Παράλληλα Υπολογιστικά Συστήματα
Εργασία 1

Γεώργιος Παπαποστόλου
Α.Μ.: 7115132200005

Τα πειραματικά δεδομένα που υπάρχουν στις απαντήσεις των ασκήσεων προέκυψαν από την εκτέλεση του κώδικα στον υπολογιστή linux28, του εργαστηρίου linux του πανεπιστημίου. Παράλληλα με τα αποτελέσματα των εργαστηρίων θα αναφέρω και εκείνα που προέκυψαν από τον προσωπικό μου υπολογιστή, γιατί σε κάποιες περιπτώσεις μου φάνηκαν πιο συμβατά με τη θεωρία.

Χαρακτηριστικά προσωπικού υπολογιστή:
•Λειτουργικό σύστημα: 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: 16.0 GB, το Virtual machine είχε διαθέσιμα 8.0 GB
•Έκδοση μεταγλωτιστή: 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 εύρεση μίας προσέγγισης της τιμής του π με τη χρήση της μεθόδου Μόντε Κάρλο, καθώς και την σύγκριση του χρόνου εκτέλεσης μεταξύ της σειριακής υλοποίησης, της υλοποίησης με τη χρήση των *pthreads* και τη χρήση του *openmp*.

Για την λύση της άσκησης έγραψα τρία αρχεία. Το αρχείο er1_serial.c, για την σειριακή υλοποίηση της μεθόδου, το αρχείο er1_pthreads.c, για την υλοποίηση της μεθόδου με τη χρήση των *pthreads* και τέλος το αρχείο er1_openmp.c, για την υλοποίηση της μεθόδου με τη χρήση του *openmp*. Η κλήση των αρχείων γίνεται με τον ακόλουθο τρόπο: $./file\_name \quad number\_of\_shots \quad thread\_count$.

Στις δύο παράλληλες υλοποιήσεις χρειάστηκε να χρησιμοποιήσω τη συνάρτηση rand_r καθώς αποτελεί μία thread safe εναλλακτική της γεννήτριας τυχαίων αριθμών rand. Επιπλέον κάθε νήμα, για να παράξει έναν τυχαίο αριθμό, χρησιμοποιεί

ένα seed το οποίο εξαρτάται από το rank του ώστε να μην παράγονται σε όλα τα νήματα οι ίδιες ακολουθίες τυχαίων αριθμών. Επίσης δεν χρειάστηκε συγχρονισμός καθώς το κάθε νήμα εκτελούσε ρίψεις οι οποίες είναι ανεξάρτητες από τις ρίψεις των άλλων νημάτων.

Πειραματικά δεδομένα linux28.

T	T_p	S_p	E_p	T'_p	S'_p	E'_p
serial	1.969	-	-	-	-	-
2	1.013	1.944	0.972	1.006	1.956	0.978
4	0.532	3.700	0.925	0.529	3.722	0.930
8	0.538	3.657	0.457	0.534	3.683	0.460
16	0.530	3.710	0.231	0.528	3.727	0.232

Πίνακας 1: 10^8 ρίψεις, (T_p, S_p, E_p δεδομένα υλοποίησης με *pthreads*, T'_p, S'_p, E'_p δεδομένα υλοποίησης με *openmp*)

Πειραματικά δεδομένα από τον προσωπικό μου υπολογιστή.

T	T_p	S_p	E_p	T'_p	S'_p	E'_p
serial	2.434	-	-	-	-	-
2	1.241	1.960	0.980	1.228	1.982	0.991
4	0.642	3.787	0.946	0.669	3.639	0.909
8	0.398	6.104	0.763	0.401	6.059	0.757

Πίνακας 2: 10^8 ρίψεις, (T_p, S_p, E_p δεδομένα υλοποίησης με *pthreads*, T'_p, S'_p, E'_p δεδομένα υλοποίησης με *openmp*)

Από τα πειραματικά δεδομένα που προέκυψαν και στα δύο μηχανήματα δεν πειρατηρούνται ιδιαίτερες διαφορές μεταξύ της υλοποίησης με *pthreads* και *openmp*. Στα πειραματικά δεδομένα του εργαστηρίου για 2 και 4 νήματα η βελτίωση είναι σχεδόν γραμμική σε σχέση με τον σειριακή υλοποίηση, ενώ για 8 και 16 νήματα δεν υπάρχει κάποια αύξηση. Αντίθετα για τα πειραματικά δεδομένα του προσωπικού μου υπολογιστή αν και η βελτίωση για 8 νήματα δεν είναι γραμμική, παρατηρείται σχεδόν τριπλασιασμός της απόδοσης των νημάτων. Γενικά όλες οι βελτιώσεις που παρατηρούνται είναι λογικές καθώς πρόκειται για ένα καλά παραλληλοποιήσιμο πρόβλημα, με ανάγκη για έλαχιστη επικοινωνία μεταξύ των νημάτων.

2 Άσκηση 2

Το ζητούμενο της 2ης άσκησης είναι η αξιολόγηση του προβλήματος ψευδούς κοινοχρησίας κατά τον πολλαπλασιασμό πίνακα με δυάνυσμα (αρχείο pth_mat_vect_rand_split.c) και την αντιμετώπισή του με δύο διαφορετικές προσεγγίσεις.

T \ D	8000000*8	8000*8000	8*8000000
1	0.241	0.203	0.203
2	0.123	0.105	0.334
4	0.069	0.057	0.349
8	0.071	0.057	0.331

Πίνακας 3: Χρόνος εκτέλεσης ανά αριθμό νημάτων και διάσταση πίνακα για το αρχείο pth_mat_vect_rand_split.c

Ελέγχοντας για πρόβλημα ψευδούς κοινοχρησίας το παραπάνω αρχείο για διαστάσεις 8000000*8, 8000*8000 και 8*8000000 παρατηρούμε ότι στη τελευταία περίπτωση καθώς αυξάνονται τα νήμτα υπάρχει αυξηση του χρόνου εκτέλεσης, οπότε αυτή είναι και η περίπτωση στην οποία υπάχει το πρόβλημα ψευδούς κοινοχρησίας. Αυτό συμβαίνει γιατί καθώς πολλαπλασιάζουμε έναν πίνακα με λίγες γράμμες και πολλές στήλες με ένα μεγάλο διάνυσμα μοιράζουμε τις λίγες γραμμές στα νήματα τα οποία συνεχώς κάνουν πράξεις με τις στήλες του διανύσματος.

Για τη 1η προσέγγιση αντιμετώπισης του προβλήματος έγραψα το αρχείο er2_1.c (κλήση *./file_name m n thread_count*). Τροποποίησα την υπάρχουσα μέθοδο προσθέτοντας ένα lock γύρω από τους κόμβους επανάληψης. Η υλοποίηση της προσέγγισης lock-step για την αντιμετώπιση της ψευδούς κοινοχρησίας είχε σαν αποτέλεσμα ο χρόνος εκτέλεσης να παραμείνει σταθερός, ανεξάρτητα από τα πόσα νήματα χρησιμοποιούμε και τη διάσταση του πίνακα.

T \ D	8000000*8	8000*8000	8*8000000
1	0.239	0.203	0.203
2	0.238	0.203	0.205
4	0.240	0.203	0.204
8	0.240	0.206	0.206

Πίνακας 4: Χρόνος εκτέλεσης ανά αριθμό νημάτων και διάσταση πίνακα για το αρχείο er2_1.c

Για τη 2η προσέγγιση αντιμετώπισης του προβλήματος έγραψα το αρχείο er2_2.c (κλήση *./file_name thread_count m n*). Στο αρχείο αυτό τροποποίησα και πάλι την υπάρχουσα μέθοδο προσθέτωντας μία τοπική μεταβλητή για την προσωρινή αποθήκευση των υπολογισμών μέσα στις επαναλήψεις και εξωτερικά χρησιμοποιώντας ένα lock επιστρέφω αυτή την τοπική μεταβλητή. Όπως φαίνεται και από τα αποτελέσματα, παρακάτω, αυτή η προσέγγιση είχε τα καλύτερα απότελέσματα καθώς τα νήμτα λειτουργούσαν ανεξάρτητα μεταξύ τους, και επικοινωνούσαν μόνο για την πρόσθεση της τοπικής τους μεταβλητής.

T \ D	8000000*8	8000*8000	8*8000000
1	0.283	0.203	0.205
2	0.166	0.106	0.106
4	0.109	0.056	0.056
8	0.096	0.060	0.059

Πίνακας 5: Χρόνος εκτέλεσης ανά αριθμό νημάτων και διάσταση πίνακα για το αρχείο er2_2.c

3 Άσκηση 3

Το ζητούμενο της 3ης άσκησης είναι η τροποποίηση του αρχείου *omp_mat_vect_rand_split.c*, το οποίο πολλαπλασιάζει έναν άνω τριγωνικό πίνακα με ένα διάνυσμα, ώστε να παρατηρήσουμε τις διαφορές των επιλογών *schedule* που παρέχει η *openmp*.

T \ D	8000000*8	8000*8000	8*8000000
1	0.299	0.247	0.247
2	0.153	0.126	0.588
4	0.124	0.062	0.628
8	0.061	0.054	0.693

Πίνακας 6: Χρόνος εκτέλεσης ανά αριθμό νημάτων και διάσταση πίνακα για το αρχείο *omp_mat_vect_rand_split.c*.

T \ D	8000000*8	8000*8000	8*8000000	8000000*8	8000*8000	8*8000000
1	0.072	0.094	0.186	0.051	0.095	0.187
2	0.048	0.048	0.096	0.026	0.071	0.094
4	0.038	0.026	0.050	0.014	0.042	0.051
8	0.037	0.019	0.034	0.009	0.025	0.036

Πίνακας 7: Χρόνος εκτέλεσης ανά αριθμό νημάτων και διάσταση πίνακα για τα αρχεία *er3_static.c* (πρώτες 3 στήλες) και *er3_guided.c* (τελευταίες 3 στήλες).

Για την λύση της άσκησης έγραψα δύο αρχεία, *er3_static.c* και *er3_guided.c* (κλήση *./file_name thread_count m n*) τροποποιώντας τον δοθέντα κώδικα. Και στις δύο τροποποιήσεις, με static και guided schedule, τροποποίησα τις επαναλήψεις ώστε να γίνονται πράξεις μόνο με τα στοιχεία του πίνακα που βρίσκονται πάνω από την διαγώνιο. Παράλληλα τροποποίησα τον εσωτερικό κόμβο της επανάληψης, ώστε να υπολογίζεται τοπικά το αποτέλεσμα των πράξεων και στη συνέχεια να προστίθεται στην κοινή μεταβλητή.

Το guided schedule παρατηρώ ότι μειώνει τον χρόνο εκτέλεσης περισσότερο από το static schedule όταν ο πίνακας έχει πολύ περισσότερες γραμμές από ότι στήλες, ενώ διαφορετικά δεν παρατηρούνται μεγάλες διαφορές.

4 Άσκηση 4

Το ζητούμενο της 4ης άσκησης είναι η παραλληλοποίηση της προς τα πίσω αντικατάστασης σε ένα άνω τριγωνικό σύστημα γραμμικών εξισώσεων.

Για την λύση της άσκησης έγραψα το αρχείο *er4.c* (κλήση $./file\_name \quad thread\_count \quad n$). Για την υλοποίηση, αρχικά παρατηρούμε ότι η παραλληλοποίηση του εξωτερικού κόμβου επανάληψης δεν είναι δυνατή, καθώς οι επάνω γραμμές του γραμμικού συστήματος εξαρτώνται από τις από κάτω τους. Αντίθετα ο εσωτερικός κόμβος επανάληψης μπορεί να παραλληλοποιηθεί. Πιο συγκεκριμένα μπορούμε να χωρίσουμε τα στοιχεία κάθε γραμμής στα νήματα που διαθέτουμε, να πραγματοποιήσουμε τοπικούς υπολογισμούς και στη συνέχεια χρησιμοποιώντας συγχρονισμό να ενημερώσουμε την αρχική μεταβλητή.

T	T_p	S_p	E_p	T'_p	S'_p	E'_p
1	0.135	-	-	0.314	-	-
2	0.101	1.329	0.664	0.260	1.210	0.605
4	0.069	1.936	0.484	0.190	1.656	0.414
8	0.061	2.214	0.276	0.109	2.875	0.359

Πίνακας 8: Χρόνος εκτέλεσης, επιτάχυνση και απόδοση ανά αριθμό νημάτων για *linux28* (πρώτες 3 στήλες) και προσωπικό υπολογιστή (τελευταίες 3 στήλες). Διάσταση συστήματος: 8000*8000.

Αν και παρατηρείται βελτίωση αυξάνοντας τον αριθμό των νημάτων, αυτή δεν είναι βέλτιστη. Βασικός λόγος για αυτό πιστεύω ότι είναι ότι ο εξωτερικός κόμβος της επανάληψης δεν μπορεί να παραλληλοποιηθεί. Επίσης οι υπολογισμοί που πραγματοποιούνται στις κάτω γραμμές είναι λίγοι, με αποτέλεσμα όσο αυξάνεται ο αριθμός των νημάτων να μειώνεται η απόδοσή τους.