Παράλληλα συστήματα - Χειμερινό ‘22

Εργασία 3/18-1-2022

Ομάδα:

sdi2000064 Κωνσταντίνος Κανελλάκης

sdi2000212 Μιχάλης Χατζησπύρου

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

Άσκηση 3.1

Για τον υπολογισμό των βελών εντός του κύκλου αναπτύχθηκε μία συνάρτηση η οποία καλείται από την υλοποίηση με MPI. 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). Επιπλέον, η διεργασία 0 μεταδίδει τις κατάλληλες πληροφορίες στα υπόλοιπα νήματα μέσω της συνάρτησης MPI_Bcast. Κάθε νήμα-διεργασία καλεί την συνάρτηση monte_carlo η οποία επιστρέφει το τοπικό πλήθος των βελών μέσα στον κύκλο και ύστερα το προσαρτά στην μεταβλητή arrows μέσω της συνάρτησης MPI_Reduce. Η διεργασία 0 κάνει τους τελικούς υπολογισμούς για το π και το εκτυπώνει. Τέλος, για την χρονομέτρηση της εκτέλεσης του προγράμματος χρησιμοποιήθηκε η συνάρτηση MPI_Wtime όπως ακριβώς και στο δωσμένο πρόγραμμα για την άσκηση 3.2.

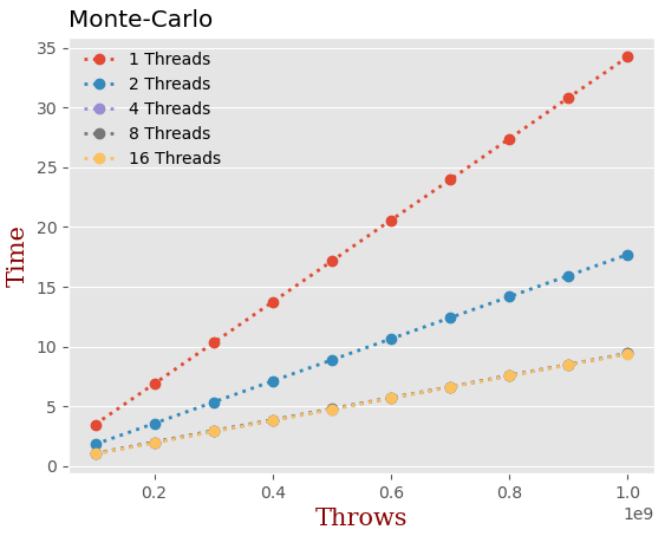
Για την εκτίμηση των αποτελεσμάτων των δύο αλγορίθμων δημιουργήθηκαν δύο επιπλέον αρχεία. Το script.sh παίρνει ως ορίσματα τον αριθμό των ρίψεων και τον αριθμό των νημάτων από τον οποίο θα ξεκινήσει. Για κάθε αριθμό ρίψεων εκτελείται το πρόγραμμα 4 φορές και υπολογίζει τον μέσο όρο εκτέλεσης του προγράμματος. 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 πολλαπλασιάζεται ο αριθμός των νημάτων με 2 και εκτελείται όλη η προηγούμενη διαδικασία. Αυτό εκτελείται για 4 διαφορετικές τιμές νημάτων. Στη συνέχεια το παραγόμενο αρχείο "averages.txt"(στον φάκελο υπάρχουν τα αρχεία averages8 και averages9 τα οποία περιέχουν πειραματικές τιμές με εύρος 10^8-10^9 και 10^9-10^{10} αντίστοιχα για να εκτελέσετε το script plots.py σε περίπτωση που θέλετε) 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 1,2,4,8,16 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	3.141952	3.141735	3.14163	3.141572	3.141546	3.141523	3.141547	3.141562	3.14158	3.141608
1 Threads	3.141952	3.141735	3.141630	3.141572	3.141546	3.141523	3.141547	3.141562	3.141580	3.141608
2 Threads	3.141855	3.141813	3.141764	3.141712	3.141678	3.141662	3.141649	3.141651	3.141655	3.141631
4 Threads	3.141771	3.141713	3.141757	3.141736	3.141691	3.141684	3.141658	3.141650	3.141633	3.141650
8 Threads	3.141748	3.141678	3.141580	3.141635	3.141676	3.141701	3.141686	3.141658	3.141665	3.141639
16 Threads	3.141726	3.141618	3.141585	3.141620	3.141585	3.141559	3.141592	3.141590	3.141597	3.141610

	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	3.547069	7.013386	10.26284	13.66805	17.07544	20.51036	23.91524	27.31771	30.72228	34.12236
1 Threads	3.4673205	6.8792360	10.3039598	13.7322520	17.1542590	20.5886890	23.9760328	27.4118145	30.8455610	34.2599328
2 Threads	1.8018400	3.5517815	5.3339565	7.1051060	8.8685640	10.6392643	12.4066725	14.1680740	15.9371940	17.7078345
4 Threads	1.0064358	1.9733945	2.8834080	3.8158100	4.7508363	5.6916693	6.6158800	7.5727790	8.4861178	9.4089438
8 Threads	1.0450588	1.9810505	2.9486275	3.8403938	4.7830575	5.7061263	6.6181815	7.5707750	8.4888703	9.4125443
16 Threads	1.0098990	1.9156930	2.8806973	3.7821295	4.7258353	5.6738735	6.6020253	7.5312458	8.4674588	9.3656638

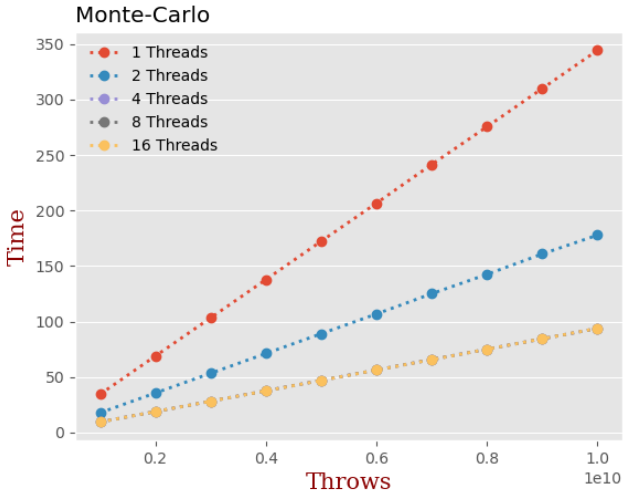
Όπως φαίνεται, από το διάγραμμα και τα δεδομένα του πίνακα, όσο αυξάνονται οι ρίψεις της συνάρτησης monte carlo τόσο αυξάνεται και ο χρόνος των υλοποιήσεων μας. Ωστόσο ο σειριακός αλγόριθμος αυξάνεται με πολύ μεγαλύτερο ρυθμό σε σχέση με την υλοποίηση με MPI όπως και περιμέναμε. Αυτό βέβαια δεν ισχύει όταν τρέχουμε το MPI πρόγραμμα μας με 1 νήμα όπου οι χρόνοι εκτέλεσης είναι παρόμοιοι ενώ οι προσεγγίσεις του π είναι ίδιες. Στην υλοποίηση με MPI παρατηρούμε ότι η αύξηση των νημάτων βελτιώνει την απόδοση του προγράμματος μέχρις ότου να ξεπεράσουμε τα 4 νήματα με αποτέλεσμα ο χρόνος εκτέλεσης του προγράμματος να μην επιφέρει καμιά περαιτέρω βελτίωση. Αυτό επιβεβαιώνεται τόσο από τα δεδόμενα του πίνακα όσο και από το διάγραμμα στο οποίο η γραφική παράσταση των 16 νημάτων έχει καλύψει τις γραφικές παραστάσεις των 4 και 8 νημάτων. Επίσης, 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). Ακόμα, παρατηρούμε ότι με 16 νήματα η προσέγγιση του π γίνεται καλύτερα από κάθε άλλη πολυνηματική εκτέλεση(2,4,8 νήματα). Ωστόσο, ο σειριακός αλγόριθμος προσεγγίζει καλύτερα την τιμή του π μέχρι και 10^9 ρίψεις σε σχέση με την εκτέλεση πολλών νημάτων σε MPI.



Για αριθμό ρίψεων 10^9 – 10^{10} και 1,2,4,8,16 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	3.141608	3.141601	3.141588	3.141595	3.141586	3.141593	3.141589	3.141594	3.141592	3.141595
1 Threads	3.141608	3.141601	3.141588	3.141595	3.141586	3.141593	3.141589	3.141594	3.141592	3.141595
2 Threads	3.141631	3.141626	3.141614	3.141605	3.141600	3.141601	3.141605	3.141600	3.141598	3.141598
4 Threads	3.141650	3.141588	3.141589	3.141598	3.141610	3.141597	3.141598	3.141597	3.141600	3.141595
8 Threads	3.141639	3.141617	3.141590	3.141589	3.141585	3.141581	3.141585	3.141591	3.141599	3.141605
16 Threads	3.141610	3.141611	3.141576	3.141585	3.141579	3.141584	3.141587	3.141588	3.141594	3.141587

	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	34.2153	68.39258	102.4135	136.5145	170.7125	209.8778	241.5023	274.0661	307.0547	343.6256
1 Threads	34.48253375	68.87985350	103.31166925	137.79354200	172.34504600	206.65397600	241.52709025	275.50330150	309.96022025	344.37372825
2 Threads	17.79131350	35.58090925	53.35703250	71.16641775	88.99954700	106.77915400	125.08552525	142.49986300	160.87477075	177.85422125
4 Threads	9.50042000	18.83587250	28.21093600	37.65692825	46.97499925	56.32281850	65.71863575	75.06031125	84.42522500	93.81568775
8 Threads	9.47323525	18.85325000	28.19296075	37.57350800	46.97287375	56.32091200	65.75469400	75.04139300	84.42252800	93.78400175
16 Threads	9.44863825	18.86736025	28.20353625	37.55880375	46.90272150	56.27722375	65.63208850	75.04905625	84.44226550	93.92350750



Σε γενικές γραμμές οι παρατηρήσεις μας ταυτίζονται με αυτές των δεδομένων για $10^8 - 10^9$. Ωστόσο παρατηρώντας τις τιμές του πίνακα για την προσέγγιση του π παρατηρούμε ότι τα 16 νήματα δεν το προσεγγίζουν καλύτερα πάντα συγκριτικά με τις υπόλοιπες πολυνηματικές εκτελέσεις.

Άσκηση 3.2

Για τον υπολογισμό της μήτρας διανύσματος κατά στήλες τροποποιήθηκε το πρόγρμμα που μας δόθηκε καταλλήλως. Συγκεκριμένα λαμβάνει τις διαστάσεις του τετραγωνικού πίνακα ως όρισμα στην γραμμή εντολών καθώς και την επιλογή για το αν ο πίνακας και το διάνυσμα θα δημιουργείται τυχαία ή θα δίνεται από τον χρήστη. Το πρώτο όρισμα αφορά την διάσταση και το δεύτερο το πως θα φτιαχτεί ο πίνακας και το διάνυσμα (0 για να δωθεί από τον χρήστη και 1 για παραγωγή τυχαίου πίνακα και διανύσματος). Για τον πειραματισμό ορθότητας του προγράμματος μπορείτε να βάλετε ως είσοδο το αρχείο test.txt το οποίο υπολογίζει τον παρακάτω πολλαπλασιαμό:

1	2	3	4	5	6	7	8		0.5		102
9	10	11	12	13	14	15	16		1		246
17	18	19	20	21	22	23	24		1.5		390
25	26	27	28	29	30	31	32	X	2	=	534
33	34	35	36	37	38	39	40		2.5		678
41	42	43	44	45	46	47	48		3		822
49	50	51	52	53	54	55	56		3.5		966
57	58	59	60	61	62	63	64		4		1110

Για την παρουσίαση των αποτελεσμάτων πρέπει να προστεθεί το flag -DDEBUG κατά τη μεταγλώττιση για να εκτυπωθούν τα A,x,y.

Γενικότερα χρειάστηκαν να τροποποιηθούν οι περισσότερες συναρτήσεις, συγκεκριμένα:

- Η Get_dims λαμβάνει δύο επιπλέων ορίσματα τα char **argv και int *flag για την κατάλληλη αρχικοποίηση των διαστάσεων του πίνακα και της επιλογής εισόδου των πινάκων από τον χρήστη ή της τυχαίας δημιουργίας αυτών.
- Η Allocate_arrays τροποποιήθηκε ώστε να δεσμεύει m θέσεις για τον πίνακα y καθώς κάθε νήμα υπολογίζει ένα μέρος από κάθε στοιχείο του τελικού διανύσματος y.
- Εφόσον στη Read_Matrix το κύριο νήμα θα διαβάζει ολόκληρο τον πίνακα ώστε να κατανείμει τα επιμέρους τμήματα του(ανά στήλες) στα υπόλοιπα νήματα αποφασίσαμε να αλλάξουμε τις θέσεις αποθήκευσης των στοιχείων καθώς διαβάζει τον πίνακα. Έτσι ο πίνακας A αποθηκεύεται ως ο ανάστροφος του πραγματικού πίνακα όπου θέλουμε να υπολογίσουμε.
- Για τον παραπάνω λόγο, στην Print_matrix εμφανίζουμε τα στοιχεία του A κατά στήλες έτσι ώστε να πάρουμε τον πραγματικό.
- Στη Mat_vect_mult αφαιρέσαμε την κλήση της MPI_Allgather καθώς τα νήματα δεν χρειάζονται όλο το διάνυσμα x αλλά μόνο το μέρος του x που έχει αποθηκεύσει τοπικά. Εφόσον κάθε νήμα υπολογίζει ένα μέρος για όλα τα στοιχεία του y χρησιμοποιούμε έναν βοηθητικό πίνακα my_y m θέσεων, προκειμένου να αποθηκεύσουμε τους τοπικούς υπολογισμούς του y μας, οι οποίοι στη συνέχεια προστίθενται στο y με την κλήση της συνάρτησης MPI_Reduce. Για τον υπολογισμό των τοπικών στοιχείων του y στο my_y χρησιμοποιείται ένα εμφωλευμένο for το οποίο για κάθε στήλη του αρχικού πίνακα A που έχει αναλάβει(local_n στήλες του A) υπολογίζει το αντίστοιχο μέρος αθροίσματος του κάθε στοιχείου της κάθε γραμμής του A(m γραμμές). Μετά την κλήση της συνάρτησης MPI_Reduce από όλα τα νήματα το διάνυσμα y περιέχει το αποτέλεσμα του πολλαπλασιασμού μήτρας-διναύσματος με κατανομή κατά μπλοκ στήλες της μήτρας.

Λόγω των απρόβλεπτων αλληλεπιδράσεων μεταξύ προγράμματος και του υπόλοιπου συστήματος (κυρίως του λειτουργικού), τα πειραματικά μας δεδομένα αφορούν τον ελάχιστο χρόνο εκτέλεσης και όχι τον μέσο όρο. Για τον σκοπό αυτό, δημιουργήθηκαν δύο script ένα σε bash και ένα σε python. Το πρώτο εκτελείται με την εντολή ./script.sh 1 και τρέχει κάθε πρόγραμμα (πολλ/σμός κατά γραμμές και πολλ/σμός κατα στήλες) για 1,2,4,8,16 νήματα 8 φορές για διαστάσεις πίνακα 1024*1024,2048*2048, 4096*4096, 8192*8192, 16384*16384, 32768*32768. Στη συνέχεια, αποθηκεύει τα παραπάνω αποτελέσματα σε ένα αρχείο output.txt, το οποίο λαμβάνει το script table.py και υπολογίζει τον μικρότερο χρόνο εκτέλεσης από τις 8 φορές που το έχει τρέξει το προηγούμενο script και δημιουργεί τον παρακάτω πίνακα:

	1024*1024	2048*2048	4096*4096	8192*8192	16384*16384	32768*32768
Per rows						
1 Thread	0.003244	0.012961	0.051924	0.207809	0.830606	3.320757
2 Thread	0.00164	0.006513	0.026222	0.104142	0.418466	1.677047
4 Thread	0.015676	0.034614	0.040806	0.136314	0.432199	1.534631
8 Thread	0.002112	0.008284	0.027394	0.108007	0.423564	1.585795
16 Thread	0.003108	0.007659	0.028243	0.110147	0.436201	1.652582
Per columns						
1 Thread	0.003052	0.012269	0.049557	0.19793	0.799504	3.217953
2 Thread	0.001633	0.006461	0.026215	0.10284	0.412875	1.643318
4 Thread	0.003169	0.003837	0.037109	0.117721	0.458587	1.729858
8 Thread	0.002629	0.008779	0.027162	0.107017	0.434021	1.68563
16 Thread	0.003155	0.01151	0.031514	0.120791	0.453632	1.677213