Παράλληλα Υπολογιστικά Συστήματα
Εργασία 2

Γεώργιος Παπαποστόλου
Α.Μ.: 7115132200005

Τα πειραματικά δεδομένα που υπάρχουν στις απαντήσεις της πρώτης και της δεύτερης άσκησης προέκυψαν από την εκτέλεση του κώδικα στο εργαστήριο linux του πανεπιστημίου (linux01 και linux10-linux24), ένω τα δεδομένα για την τρίτη και τέταρτη άσκηση προέκυψαν από τον προσωπικό μου υπολογιστή. Συγκεκριμένα στη περίπτωση της τρίτης άσκησης χρησιμοποίησα ένα Virtual machine, για να μπορέσω να χρησιμοποιήσω τη βιβλιοθήκη pthreads, η οποία υπάρχει αποκλειστικά στα linux.

Χαρακτηριστικά προσωπικού υπολογιστή:
•Λειτουργικό σύστημα: Windows 10, 64-bit / Virtual machine: Ubuntu (64-bit) 22.04.2 LTS
•Επεξεργαστής: AMD Ryzen 5 1600 3.20 GHz (6 core, 12 threads), το Virtual machine είχε διαθέσιμους 9 πυρήνες
•Μνήμη: 16GB, το Virtual machine είχε διαθέσιμα 8GB
•Κάρτα γραφικών: GTX 1050 Ti 4GB (Compute capability: 6.1, CUDA cores: 768)
•Έκδοση μεταγλωτιστή: CUDA: nvcc 12.1, Virtual machine: gcc 11.3.0

Επιπλέον όλα τα πειραματικά δεδομένα προέκυψαν ως ο μέσος όρος 4 εκτελέσεων του κάθε αρχείου, οι οποίες έγιναν με τη χρήση ένος script γραμμένου σε python (run_tests.py).

1 Άσκηση 1

Το ζητούμενο της 1ης άσκησης είναι η υλοποιήση της μεθόδου Monte Carlo με τη χρήση της βιβλιοθήκης MPI.

Για την υλοποίηση της άσκησης έγραψα δύο αρχεία σε *C*. Το αρχείο *monte_carlo_serial.c* περιέχει την ακολουθιακή υλοποίηση του αλγορίθμου και το αρχείο *monte_carlo_mpi.c* περιέχει την υλοποίηση με τη χρήση του MPI.

Κατά την εκτέλεση των αρχείων θα ζητηθεί από τον χρήστη να εισάγει τον αριθμό των ρίψεων που επιθυμεί να πραγματοποιηθούν.

	10^6	10^7	10^8
T_1	0.020	0.205	2.05

Πίνακας 1: Δεδομένα από την εκτέλεση του αρχείου *monte_carlo_serial.c*.

	2	4	8	16
	$iterations = 10^6$			
T_p	0.023	0.017	0.008	0.005
S_p	0.858	1.160	2.519	3.724
E_p	0.429	0.290	0.314	0.232
	$iterations = 10^7$			
T_p	0.111	0.068	0.045	0.030
S_p	1.843	2.987	4.509	6.828
E_p	0.921	0.746	0.563	0.426
	$iterations = 10^8$			
T_p	1.013	0.505	0.286	0.158
S_p	2.027	4.062	7.165	12.91
E_p	1.013	1.015	0.895	0.807

Πίνακας 2: Δεδομένα από την εκτέλεση του αρχείου *monte_carlo_mpi.c*.

Όπως φαίνεται από τα παραπάνω αποτελέσματα στη περίπτωση που έχουμε ρήψη 10^6 βελών, η απόδοση του παράλληλου αλγορίθμου είναι πολύ χαμηλή. Καθώς αυξάνεται ο αριθμός των ρήψεων αυξάνεται το φόρτο εργασίας του κάθε κόμβου με αποτέλεσμα στη περίπτωση με 10^8 ρήψεις να έχουμε σχεδόν γραμμική επιτάχυνση.

2 Άσκηση 2

Το ζητούμενο της 2ης άσκησης είναι η υλοποιήση του πολλαπλασιασμού μεταξύ ενός τετραγωνικού πίνακα και ενός διανύσματος με τη χρήση της βιβλιοθήκης MPI.

Για την υλοποίηση της άσκησης έγραψα δύο αρχεία σε C. Το αρχείο *mat_vec_serial.c* περιέχει την ακολουθιακή υλοποίηση του αλγορίθμου και το αρχείο *mat_vec_mpi.c* περιέχει την υλοποίηση με τη χρήση του MPI.

Κατά την εκτέλεση των αρχείων θα ζητηθεί από τον χρήστη να εισάγει τον αριθμό των γραμμών/στηλών που επιθυμεί να έχει ο πίνακας.

Για την παράλληλη υλοποιήση του αλγορίθμου χώρισα τον πίνακα σε υποπίνακες (μπλοκ-στήλες) με τη χρήση των εντολών *MPI_Type_vector* και *MPI_Type_create_resized*, για την δημιουργία τύπων για την αποστολή και λήψη των στοιχείων του πίνακα μεταξύ των κόμβων του συστήματος.

	5000	10000	20000
T_1	0.077	0.308	1.234

Πίνακας 3: Δεδομένα από την εκτέλεση του αρχείου *mat_vec_serial.c*.

	2	4	8	16
	$iterations = 5000$			
T_p	0.037	0.018	0.010	-
S_p	2.066	4.141	7.456	-
E_p	1.033	1.035	0.932	-
	$iterations = 10000$			
T_p	0.149	0.074	0.039	0.027
S_p	2.061	4.154	7.740	11.34
E_p	1.030	1.038	0.967	0.708
	$iterations = 20000$			
T_p	0.619	0.296	0.158	0.079
S_p	1.993	4.164	7.801	15.56
E_p	0.996	1.041	0.975	0.972

Πίνακας 4: Δεδομένα από την εκτέλεση του αρχείου *mat_vec_mpi.c*.

Η μέτρηση του χρόνου για την παράλληλη υλοποίηση αφορά μόνο το κομμάτι των υπολογισμών και τα αποτελέσματα είναι αυτά που περιμένουμε από τη θεωρία. Αν στον υπολογισμό του χρόνου συμπεριλάβουμε και τον χρόνο που απαιτείται για τη μεταφορά των δεδομένων καθώς και της δέσμευσης χώρου στους κόμβους τότε τα αποτελέσματα της παράλληλης υλοποίησης γίνονται χειρότερα από εκείνα της ακολουθιακής υλοποίησης.

3 Άσκηση 3

Το ζητούμενο της 3ης άσκησης είναι η υλοποιήση ενός προγράμματος στο οποίο ανανεώνεται μία κοινόχρηστη μεταβλητή με δύο διαφορετικούς τρόπου, *pthread* κλειδώματα και ατομικές εντολές, ώστε να αντιμετωπιστεί το *race condition* που προκύπτει.

Για την υλοποίηση της άσκησης έγραψα τρία αρχεία σε C. Το αρχείο *counter_sequential.c* περιέχει την ακολουθιακή υλοποίηση του αλγορίθμου και τα αρχεία *counter_pthreads_atomics.c* και *counter_pthreads_locks.c* περιέχουν τις υλοποιήσεις με ατομικές εντολές και κλειδώματα αντίστοιχα.

Στα αρχεία με τις παράλληλες υλοποιήσεις δοκίμασα δύο διαφορετικούς τρόπους για την παραλληλοποίησης (option = 0, option = 1). Στη πρώτη επιλογή χρησιμοποίησα κλείδωμα (ή ατομική εντολή) σε κάθε *iteration* της επανάληψης, ενώ στη

δεύτερη υπάρχει μία τοπική μεταβλητή και ανανεώνεται η κοινόχρηστη μεταβλητή μόνο μία φορά.

Η εκτέλεση των αρχείων γίνεται με την ακόλουθη εντολή: ./<file> <processors> <iterations> <option>.

	10^6	10^7	10^8
T_1	0.002	0.025	0.252

Πίνακας 5: Δεδομένα από την εκτέλεση του αρχείου *counter_sequential.c*.

threads	2	4	8
	iterations $= 10^6$		
T_p	0.007	0.001	0.002
S_p	0.349	1.486	0.873
E_p	0.174	0.371	0.109
	iterations $= 10^7$		
T_p	0.012	0.006	0.005
S_p	2.080	3.761	4.480
E_p	1.040	0.940	0.560
	iterations $= 10^8$		
T_p	0.122	0.062	0.048
S_p	2.056	4.064	5.209
E_p	1.028	1.016	0.651

Πίνακας 6: Δεδομένα από την εκτέλεση του αρχείου *counter_pthreads_locks.c*, option = 1.

threads	2	4	8
	$iterations = 10^6$		
T_p	0.001	0.0008	0.003
S_p	1.380	3.163	0.840
E_p	0.690	0.790	0.105
	$iterations = 10^7$		
T_p	0.012	0.06	0.005
S_p	2.151	3.771	4.533
E_p	1.075	0.944	0.566
	$iterations = 10^8$		
T_p	0.113	0.057	0.045
S_p	2.229	4.357	5.582
E_p	1.114	1.089	0.697

Πίνακας 7: Δεδομένα από την εκτέλεση του αρχείου *counter_pthreads_atomics.c*, option = 1.

Τα αποτελέσματα είναι τα αναεμενόμενα. Όσο αυξάνονται τα iterations τόσο αυξάνεται και η απόδοση της παραλληλοποίησης με πολλά threads. Παρατηρούμε ότι η απόδοση της υλοποίησης με ατομικές εντολές μας δίνει καλύτερα αποτελέσματα. Τα αποτελέσματα για option = 0 δεν έδειξαν κάποια βελτίωση. Παρατηρούμε, βέβαια, και πάλι ότι η υλοποίηση με τις ατομικές εντολές προσφέρει καλύτερη απόδοση. Χαρακτηριστικά είναι τα ακόλουθα αποτελέσματα.

threads	2	4	8
	counter_pthreads_locks.c		
T_p	4.408	5.160	5.700
S_p	0.057	0.048	0.044
E_p	0.028	0.012	0.005
	counter_pthreads_atomics.c		
T_p	1.298	1.645	1.749
S_p	0.194	0.153	0.144
E_p	0.097	0.0383	0.018

Πίνακας 8: $iterations = 10^8$, $option = 0$.

4 Άσκηση 4

Το ζητούμενο της 4ης άσκησης είναι η υλοποιήση ενός προγράμματος *CUDA* που θα υπολογίζει τον πολλαπλασιασμό πινάκων. Για την επίλυση της άσκησης έγραψα το αρχείο *mat_mat_mult_serial.c* που περιέχει την ακολουθιακή υλοποίηση του πολλαπλασιασμού πινάκων και τα αρχεία *CUDA mat_mat_mult.cu* και

mat_mat_mult_2.cu. Στο πρώτο αρχείο παραλληλοποιώ το πρόβλημα δίνοντας στον κάθε πυρήνα της κάρτας γραφικών μπλοκ-γραμμές του αριστερού πίνακα κατά τον πολλαπλασιασμό, ενώ στο δεύτερο παραλληλοποιώ το πρόβλημα αναθέτωντας τον υπολογισμό κάθε στοιχείου του πίνακα των αποτελεσμάτων σε έναν απο τους πυρήνες της κάρτας γραφικών.

Ο χρόνος που χρειάζεται ο ακολουθιακός αλγόριθμος για να πολλαπλασιάσει δύο πίνακες με 1024*1024 στοιχεία είναι 7.22 δευτερόλεπτα.

threads	128	256	512
T_p	3.452	2.043	2.017
S_p	2.092	3.536	3.581
E_p	0.016	0.013	0.006

Πίνακας 9: Δεδομένα από την εκτέλεση του αρχείου *mat_mat_mult.cu*.

threads/blocks	160/6554	6554/160	320/3277	3277/320	1024/1024
T_p	2.530	0.0002	1.921	0.0005	2.02
S_p	2.855	28899	3.759	14449	3.568
E_p	$2/10^6$	0.02	$3/10^6$	0.01	$3/10^6$

Πίνακας 10: Δεδομένα από την εκτέλεση του αρχείου *mat_mat_mult_2.cu*. Η πρώτη γραμμή είναι ο αριθμός των *thread* ανά *block*.

Όπως φαίνεται από τα παραπάνω δεδομένα η δεύτερη υλοποίηση πετυχαίνει συνολικά καλύτερη επιτάχυνση. Ιδιαίτερα στις περιπτώσεις που χρησιμοποιούμε πολλά *threads* ανά *block*, κατά την κλήση της *__global__* συνάρτησης, η επιτάχυνση του αλγορίθμου είναι πολύ μεγαλύτερη.