Ελευθεριάδης Χαράλαμπος Παράλληλα και Διανεμημένα ΑΕΜ: 9257 | 6936981498
elefchar@ece.auth.gr

Ising Model – Serial and CUDA Implementation in C

Github Link: https://github.com/LambisElef/ising_cuda

Περιγραφή του προβλήματος:

Ζητούμενο της εργασίας ήταν η υλοποίηση του φερρομαγνητικού μοντέλου ising, αρχικά σειριακά και κατόπιν παράλληλα με χρήση GPU και CUDA. Μέσα στον κώδικα υπάρχουν εκτενή σχόλια που περιγράφουν τη λειτουργία του.

Σειριακή υλοποίηση (v0):

Η συνάρτηση spin(int *G, double *w, int *newG, int n) υπολογίζει από ένα τετραγωνικό πίνακα μαγνητικών δίπολων G διαστάσεων n*n ένα νέο πίνακα μαγνητικών δίπολων newG ιδίων διαστάσεων με βάση το μοντέλο ising 5x5 και τον σχετικό πίνακα βαρών w. Η υλοποίησή της είναι φτιαγμένη με τέτοιο τρόπο, ώστε να μπορέσει αργότερα να μετατραπεί εύκολα σε πυρήνα (kernel) για την υλοποίηση σε CUDA. Έτσι, οι έλεγχοι που απαιτούνται για τα ακραία σημεία του πίνακα μαγνητικών δίπολων σε περίπτωση αναδίπλωσης έχουν ενσωματωθεί στις πράξεις. Η επιλογή του κατάλληλου σημείου του G που θα επιδράσει με το ανάλογο βάρος που προσδιορίζεται από τον πίνακα βαρών w γίνεται με τον ακόλουθο τρόπο:

Αρχικά, υπολογίζουμε τη **σειρά** στην οποία βρίσκεται κάνοντας **ακέραια διαίρεση με το n** του δίπολου i για το οποίο γίνονται οι υπολογισμοί αυτής της επανάληψης και **αφαιρώντας ή προσθέτοντας τον κατάλληλο αριθμό σειρών** σύμφωνα με τις σχετικές συντεταγμένες του σημείου στον w. Κατόπιν, **προσθέτουμε n σειρές και κάνουμε modulo με το n** για την περίπτωση που βγούμε εκτός ορίων του πίνακα και **πολλαπλασιάζουμε με τον αριθμό των στοιχείων ανά σειρά n**, ώστε να επιστρέψουμε στο πρώτο σημείο της γραμμής. Τώρα βρισκόμαστε στην κατάλληλη γραμμή.

Προχωράμε, λοιπόν, στην εύρεση της κατάλληλης στήλης. Προσθέτουμε στον παραπάνω υπολογισμό το υπόλοιπο της διαίρεσης του δίπολου i με τον αριθμό των στηλών n αφού αφαιρέσουμε ή προσθέσουμε κατάλληλο αριθμό στηλών σύμφωνα με τις σχετικές συντεταγμένες του σημείου στον w και προσθέσουμε n για την περίπτωση που καταλήξουμε με αρνητικό αριθμό στηλών.

Η συνάρτηση check(int *G, int *newG, int n) ελέγχει τους δυο πίνακες G και newG για την περίπτωση που είναι ίδιοι και αν αυτό συμβεί, επιστρέφει 1 και σταματά η επαναληπτική εκτέλεση της spin. απαιτούνται επιπλέον επαναλήψεις.

Η συνάρτηση ising(int *G, double *w, int k, int n) έχει αναλάβει την όλη λογική του προγράμματος, αναλαμβάνοντας την επαναληπτική εκτέλεση της spin() για k επαναλήψεις, καθώς και της check() για την περίπτωση που δεν απαιτούνται περαιτέρω υπολογισμοί. Ακόμη, κάνει τις σχετικές μετρήσεις χρόνου εκτέλεσης της spin και βγάζει το μέσο χρόνο ανά επανάληψη.

Τέλος, η main() απλά περιλαμβάνει τη δημιουργία των πινάκων w και G. Ο G δημιουργείται είτε από τυχαία επιλογή 1 και -1 από ομοιόμορφη κατανομή, είτε από το αρχείο config-init.bin ώστε στη συνέχεια να γίνουν κατάλληλοι έλεγχοι ορθότητας και σφαλμάτων.

Παράλληλη (CUDA) υλοποίηση (v1):

Εδώ η συνάρτηση ising() έχει αναλάβει τον επιπλέον ρόλο της μεταφοράς των πινάκων w και G στη GPU, καθώς και της επιστροφής του τελικού πίνακα G μετά την εκτέλεση των k επαναλήψεων της spin() πίσω στην κύρια μνήμη του συστήματος. Οι συναρτήσεις spin() και check() έχουν γίνει πλέον πυρήνες (kernels) και για την κλήση τους δημιουργούνται threadsNum threads και (n*n+threadsNum-1)/threadsNum blocks για την κάλυψη όλου του πίνακα G, όπου κάθε thread υπολογίζει ένα νέο δίπολο.

Τα benchmarks χρόνου γίνονται με τη βοήθεια του nvidia profiler (nvprof).

Παράλληλη (CUDA) υλοποίηση (v2):

Παρόμοια με την προηγούμενη υλοποίηση, μόνο που εδώ το κάθε thread αναλαμβάνει τον υπολογισμό περισσότερων μαγνητικών δίπολων. Συγκεκριμένα, κάθε block με threadsNum threads αναλαμβάνει να υπολογίσει ένα τετράγωνο διάστασης threadsNum*threadsNum στον G. Το κάθε thread υπολογίζει μια στήλη δίπολων του block. Δηλαδή, ο αριθμός των threads ήταν κλειδωμένος, ανάλογα με τη διάσταση του block. Στην επόμενη υλοποίηση v3 αυτό παύει να ισχύει, διότι με ξεκλείδωτο αριθμό threads βλέπουμε καλύτερη απόδοση.

Παράλληλη (CUDA) υλοποίηση (v3):

Αυτή η υλοποίηση δουλεύει ακριβώς όπως η v2 με τη διαφορά του ξεκλείδωτου αριθμού threads threadsNum από τη διάσταση του block blockSize, καθώς και του ζητούμενου της εργασίας, τη χρήση της shared (on-chip) cache για caching του block για το οποίο γίνονται οι υπολογισμοί και τη μείωση των αναγνώσεων από την global GDDR5/GDDR5X/etc μνήμη. Τα δίπολα που πρέπει να αποθηκευτούν στην shared μνήμη χωρίζονται ισομερώς και το κάθε thread αναλαμβάνει να μεταφέρει G_length*G_length/threadsNum στοιχεία από τη global στη shared μνήμη, όπου G_length = blockSize+4, εξαιτίας των ακραίων σημείων που πρέπει να αποθηκευτούν για κάθε block όπως υπαγορεύει το μοντέλο 5x5. Κατόπιν, το κάθε thread αναλαμβάνει να υπολογίσει blockSize*blockSize/threadsNum δίπολα. Με αυτό τον τρόπο, το μέγεθος του block είναι ανεξάρτητο του αριθμού των threads.

Αποτελέσματα:

Για τα benchmarks χρησιμοποιήθηκε ένα σύστημα **με Ryzen 1600 @3.7GHz** και **NVidia GTX 650 1GB** με **nvidia-driver-440** και **cuda-version-10.1**. Η GPU είναι δυστυχώς αρκετά παλιά και σχετικά χαμηλών επιδόσεων. Ο χρόνος είναι σε ms και είναι ο μέσος χρόνος εκτέλεσης του kernel spin() για k = 10 επαναλήψεις. Επιλέχθηκε το βέλτιστο blockSize για τη v3 μετά από δοκιμαστικές μετρήσεις.

Σειριακή υλοποίηση (v0)

N	512	1024	2048	4096	8192	16384
Time	12.34	49.756	202.76	816.573	3319.04	13418.04

Παράλληλη (CUDA) υλοποίηση (v1)

ThreadsNum \ N	512	1024	2048	4096	8192
32	1.396	5.471	21.875	87.497	349.98
64	1.1214	4.4758	17.895	71.577	286.3
128	1.1112	4.4345	17.728	70.879	283.58
256	1.1112	4.4345	17.728	70.879	283.58
512	1.1112	4.4345	17.728	70.879	283.58

Παράλληλη (CUDA) υλοποίηση (v2)

ThreadsNum	N	512	1024	2048	4096	8192
32		1.6648	6.6482	26.68	106.55	425.97
64		1.2899	5.1257	20.299	80.373	319.99
128		1.3092	5.0881	20.218	80.031	319.03
256		1.67	5.0807	19.99	77.791	296.26
512		3.3372	5.0142	19.731	75.571	294.25
1024		N/A	9.961	19.863	78.835	304.44

Παράλληλη (CUDA) υλοποίηση (v3)

ThreadsNum	N	512	1024	2048	4096	8192	blockSize
32		1.1856	4.6859	18.682	74.669	298.64	16
64		0.9879	3.9198	15.645	62.539	250.13	16
128		0.9476	3.7463	14.93	59.682	238.66	32
256		0.9512	3.7802	15.091	60.343	241.36	32
512		0.9883	3.9571	15.828	62.244	253.24	32
1024		0.9706	3.9593	16.554	N/A	N/A	64

Μία βδομάδα νωρίτερα είχα στη διάθεσή μου ένα laptop με την **GTX 1650 4GB** με **nvidia-driver-435.21** και **cuda-version-10.1**, κάρτα σημαντικά νεότερη, υψηλότερων επιδόσεων και με πολύ γρηγορότερη global μνήμη. Παρακάτω παραθέτω τα σχετικά αποτελέσματα με σχεδόν ίδιο κώδικα με μόνη διαφορά στην τρίτη έκδοση (v3), όπου ο αριθμός των threads και το μέγεθος του block ήταν ίσα. Ο αριθμός των threads threadsNum εδώ είναι 64.

Παράλληλη (CUDA) υλοποίηση

N	V1	V2	V3
512	0.338	0.632	0.393
1024	1.34	2.85	1.49
2048	5.35	13.1	5.6
4096	21.4	54.1	21.8
16384	246	1230	252

Σε αυτή την κάρτα λόγω της πολύ γρηγορότερης global μνήμης, οι διαφορές μεταξύ v1 και v3 είναι ελάχιστες. Ίσως η v3 γινόταν λίγο γρηγορότερη εάν είχα κάνει ανεξάρτητο από τότε τον αριθμό των threads από το μέγεθος του block.