Παράλληλα συστήματα - Χειμερινό ‘22

Εργασία 1/8-12-2022

Ομάδα:

sdi2000064 Κωνσταντίνος Κανελλάκης

sdi2000212 Μιχάλης Χατζησπύρου

Άσκηση 1.1

Για τον υπολογισμό των βελών εντός του κύκλου αναπτύχθηκε μία συνάρτηση η οποία καλείται τόσο από τον σειριακό αλγόριθμο όσο και από την υλοποίηση με threads. Για τον υπολογισμό των τυχαίων μεταβλητών χρησιμοποιήσαμε τον δοσμένο κώδικα εξασφαλίζοντας ότι για κάθε thread το seed είναι διαφορετικό. Επίσης, κάναμε τις απαραίτητες μετατροπές για τον υπολογισμό τυχαίων αριθμών στο διάστημα (-1,1).

Για την σειριακή υλοποίηση το κύριο πρόγραμμα καλεί την συνάρτηση για τον υπολογισμό των βελών εντός του κύκλου και στη συνέχει κάνει τις απαραίτητες πράξεις για την εκτίμηση του π.

Για τον παράλληλο αλγόριθμο κάθε νήμα υπολογίζει με την βοήθεια της συνάρτησης υπολογισμού των βελών εντός του κύκλου ένα μέρος του συνολικού αθροίσματος το οποίο και προσθέτει στην global μεταβλητή arrows αν και μόνο αν κανένα άλλο νήμα δεν γράφει σε αυτή. Αφού όλα τα νήματα τερματίσουν η συνάρτηση main κάνει τους υπολογισμούς για το π.

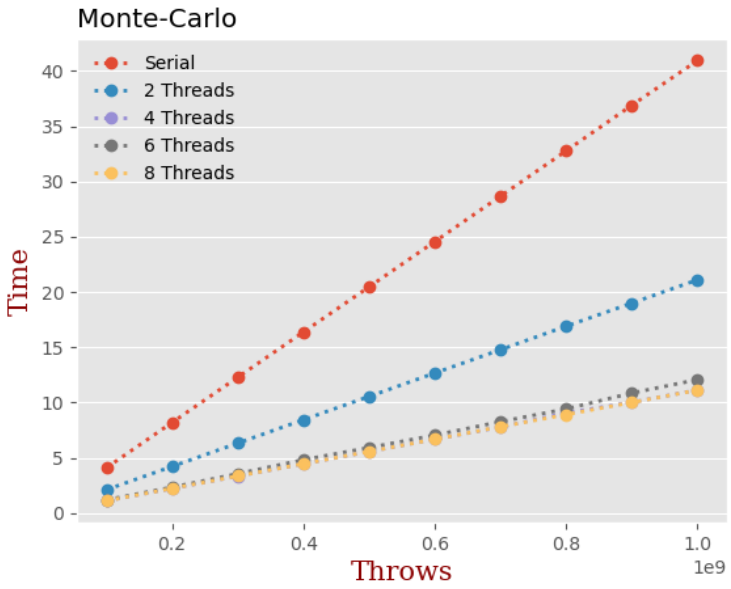
Για την εκτίμηση των αποτελεσμάτων των δύο αλγορίθμων δημιουργήθηκαν δύο επιπλέον αρχεία. Το script.sh παίρνει ως ορίσματα τον αριθμό των ρίψεων και τον αριθμό των νημάτων από τον οποίο θα ξεκινήσει. Για κάθε αριθμό ρίψεων εκτελείται το πρόγραμμα 4 φορές και υπολογίζει τον μέσο όρο εκτέλεσης του προγράμματος. Στην συνέχεια αυξάνεται ο αριθμός των ρίψεων κατά τον αριθμό των ρίψεων που δόθηκαν ως όρισμα. Αυτή η διαδικασία επαναλαμβάνεται για 10 φορές και στη συνέχεια αυξάνεται ο αριθμός των νημάτων ανά 2 και εκτελείται όλη η προηγούμενη διαδικασία. Αυτό εκτελείται για 4 διαφορετικές τιμές νημάτων. Στη συνέχεια το παραγόμενο αρχείο “averages.txt” χρησιμοποιείται από το plots.py για την δημιουργία γραφικής παράστασης και πίνακα με τα δεδομένα όπως φαίνεται παρακάτω:

Για αριθμό ρίψεων 10^8 – 10^9 και 2,4,6,8 νήματα έχουμε:

	1.00E+08	2.00E+08	3.00E+08	4.00E+08	5.00E+08	6.00E+08	7.00E+08	8.00E+08	9.00E+08	1.00E+09
Serial	3.141474	3.141467	3.141497	3.1415	3.141535	3.141536	3.141526	3.141568	3.141561	3.141587
2 Threads	3.141628	3.141648	3.141546	3.141601	3.141629	3.141627	3.141589	3.141574	3.14159	3.141577
4 Threads	3.141644	3.14154	3.141589	3.14155	3.141515	3.141527	3.141565	3.141597	3.14162	3.14159
6 Threads	3.141799	3.14165	3.141592	3.141598	3.141571	3.141582	3.141557	3.14154	3.141538	3.141575
8 Threads	3.141632	3.141539	3.141513	3.141559	3.141557	3.141625	3.141589	3.141549	3.141529	3.141501

	1.00E+08	2.00E+08	3.00E+08	4.00E+08	5.00E+08	6.00E+08	7.00E+08	8.00E+08	9.00E+08	1.00E+09
Serial	4.107479	8.200755	12.29332	16.3885	20.47286	24.57439	28.67477	32.78002	36.90504	40.95253
2 Threads	2.118032	4.224554	6.334786	8.447862	10.55922	12.67458	14.78629	16.91335	19.00412	21.11403
4 Threads	1.118747	2.228544	3.352638	4.463719	5.569781	6.687435	7.792895	9.068783	10.02748	11.14577
6 Threads	1.20245	2.345439	3.563628	4.829258	5.927413	7.05669	8.275618	9.426218	10.86919	12.07845
8 Threads	1.123645	2.23776	3.361655	4.466759	5.580244	6.689588	7.821064	8.920788	10.01363	11.13492

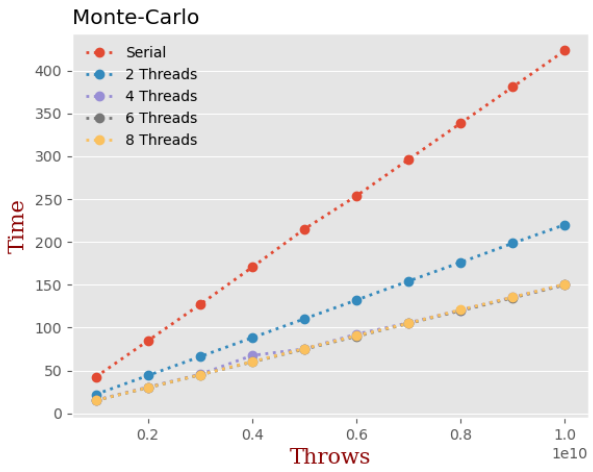
Όπως φαίνεται, από το διάγραμμα και τα δεδομένα του πίνακα, όσο αυξάνονται οι ρίψεις της συνάρτησης monte carlo τόσο αυξάνεται και ο χρόνος των υλοποιήσεων μας. Ωστόσο ο σειριακός αλγόριθμος αυξάνεται με πολύ μεγαλύτερο ρυθμό σε σχέση με τις πολυνηματικές υλοποιήσεις όπως και περιμέναμε. Επίσης όσο αυξάνεται ο αριθμός των νημάτων παρατηρούμε όλο και μικρότερη βελτίωση στον χρόνο. Όταν πια ο αριθμός των νημάτων ξεπερνά τα 4 είναι φανερό ότι η βελτίωση στον χρόνο είναι σχεδόν μηδαμινή. Για παράδειγμα η μωβ γραμμή είναι σχεδόν άφαντη στο διάγραμμα και αυτό γιατί έχει σχεδόν ίδιους χρόνους με τα 8 νήματα και την επικαλύπτει(η μωβ γραμμή φαίνεται αμυδρα στο 3ο κυκλάκι της κίτρινης γραμμής από τα αριστερά). Το γεγονός αυτό οφείλεται στο ότι ο επεξεργαστής μας είναι 4 πυρήνων και άρα η νημάτωση με περισσότερα των 4 νημάτων είναι περιττή. Τέλος, αξίζει να σημειωθεί ότι , σύμφωνα με τα δεδομένα του πίνακα, η αύξηση στον αριθμό των ρίψεων παρέχει καλύτερη εκτίμηση του π(ακρίβεια 7 σημαντικών ψηφίων).



Για αριθμό ρίψεων 10^9 – 10^{10} και 2,4,6,8 νήματα έχουμε:

	1.00E+09	2.00E+09	3.00E+09	4.00E+09	5.00E+09	6.00E+09	7.00E+09	8.00E+09	9.00E+09	1.00E+10
Serial	3.141587	3.141554	3.141575	3.141588	3.141584	3.141589	3.141588	3.141583	3.14159	3.141592
2 Threads	3.141577	3.141591	3.141599	3.141577	3.141584	3.141584	3.14157	3.141586	3.141594	3.141589
4 Threads	3.14159	3.14158	3.141579	3.141586	3.141599	3.141611	3.141602	3.141592	3.141585	3.141589
6 Threads	3.141575	3.141592	3.141567	3.141557	3.141562	3.141563	3.141583	3.141593	3.141608	3.141601
8 Threads	3.141501	3.141554	3.141563	3.141564	3.141559	3.141566	3.141579	3.141579	3.141594	3.1416

	1.00E+09	2.00E+09	3.00E+09	4.00E+09	5.00E+09	6.00E+09	7.00E+09	8.00E+09	9.00E+09	1.00E+10
Serial	42.42681	84.68907	126.9811	170.5107	215.0225	253.8154	296.2896	338.6576	381.0079	423.5081
2 Threads	22.09997	44.11731	66.17018	88.24409	110.2888	132.1818	154.3947	176.2132	198.5782	220.2625
4 Threads	15.30986	30.67401	45.48704	67.53722	75.22375	92.36156	105.6864	120.5918	135.62	150.3814
6 Threads	15.00382	29.95462	44.91625	59.94652	74.88803	89.83971	104.8782	119.8514	134.8058	149.9083
8 Threads	15.04583	30.1757	45.02473	60.33178	75.50426	90.7116	105.1342	120.9054	135.6009	150.8066



Σύμφωνα με τα δεδομένα για ρίψεις 10^9-10^{10} παρατηρούμε αρκετά μεγαλύτερη αύξηση όσο αναφορά τον χρόνο διεκπεραίωσης. Ωστόσο οι ρυθμοί αύξησης του χρόνου παραμένουν παρόμοιοι. Τέλος, παρατηρούμε ακόμα καλύτερες εκτιμήσεις για το π συγκρίνοντας και τον πίνακα για το εύρος τιμών 10^8-10^9.

Άσκηση 1.2

Για την υλοποίηση των προσεγγίσεων μας χρησιμοποιήσαμε το δοσμένο αρχείο pth_mat_vect_rand_split.c και το τροποποιήσαμε καταλλήλως προκειμένου να ελέγχονται οι είσοδοι για τέλεια διαίρεση μεταξύ m και αριθμού νημάτων.

Στο αρχείο pth_mat_vect_rand_split_first_approach.c υλοποιούμε την μέθοδο προσθέτοντας cache_line*thread_count στοιχεία στο y. Αυτό αποσκοπεί στην «προσθήκη» ενός ολόκληρου cache line (cache_line/sizeof(data type) θέσεις) στο τέλος κάθε μπλοκ στοιχείων που ενημερώνει το κάθε νήμα. Στην πραγματικότητα απλά αφήνουμε κενές αυτές τις θέσεις του πίνακα ώστε κανένα νήμα να μην τις χρησιμοποιήσει για να μη γίνει ψευδή κοινοχρησία.

Στο αρχείο pth_mat_vect_rand_split_second_approach.c υλοποιούμε την μέθοδο τοπικής αποθήκευσης αντικαθιστώντας την άμεση προσαύξηση της τιμής y[i] με έναν αθροιστή sum και του οποίου την τιμή αναθέτουμε στη συνέχεια στο y[i]. Αυτό αποσκοπεί στην μείωση της ψευδής κοινοχρησίας.

Για την παρουσίαση των αποτελεσμάτων χρησιμοποιήθηκαν 2 επιπλέον αρχεία script.sh και table.py. Το script.sh παίρνει ως όρισμα τον αριθμό των νημάτων από το οποίο θα ξεκινήσει να εκτελεί και για τις τρεις υλοποιήσεις (αρχική,προσέγγιση1 και προσέγγιση2) τα προγράμματα για τις διαστάσεις πινάκων που δίνονται στην εκφώνηση. Επίσης κάνει την ίδια διαδικασία για 2, 4, 8 νήματα και εξάγει τα αποτελέσματα σε αρχείο txt. Το table.py αντλεί τα αποτελέσματα από το αρχείο αυτό, τα αποθηκεύει σε έναν πίνακα 3 διαστάσεων και δημιουργεί τον παρακάτω πίνακα.

	8000000*8	8000*8000	8*8000000	8*80000000
Original				
1 Threads	0.234107	0.204744	0.205768	2.060585
2 Threads	0.11987	0.106658	0.340224	3.347007
4 Threads	0.09843	0.086126	0.311313	3.436154
8 Threads	0.087496	0.07729	0.332711	3.06824
Approach 1				
1 Threads	0.234834	0.205735	0.206258	2.064222
2 Threads	0.121195	0.106322	0.106354	1.061701
4 Threads	0.098784	0.088	0.08849	0.774377
8 Threads	0.08866	0.081355	0.084381	0.737731
Approach 2				
1 Threads	0.282768	0.205151	0.205462	2.055744
2 Threads	0.15923	0.106678	0.106224	1.062443
4 Threads	0.127028	0.085782	0.087197	0.768509
8 Threads	0.119295	0.089476	0.085688	0.74078

Τέλος, συμπεραίνουμε ότι για μικρών διαστάσεων πίνακα y και οι δύο υλοποιήσεις έχουν πανομοιότυπο χρόνο εκτέλεσης (για πολλά νήματα), σαφώς μικρότερο από την αρχική υλοποίηση, γεγονός που ήταν αναμενόμενο αφού η ψευδής κοινοχρησία εμφανίζεται πιο συχνά. Για μεγάλων διαστάσεων πίνακα y η ψεδυής κοινοχρησία εμφανίζεται πιο σπάνια γεγονός που επιβεβαιώνεται από τους πανομοιότυπους χρόνους εκτέλσης και των δύο προσεγγίσεων αλλά και της αρχική υλοποίησης ανεξάρτητα με το πλήθος των νημάτων. Με μια ματιά στον πίνακα είναι φανερό ότι η γρηγορότερη προσέγγιση είναι η πρώτη. Λογικό, εφόσον από την θεωρία η πρώτη προσέγγιση εξαλείφει το false sharing και άρα τρέχει γρηγορότερα(Τις περισσότερες φορές και με πολυνημάτωση).

Ο κώδικας αναπτύχθηκε και δοκιμάστηκε στο υπολογιστικό σύστημα linux. Τα scripts της bash επίσης έτρεξαν στα μηχανήματα λινουξ της σχολής ωστόσο τα python scripts έτρεξαν σε προσωπικό υπολογιστή με έδοση 3.6.13 όπου εγκαταστήθηκαν και οι απαιτούμενες βιβλιοθήκες. Να σημειωθεί ότι οι πίνακες και τα δεδομένα που παρουσιάζονται πιο πάνω μπορεί να παρουσιάσουν μερικές μικρές αλλοιώσεις σε σχέση με την θεωρία (αυτά που δηλαδή θα περιμέναμε να βγουν) καθώς στο μηχάνημα που τρέχαμε τα προγράμματα συχνά ήταν και άλλοι συνδεδεμένοι. Συγκεκριμένα χρησιμοποιήθηκε το μηχάνημα με αριθμό 25 του οποίου τα χαρακτηριστηκά είναι τα εξής:

Operating system: VERSION="18.04.6 LTS (Bionic Beaver)"

Architecture: x86_64

CPU op-mode(s): 32-bit, 64-bit

CPU(s): 4

Thread(s) per core: 1

Core(s) per socket: 4

Socket(s): 1

Model name: Intel(R) Core(TM) i5-6500 CPU @ 3.20GHz

Cache line size: 64

Compiler version: gcc version 7.5.0