CUDA project

Fotiou Dimitrios
AEM 9650
Department of Electrical and Computer Engineering, AUTH
email: fdimitri@ece.auth.gr
github: https://github.com/dimfot3/CUDA_proj

***Abstract*—**Το παρόν έγγραφο αποτελεί την εργασία του μαθήματος Παράλληλα και Διανεμημένα Συστήματα του τμήματος Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών**.** Επίκεντρο αυτής της εργασίας είναι η παραλληλοποίηση μέσω της κάρτας γραφικών και συγκεκριμένα μέσω των **Nvidia** καρτών γραφικών με τη χρήση της **CUDA.** Για το σκοπό αυτό**,** καλούμαστε να προσομοιώσουμε και να συγκρίνουμε διαφορετικές παραλληλοποιήσεις του **Ising** μοντέλου**.** Όλοι οι κώδικες που χρησιμοποιήθηκαν για αυτό το **project** βρίσκονται στο **[5].**

***Index Terms*—Ising model, Nvidia, CUDA, parallelism**

I. Introduction

Το Ising model [1] προτάθηκε από του φυσικούς Ernst Ising και Wilhelm Lenz το 1924. Είναι ένα μοντέλο που μπορεί να δείξει τις μεταβολές των φάσεων των μαγνητικών διπόλων και την μεταβολή του συστήματος από υψηλότερες και χαμηλότερες ενεργειακές στάθμες.

Η πιο συνηθισμένη μορφή του Ising μοντέλου είναι σε δισδιάστατα πλέγματα όπου κάθε σημείο του αποτελεί ένα μαγνητικό δίπολο που παίρνει τιμές $\{-1, 1\}$. Κάθε δίπολο επηρεάζεται από τα 4 γειτονικά του και μεταβάλει την φάση του, έτσι ώστε να οδηγηθούμε σε μικρότερη στάθμη ενέργειας. Τη μικρότερη δυνατή ενέργεια έχουμε όταν τα δίπολα συμφωνούν μεταξύ τους.

Για αυτή την εργασία θα πρέπει να προσομοιώσουμε το Ising model. Συγκεκριμένα το δισδιάστατο πλέγμα έχει τετραγωνικό σχήμα και η νέα φάση του (i,j) διπόλου μετά την μεταβολή στο δισδιάστατο πίνακα δίνεται από τον τύπο

$$sign(G(i,j)+G(i+1,j)+G(i,j+1)+G(i-1,j)+G(i,j-1))$$

Γενικά το Ising μοντέλο δεν περιέχει κάποιο περίπλοκο ή δαπανηρό υπολογισμό, αλλά περισσότερο αποτελεί μια επαναληπτική διαδικασία που ενδείκνυται για παραλληλοποίηση. Για τον σκοπό αυτό καλούμαστε να παραλληλοποιήσουμε το παραπάνω σύστημα σε Nvidia κάρτες γραφικών με τη προγραμματιστική διεπαφή CUDA [2].

Οι κάρτες γραφικών αρχικά δημιουργήθηκαν για την υποστήριξη γραφικού περιβάλοντος και στα μετέπειτα χρόνια αξιοποιήθηκαν για ένα ευρύτερο φάσμα εφαρμογών. Συγκρίνοντας ένα μεμονωμένο πυρήνα της gpu και της cpu, της τελευταίας είναι γρηγορότερο και έχει μεγαλύτερο σύνολο εντολών και μνήμη cache. Παρόλα αυτά η δύναμη της gpu στειρίζεται στο συνολικό αριθμό των πυρήνων που είναι αρκετές τάξεις μεγαλύτερος από αυτόν της cpu. Η gpu είναι αρκετά ισχυρή όταν η εργασία αποτελείται από τόσες πολλές επαναλαμβανομενες ρουτίνες και υπολογισμούς έτσι ώστε η καθυστέρηση από τη μεταφορά δεδομένων και τη χαμηλότερη συχνότητα λειτουργίας της gpu να μην παίζουν ουσιαστικό ρόλο.

Όσο αναφορά την CUDA, όπως αναφέρθηκε αποτελεί μια προγραμματιστική διεπαφή ώστε να μπορούμε να έχουμε πρόσβαση στη μνήμη και στους πυρήνες της NVIDIA gpu. Σε αντίθεση με άλλες διεπαφές όπως την OpenGL, δεν απαιτεί ειδικές γνώσεις σε προγραμματισμό γραφικών και έτσι μπορεί να χρησιμοποιηθεί ευκολότερα σε προβλήματα γενικού σκοπού που δεν σχετίζονται με δημιουργία γραφικού περιβάλλοντος.

Παρακάτω παρουσιάζεται αρχικά η υλοποίηση της σειριακής μορφής του Ising model. Στη συνέχεια παρουσιάζονται διάφορες παραλληλοποιήσεις με CUDA και στο τέλος συγκρίνονται όλες μέθοδοι για να εξετάσουμε ποιος είναι ο αποτελεσματικότερος τρόπος διεκπεραίωσης ενός Ising model και πόσο σημαντική είναι η συμβολή των καρτών γραφικών στην επίλυση προβλημάτων παρόμοιας φύσης.

II. Implementation

Ο κώδικας γράφτηκε σε C και σαν compiling tool χρησιμοποιήθηκε CMake. Κύριο API για την διεκπεραίωση της εργασίας ήταν CUDA και σαν compiler το nvcc. Η εργασία αναπτύχθηκε αρχικά τοπικά σε Ubuntu 21.04, με CUDA 11.4 , AMD Ryzen 2600 series και κάρτα γραφικών NVIDIA 1650ti. Παρολα αυτα όλα τα αποτελέσματα που παρουσιάζονται παρακάτω έγιναν στη συστυχία του ΑΠΘ προκειμένου να συγκριθούν σε κοινή βάση με τους υπόλοιπους φοιτητές του μαθήματος.

A. Sequential

Η σειριακή υλοποίηση με εμφωλευμένες επαναλήψεις υπολογίζει σε κάθε σημείο την νέα τιμή του, σύμφωνα με τον τύπο που αναφέρθηκε παραπάνω. Επιπλέον αποθηκεύεται σε έναν προσωρινό πίνακα και μετά την μετάβαση όλων των στοιχείων οι δείκτες του αρχικού πίνακα και του προσωρινού ανταλλάζουν μεταξύ τους ώστε να επαναληφθεί η διαδικασία k φορές όπως αναφέρεται στην εκφώνηση.

B. Cuda V1

Στην πρώτη cuda υλοποίηση κάθε thread αναλαμβάνει να υπολογίσει ακριβώς ένα στοιχείο. Για τον σκοπό αυτό πρέπει να ρυθμίσουμε πόσα threads αναλογούν σε κάθε block. Ο μέγιστος αριθμός thread ανά block είναι 1024 και δεδομένου ότι θα χρησιμοποιήσουμε δυσδιάστατα block είναι $32x32$. Άρα έχοντας τη μέγιστη διάσταση των thread ανά block μπορούμε να δούμε πόσα thread χρειαζόμαστε, στογγυλλοποιόντας προς τα πάνω το πολίκο $n/32$, όπου n ο αριθμό της διάστασης του μοντέλου Ising.

Όσο αναφορά τον διαμοιρασμό των threads, αν γνωρίζαμε εξαρχής ότι ο αριθμός του πίνακα θα ήταν μικρός ώστε να απαιτούνταν λιγότερα από 8 blocks θα σύμφερε να σπάσουμε τα threads σε περισσότερα blocks καθώς κάθε 8 blocks τρέχουν σε διαφορετικό streaming multiprocessor το οποίο με τη σειρά του έχει όριο threads(τυπικά 2048). Παρόλα αυτά μιας και δεν μας δόθηκε κάποιο όριο και θέλουμε να λειτουργεί ακόμα και για μεγάλο αριθμό n, πχ 10000 που απαιτεί 313χ313 blocks πρακτικά δεν επηρεάζει η διασπορά των threads μιας και ο συνωστισμός θα γίνόταν μεταξύ των blocks που περιμένουν ελεύθερο streaming multiprocessor.

Επιπλέον στην cuda υλοποίηση πρέπει να αποφασίσουμε αν τα kernels θα ξεκινούν μια φορά κι όλες οι μεταβάσεις θα γίνονται με μια επανάληψη μέσα στο kernel. Κάτι τέτοιο σε περίπτωση που είχαμε λίγα στοιχεία και ένα μόνο block πιθανώς θα σύμφερε και ο συγχρονισμός των thread θα γινόταν εύκολα και άμεσα. Τώρα που θέλουμε να δουλεύει σωστά για μεγάλο αριθμό διάστασης πίνακα και δεδομένου ότι το overhead δημιουργίας των kernels είναι πολύ μικρό θα κρατήσουμε την επανάληψη των μεταβάσεων του Ising model εκτός των kernels.

Μια τεχνική βελτιστοποίησης αυτής της μεθόδου, χωρίς προφανώς να παραβιάσουμε τις απαιτήσεις της εκφώνησης, είναι ο τρόπος με τον οποίο γίνονται οι κλήσεις στην global memory ανά block. Συγκεκριμένα, αν καταφέρουμε τα thread που βρίσκονται στο ίδιο wrap (συλλογή threads που εκτελεί παράλληλα ο streaming multiprocessor) να ζητούν κομμάτια μνήμης που βρίσκονται στην σειρά τότε θα γίνει μία ενωμένη κλήση (Global Memory Coalescing) στην global memory και θα γλιτώσουμε πολύτιμο χρόνο και bandwidth. Για τον σκοπό αυτό στις κλήσεις λαμβάνεται υπόψη ότι τα threads είναι αποθηκευμένα κατά column major (το x μεταβάλλεται πιο γρήγορα από y και το y από z) όπως αναφέρεται στην [3] και επομένως οι κλήσεις στον πίνακα γίνονται με παρόμοια σύμβαση. Όπως βλέπουμε στο 1 στα αποτελέσματα οι δύο μέθοδοι έχουν αρκετά διαφορετικούς χρόνους εκτέλεσης για μεγάλο αριθμό διάστασης πίνακα όπου γίνονται πολλές κλήσεις στη μνήμη.

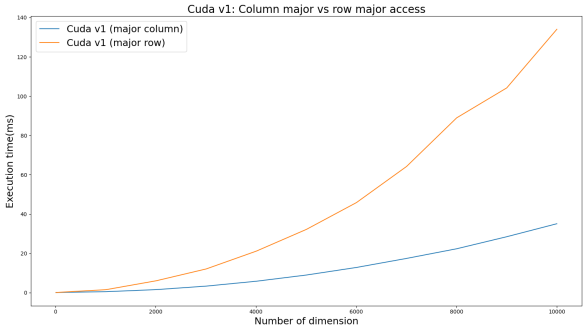

Figure 1. Cuda implementation v1: major column vs major row

C. Cuda V2

Για την δεύτερη υλοποίηση σε cuda ζητείται κάθε thread να αναλαμβάνει ένα block μεγέθους bxb για να υπολογίσει. Εδώ ο τρόπος που θα διαμειράσουμε τα threads είναι στρογγυλοποιώντας προς τα πάνω τον αριθμό της διαίρεσης n/b, όπου n το μέγεθος της διάστασης του πίνακα. Έτσι δημιουργούμε kxk συλλογές σημείων μεγέθους bxb η κάθε μία που τις μοιράζουμε σε κάθε thread ενός block όπως περιγράφηκε και πάνω. Όπως και στην παραπάνω περίπτωση εξετάζεται ο ρόλος της σύμβασης για τις κλήσεις στην global memory και επιλέγεται ο καλύτερος.

D. Cuda V3

Ως τελευταία περίπτωση έχουμε την υλοποίηση σε cuda ακριβώς όπως τη προηγούμενη υλοποίηση, απλά με την αξιοποίηση της shared memory. Η διαμοιραζόμενη μνήμη έχει πολύ μικρότερο χρόνο προσπέλασης από τη global και μοιράζεται μεταξύ threads στο ίδιο block. Εδώ πρέπει να επισημάνουμε ένα περιορισμό που μας τροποποιεί τον διαμοιρασμό των thread ανά block. Η μέγιστη τιμή που μπορεί να πάρει η κοινή μνήμη είναι 48kB (σε καινούργιες αρχιτεκτονικές μπορεί να αυξηθεί πχ μέχρι 96kB) επομένως ο αριθμός thread ανά block τροποποιείται ώστε να μην ξεφεύγει η μνήμη για δοσμένο b. Για τον σκοπό αυτό σε μια while οσο εντοπίζεται παραβίαση ορίων κοινής μνήμης μειώνεται ο μέγιστος αριθμός thread ανά block.

Εδώ να αναφερθεί ότι αυξάνεται σημαντικά η πολυπλοκότητα στον κώδικα του kernel. Σε περίπτωση που είχαμε να αντιμετωπίσουμε πίνακα με μικρό αριθμό διάστασης που χωρούσε σε ένα block, θα ήταν αρκετά εύκολο μιας και θα φορτώναμε όλο το πίνακα στην shared memory.

Υπάρχουν δύο σχεδιαστικές επιλογές. Η πρώτη είναι να φορτώσουν μόνο όσα στοιχεία ανήκουν σε ένα block στη shared memory που σημαίνει ότι όταν χρειάζονται στοιχεία γειτουνικού block θα πρέπει να γίνεται κλίση σε global. Το κακό με αυτή την περίπτωση είναι ότι υπάρχουν πολλοί έλεγχοι μέσα στο kernel και επομένως καθυστέρηση. Η δεύτερη επιλογή είναι να φορτώνουμε στη shared memory και τα όρια του block, με πλεονέκτημα ότι μειώνονται οι έλεγχοι αλλά αυξάνονται οι αχρείαστες κλήσεις.

Για εκπαιδευτικούς λόγους θα δοκιμάσουμε και τους δύο και θα καταλήγουμε στο καλύτερο. Στο σχήμα 2 βλέπουμε ότι δεν έχουν πολύ μεγάλη διαφορά οι δύο τρόποι.

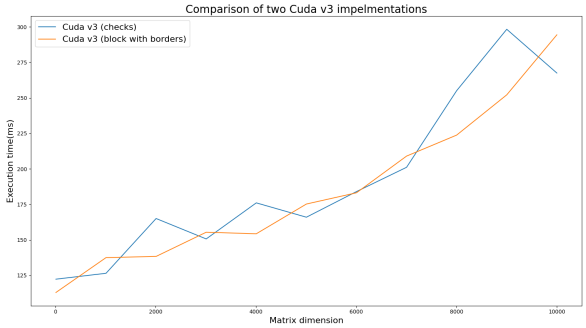

Figure 2. Cuda implementation v3: block with borders and without

E. Validation

Ένα επιπλέον ζητούμενο της εργασίας ήταν η τεκμηρίωση των αποτελεσμάτων. Πιο συγκεκριμένα ζητείται να βρούμε τρόπους απόδειξης ότι τόσο ο σειριακός κώδικας όσο και ο παράλληλος είναι σωστοί και το Ising model που παράγεται είναι επιθυμητό για δεδομένο k. Κάτι τέτοιο γενικά είναι σύνθετη διαδικασία μιας και το αρχικό μοντέλο είναι τυχαίο και η αναλυτική λύση ενός τέτοιου συστήματος για δεδομένο k είναι ιδιαίτερα σύνθετη [3]. Γενικά θα προσπαθήσουμε να ενισχύσουμε την εμπιστοσύνη στη σειριακή εκδοχή και οι παράλληλες εκδοχές θα συγκρίνονται με την σειριακή εκτέλεση.

Για να εμπιστευτούμε τα αποτελέσματά μας θα χρησιμοποιήσουμε μια βασική ιδιότητα του Ising model που λέει ότι με τη πάροδο των επαναλήψεων οδηγούμαστε σε καταστάσεις με μικρότερη ενέργεια και άρα περισσότερη συμφωνία μεταξύ γειτονικών διπόλων. Μαθηματικά θα λέγαμε ότι το άθροισμα των απόλυτων τιμών του αθροίσματος των γειτονιών

$$\sum_{i,j} |G(i,j)+G(i+1,j)+G(i,j+1)+G(i-1,j)+G(i,j-1)|$$

δεν θα πρέπει να μειώνεται σε κάθε επανάληψη. Φυσικά αυτό ισχύει στο ιδεατό σύστημα Ising όπου δεν έχουμε επίδραση θερμότητας που μπορεί να δημιουργήσει διαφορετικές καταστάσεις από τις προβλεπόμενες.

Αυτή η μέθοδος βέβαια μας δείχνει ότι ισχύει μια ιδιότητα του Ising μοντέλου αλλά δεν αποτελεί αναλυτική απόδειξη ή επιβεβαίωση. Για να βεβαιωθούμε επιπλέον ότι τόσο η βασικές συναρτήσεις όσο και οι υλοποιήσεις του Ising model είναι σωστές, δημιουργήθηκαν unit tests με Google tests που ελέγχουν την ορθότητα των συναρτήσεων με τεχνική μαύρου κουτιού παρέχοντας μια είσοδο και περιμένοντας μια συγκεκριμένη έξοδο.

III. Results

Στα αποτελέσματα θα εξετάσουμε κυρίως τους χρόνους εκτέλεσης για μεταβαλλόμενο αριθμό διάστασης πίνακα n. Ο αριθμός των μεταβάσεων k έχει αναμενόμενη γραμμική επίδραση, όπως βλέπουμε στο 3 σχήμα ενδεικτικά για τις cuda υλοποιήσεις.

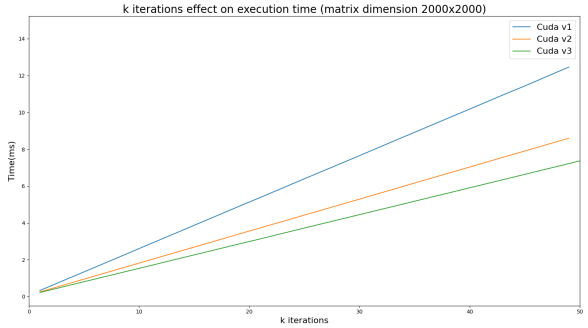

Figure 3. k iteratios effect on execution time

Ο μέγιστος αριθμός των διαστάσεων σε αυτή την εργασία είναι $10000\chi10000$ και είναι μια τιμή που έφτανε, τουλάχιστον στον τοπικό μου υπολογιστή, τη malloc στα όρια της. Στο σχήμα 4 βλέπουμε την σύγκριση όλων των υλοποιήσεων. Να σημειωθεί ότι στη σύγκριση αυτή οι cuda υλοποιήσεις περιλαμβάνουν και τον χρόνο μεταφοράς δεδομένων από host σε device.

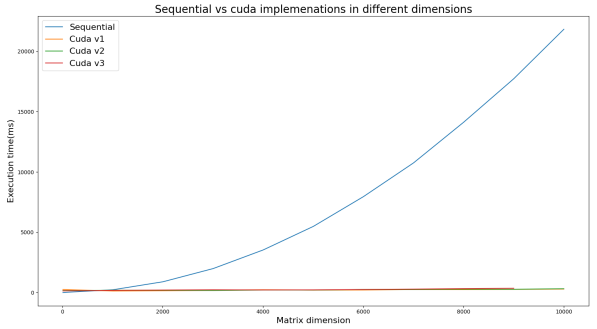

Figure 4. Serial vs Cuda implementations comparison (k=10)

Φαίνεται ξεκάθαρα η επικράτηση των cuda υλοποιήσεων και στο σχήμα 5 απεικονείζεται και η επιτάχυνση που πετυχαίνουν οι cuda υλοποιήσεις σε σύγκριση με τη σειριακή. Πριν εξετάσουμε καλύτερα τη σύγκριση μεταξύ cuda υλοποιήσεων ας δούμε την βελτιστοποίηση της v2 και v3 ως προς τον αριθμό του block. Βλέπουμε στο 6 ότι αν και αρχικά η v2 είναι καλύτερη αν θέλουμε για οποιοδήποτε λόγο να αυξήσουμε τον αριθμό του block τότε η υλοποίηση με shared μνήμη είναι πιο αποδοτική. Να σημειωθεί ότι στη βελτιστοποίηση δεν λαμβάνουμε υπόψη τη μεταφορά δεδομένων αλλά αποκλειστικά τους χρόνους εκτέλεσης των kernel.

Τέλος στο 7 έχουμε τη σύγκριση μεταξύ όλων των cuda υλοποιήσεων, και για τις v2 και v3 χρησιμοποιείται βέλτιστος αριθμός block. Οι v1 με την v2 υλοποίση φαίνεται να είναι αρκετά κοντά ενώ η v3 φαίνεται να είναι πιο κάτω από τις άλλες δύο. Αυτό μας φανερώνει ότι το overhead δημιουργείας kernel είναι

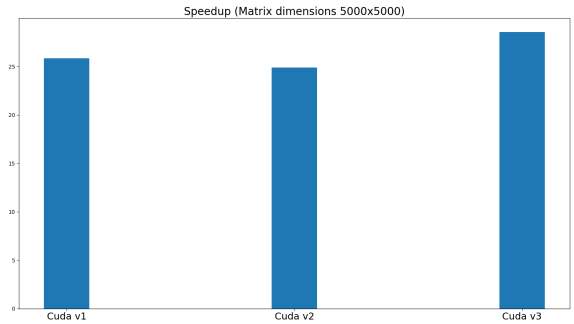

Figure 5. Speedup of Cuda implementations

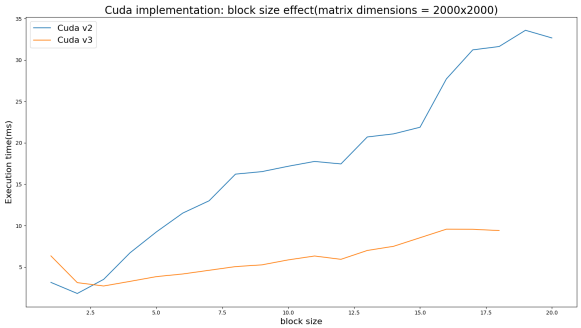

Figure 6. Block size search for v2 and v3 implementations

αρκετά μικρό αλλά και ότι η χρήση της shared memory μπορεί όντως να μειώσει τον συνολικό χρόνο εκτέλεσης.

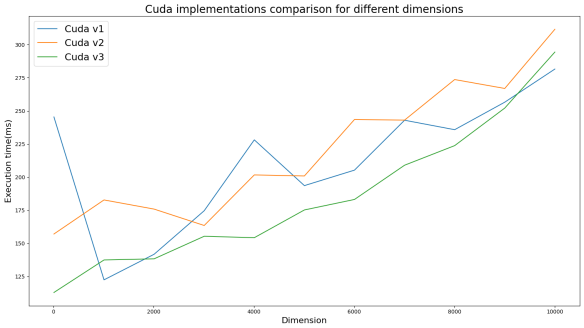

Figure 7. Comparison of Cuda implementations

IV. Conclusion

Κλείνοντας θα λέγαμε ότι η επικράτηση της παράλληλης υλοποίηση στη κάρτα γραφικών NVIDIA έναντι της σειριακής είναι δεδομένη σε εφαρμογές με επαναληπτικό χαρακτήρα όπως υπολογισμός Ising model. Από κει και πέρα, υπάρχουν πολλοί παράγοντες που επηρεάζουν το χρόνο εκτέλεσης και γενικά αν θέλουμε να δουλεύει η λύση μας σε διαφορετικά συστήματα

και για μεγάλη ποικιλία εισόδων πρέπει να κάνουμε κάποιους συμβιβασμούς.

Acknowledgment

Τα αποτελέσματα που παρουσιάζονται έχουν παραχθεί αξιοποιώντας την Υπολογιστική Συστοιχία και τις παρεχόμενες υπηρεσίες υποστήριξης του Κέντρου Ηλεκτρονικής Διακυβέρνησης του Α.Π.Θ..

References

[1] Ernst Ising “Ernst Ising, Contribution to the Theory of Ferromagnetism“ 1924

[2] Cuda API platform https://developer.nvidia.com/cuda-downloads

[3] Cuda programming guide https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html

[4] Kalz, A. and Honecker, Andreas and Moliner, Marion. (2011). Analysis of the phase transition for the Ising model on the frustrated square lattice. Physical Review B. 84. 174407. 10.1103/PhysRevB.84.174407.

[5] This project’s source code: https://github.com/dimfot3/CUDA_proj