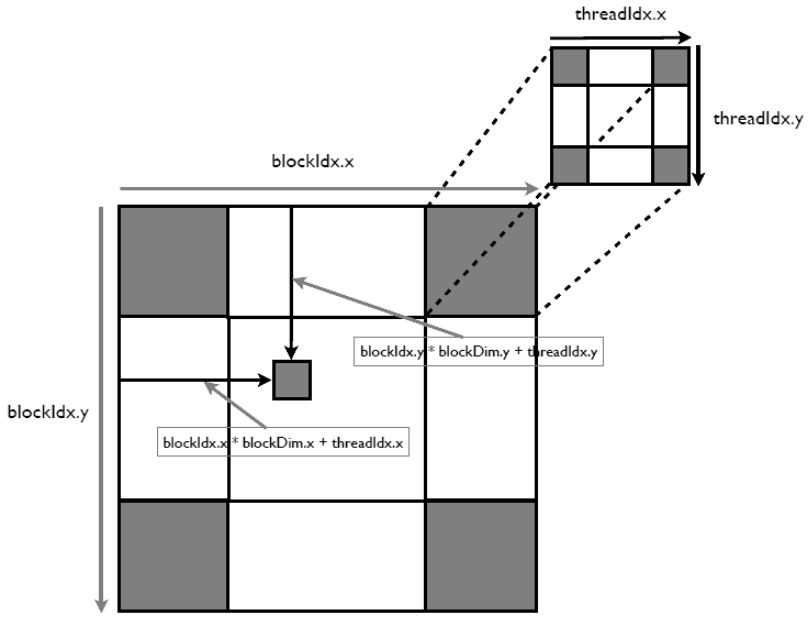
3. Μέθοδος παραλληλοποίησης με CUDA

Προκειμένου να γίνει χρήση της GPU και να καλέσουμε το CUDA kernel μας μέσα από το MATLAB, χρησιμοποιούμε τη συνάρτηση parallel.gpu.CUDAKernel(). Η συνάρτηση αυτή δημιουργεί ένα GPU CUDA kernel αντικείμενο με βάση τα .ptx και .cu αρχεία που του εισάγουμε. Αυτό το αντικείμενο μπορεί να χρησιμοποιηθεί για να καλεσθεί το CUDA kernel μας στη GPU. Επιπλέον, μέσω αυτού του αντικειμένου γίνεται προσπέλαση των μεταβλητών ThreadBlockSize και GridSize και ορίζεται ο αριθμός των νημάτων από τα οποία θα αποτελείται ένα block και το πλήθος των blocks που θα απαρτίζουν το grid. Με τη συνάρτηση feval() περνάμε στο CUDA kernel μας τα ορίσματα που απαιτούνται και μέσω της gather() λαμβάνουμε τα αποτελέσματα από την εκτέλεση του CUDA kernel πίσω στο MATLAB. Αξίζει να σημειωθεί πως για να περαστούν ως ορίσματα οι πίνακες, γίνεται χρήση της συνάρτησης gpuArray() προκειμένου να αντιγραφούν στην GPU. Το αποτέλεσμα που επιστρέφεται από αυτή τη συνάρτηση είναι ένα gpuArray αντικείμενο που περνιέται ως όρισμα στην feval() και συνεπώς στο CUDA kernel μας.

Έχοντας περάσει μέσα στο CUDA kernel, γίνεται χρήση του προθέματος __global__ που υποδηλώνει ότι πρόκειται για μια συνάρτηση που εκτελείται στην GPU και καλείται από την CPU. Για την υλοποίηση του αλγορίθμου και την αποθορυβοποίηση της εισαγόμενης εικόνας, επιλέξαμε να αναθέσουμε σε κάθε νήμα ένα pixel με συντεταγμένες (pi,pj). Γνωρίζοντας πως οι μεταβλητές blockIdx.y, blockIdx.x, threadIdx.y, threadIdx.x χρησιμοποιούνται όπως παρακάτω, και πως οι μεταβλητές blockDim.y και blockDim.x επιστρέφουν τον αριθμό των νημάτων ανά block, η δεικτοδότηση του pixel (pi,pj) που πρόκειται να αποθορυβοποιηθεί γίνεται ως εξής:

```
int pi = blockIdx.y * blockDim.y + threadIdx.y;

int pj = blockIdx.x * blockDim.x + threadIdx.x;
```



Τέλος, αξιοποιήθηκε η shared memory της GPU προκειμένου να μειωθούν οι αναγνώσεις από την global memory και να επιταχυνθεί περαιτέρω η υλοποίηση. Επιλέχθηκε, λοιπόν, στην shared memory να περαστεί ο πίνακας Gauss και να δημιουργηθεί επίσης, ένας τρισδιάστατος πίνακας στον οποίο κάθε threadIdx.x και threadIdx.y περνά τους γείτονές του, ανάλογα με το μέγεθος της γειτονιάς του, έτσι ώστε να μη χρειάζεται κάθε φορά να τους αναζητά στην global memory για να τους αναγνώσει. Οι πίνακες αυτοί αρχικοποιήθηκαν ως εξής με το πρόθεμα __shared__ :

__shared__ float sharedGauss[49];

__shared__ float sharedJ[8][8][49];

Η διάσταση του sharedGauss επιλέχθηκε για να ικανοποιείται η περίπτωση όπου το μέγεθος της γειτονιάς είναι μέγιστο, δηλαδή 7*7. Παρομοίως, οι διαστάσεις του πίνακα shared επιλέχθηκαν ώστε να ικανοποιούν την απαίτηση για 8 νήματα σε κάθε block και για μέγιστο μέγεθος γειτονιάς 7*7.

Στο σημείο αυτό αξίζει να αναφέρουμε πως η εργασία αυτή υλοποιείται μόνο για 8 νήματα σε κάθε block. Ο λόγος για τον οποίο προχωρήσαμε σε αυτήν την υλοποίηση είναι αρχικά το γεγονός πως ανατέθηκε ένα pixel σε ένα νήμα. Συγκεκριμένα, αυτό συνεπάγεται ότι όσο αυξάνουμε το πλήθος των νημάτων για ένα block, τόσο θα μεγαλώνει και η απαίτηση για δέσμευση περισσότερης shared memory ανά block. Ένας ακόμη λόγος είναι πως δοκιμάζοντας αρχικά να υλοποιήσουμε τη συνάρτηση με περισσότερα νήματα, δεν οδηγηθήκαμε σε επιτάχυνση της όλης διαδικασίας.

4. Αποτελέσματα και σχόλια

Στο στάδιο αυτό της σύγκρισης της υλοποίησης μας με τη δοθείσα, προχωρήσαμε σε τροποποιήσεις του δοσμένου κώδικα, έτσι ώστε να προσμετράται ο χρόνος όσο το δυνατόν πιο ισοδύναμου κώδικα. Πιο συγκεκριμένα, παρατηρήσαμε ότι το φίλτρο Gauss στη δοθείσα υλοποίηση δημιουργούταν στη συνάρτηση Non local means, ενώ στη δική μας περίπτωση στην Pipeline non local means. Για το λόγο αυτό, μετακινήθηκε το συγκεκριμένο κομμάτι κώδικα στην Pipeline non local means. Αξιοσημείωτο είναι πως οι μετρήσεις πραγματοποιήθηκαν στο παραχωρημένο server του πανεπιστημίου, «διάδη», στις 29/1/17 και ώρα 19:00. Παρακάτω παρουσιάζονται τα αποτελέσματα σύγκρισης τόσο των χρόνων εκτέλεσης, όσο και των λόγων ομοιότητας.

64*64	filtSigma 0.02, patchSigma 1.66, threadsPerBlock 8*8		
	PatchSize		
Time in: (sec)	3x3	5x5	7x7
CPU	0.858238	0.870134	0.935030
GPU	0.038967	0.110643	0.114855
PSNR in: (dB)			
CPU	33.08980	33.16940	33.01733
GPU	29.73125	30.56482	30.39954

128*128	filtSigma 0.02, patchSigma 1.66, threadsPerBlock 8*8		
	PatchSize		
Time in: (sec)	3x3	5x5	7x7
CPU	16.725978	25.783544	38.989052
GPU	0.493853	0.922760	1.533823
PSNR in: (dB)			
CPU	35.48410	36.12529	36.16575
GPU	31.52149	35.08195	36.20786

256*256	filtSigma 0.02, patchSigma 1.66, threadsPerBlock 8*8		
	PatchSize		
Time in: (sec)	3x3	5x5	7x7
CPU	-	-	-
GPU	6.908576	12.883628	21.588617
PSNR in: (dB)			
CPU	-	-	-
GPU	32.07518	32.02780	31.64467

Οι χρόνοι αυτοί αναφέρονται για μέγεθος 64*64 στο αρχείο house.mat, που δόθηκε, για 128*128 στο αρχείο poke.mat και για 256*256 στο αρχείο classic.mat, τα οποία δημιουργήθηκαν από εικόνες που εισήγαμε εμείς. Τα αποτελέσματα τόσο της αποθορυβοποιημένης εικόνας, όσο και του θορύβου που απομένει μετά τη διαδικασία, συμπεριλαμβάνονται στο results στο παραδοτέο αρχείο, για κάθε μια από τις παραπάνω περιπτώσεις.

Καταληκτικά, πρέπει να αναφέρουμε ότι παρόλο που οι χρόνοι εκτέλεσης είναι σημαντικά βελτιωμένοι με τη χρήση της μεθόδου CUDA, ωστόσο, η υλοποίηση μας δεν παρουσιάζει εξίσου μεγάλους λόγους ομοιότητας συγκριτικά με το δοσμένο κώδικα. Σε αντίστοιχα αποτελέσματα θα καταλήγαμε κι αν θέταμε ως είσοδο μια εικόνα που δημιουργούσαμε εμείς με τιμές 0 και 1. Ο τρόπος για να το κάνουμε αυτό είναι να χρησιμοποιήσουμε την εντολή randi([0 1], m,n) στο matlab, θέτοντας ως m και n τις διαστάσεις της εικόνας που επιθυμούμε να υλοποιήσουμε.

Η διαδικασία μετατροπής μια εικόνας σε αρχείο .mat είναι παραδείγματος χάρη για την εικόνα poke η εξής:

```
poke= imread(pokeball.jpg');

poke = rgb2gray(poke);

poke=im2double(poke);

save(poke.mat', poke);
```

5. Αναφορές

Παρακάτω αναγράφονται ενδεικτικά ορισμένοι ιστότοποι από όπου συλλέχθηκαν στοιχεία για τη διεκπεραίωση της εργασίας αυτής:

https://docs.nvidia.com/cuda/cuda-c-programming-guide/#introduction

https://devblogs.nvidia.com/parallelforall/cuda-pro-tip-write-flexible-kernels-grid-stride-loops/

http://stackoverflow.com/questions/15240432/does-syncthreads-synchronize-all-threads-in-the-grid

http://www.sciencedirect.com/science/article/pii/S1877050916306585

https://docs.nvidia.com/cuda/cuda-c-programming-guide/#introduction

https://www.mathworks.com/products/matlab.html