Παράλληλα Συστήματα
Εργασία: Game Of Life
Δημήτριος Μπούσουλας
1115201500106

1. Εισαγωγή

Στην παρούσα εργασία ασχολήθηκα με την υλοποίηση του Game Of Life. Η υλοποίηση έγινε με στόχο την παραλληλοποίηση του αλγορίθμου με την χρήση των MPI και OpenMP. Επίσης στην εργασία αυτή έκανα μελέτη κλιμάκωσης για διάφορες τιμές του μεγέθους του προβλήματος αλλά και του πλήθους των πόρων (proccesses και threads) .

2. Περιγραφή

2.1 Γενικά

Σε ότι αφορά τον κώδικα. Αυτός αποτελείται από 4 αρχεία:

1) main.c
2) functions.c
3) functions.h
4) Makefile

Η μεταγλώττιση γίνεται με την εντολή make και η εκτέλεση με την εντολή:
mpirun -np P ./gol N
, όπου P ο αριθμός των διεργασιών και N το μέγεθος του πίνακα (N*N).

Η μελέτη έγινε ξεχωριστά για MPI, MPI + AllReduce και MPI + AllReduce + OpenMp, αλλά όπως ζήταγε η εκφώνηση ο κώδικας που παραδόχηκε είναι η υλοποίηση του τρίτου από τα παραπάνω.

2.2 Υλοποίηση

Η main αρχικά ελέγχει αν οι τιμές που δόθηκαν (P, N) είναι επιτρεπτές ώστε ο πίνακας να σπάσει σωστά σε μπλοκς. Έπειτα υπολογίζει το blockSize ώστε κάθε διεργασία να αρχικοποιήσει με τυχαίες τιμές το κομμάτι του πίνακα που της αντιστοιχεί. Κάθε διεργασία έχει ενα κομμάτι blockSize*blockSize όμως κάνει malloc για (blockSize+2)*(blockSize+2) ώστε να αποθηκεύσει αργότερα και τα halo points που θα λάβει. Έπειτα βρίσκει και αποθηκεύει κάθε διεργασία τις γειτονικές της εργασίες ώστε να ξεκινήσει η κεντρική επανάληψη, η οποία κάνει κάθε φορά ακριβώς 100 βήματα. Στην κεντρική επανάληψη κάθε proccess στέλνει στους γείτονες του τις τιμές των halo points και μετά δέχεται απ’

τους γείτονες τις δικές του. Εκεί καλείται η συνάρτηση για τον υπολογισμό της επόμενης γενιάς των εσωτερικών σημείων. Ακολουθούν οι wait για τα send και receive ώστε να κληθεί και η συνάρτηση για την νέα γενιά των εξωτερικών σημείων. Έπειτα καλείται το AllReduce για τον έλεγχο της μη αλλαγής ή του κενού πίνακα. Στο τέλος της επανάληψης κάνω swap τον cur πίνακα με τον next.

Στο τέλος του προγράμματος απελευθερώνω ότι δέσμευσα δυναμικά. Σε ότι αφορά το OpenMp το έχω χρησιμοποιήσει αποκλειστικά στις συναρτήσεις που βρίσκονται στο functions.c.

3. Μελέτη Κλιμάκωσεις

3.1 Πίνακες Αποτελεσμάτων

MPI

Times

	1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
840	2.18	0.644	0.388	0.308	0.249	0.176	0.134	0.106
1680	9.036	2.459	1.545	1.495	0.963	0.669	0.504	0.384
3360	37.645	9.781	6.163	5.921	3.803	2.643	1.958	1.507
6720	144.035	39.703	24.426	22.008	15.160	10.536	7.747	5.932
13440	557.994	158.041	94.847	76.610	60.698	42.137	31.073	23.767

Speedup

	4	**9**	**16**	**25**	**36**	**49**	**64**
840	3,39	5,62	7,08	8,76	12,4	16,27	20,57
1680	3,68	5,85	6	9,38	13,5	17	23,5
3360	3,85	6,1	6,36	9,9	14,24	19,23	24,98
6720	3,63	5,9	6,55	9,5	13,67	18,6	24,29
13440	3,53	5,88	7,28	9,19	13,24	17,96	23,48

Efficiency

	4	9	16	25	36	49	64
840	0,85	0,63	0,44	0,35	0,34	0,33	0,32
1680	0,92	0,65	0,38	0,38	0,37	0,35	0,35
3360	0,96	0,68	0,4	0,4	0,39	0,39	0,39
6720	0,91	0,66	0,41	0,38	0,38	0,38	0,38
13440	0,88	0,65	0,46	0,37	0,37	0,36	0,36

MPI + AllReduce

Times

	1	4	9	16	25	36	49	64
840	2.282	0.684	0.385	0.385	0.289	0.207	0.159	0.126
1680	9.043	2.386	1.529	1.167	1.110	0.780	0.589	0.448
3360	36.157	9.620	6.122	6.057	4.391	3.058	2.247	1.732
6720	137.636	38.448	24.415	22.008	17.429	12.145	8.976	6.867
13440	546.831	146.524	97.118	80.157	69.893	48.590	35.752	27.431

Speedup

	4	9	16	25	36	49	64
840	3.34	5.93	5.93	7.9	11	14,35	18,11
1680	3.79	5.91	7.75	8.15	11.59	15.35	20.19
3360	3.76	5.91	5.97	8.23	11.82	16.09	20.88
6720	3.58	5.64	6.25	7.9	11.33	15.33	20.04
13440	3.73	5.63	6.82	7.82	11.25	15.3	19.93

Efficiency

	4	9	16	25	36	49	64
840	0.83	0.66	0.37	0.32	0.31	0.29	0.28
1680	0.95	0.66	0.48	0.33	0.32	0.31	0.32
3360	0.94	0.66	0.37	0.33	0.33	0.33	0.33
6720	0.89	0.63	0.39	0.32	0.31	0.31	0.31
13440	0.93	0.63	0.43	0.31	0.31	0.31	0.31

MPI + AllReduce + OpenMp

Times

	1	4	9	16	25	36	49	64
840	2.024	0.612	0.462	0.362	0.240	0.173	0.133	0.107
1680	8.794	2.365	1.430	1.387	0.917	0.654	0.481	0.374
3360	35.982	9.403	6.009	5.589	3.604	2.531	1.853	1.431
6720	138.600	37.180	23.590	22.142	14.152	10.120	7.372	5.661
13440	539.779	155.113	95.256	79.550	56.754	39.797	29.078	22.254

Speedup

	4	9	16	25	36	49	64
840	3.31	4.38	5.59	8.43	11.7	15.22	18.92
1680	3.72	6.15	6.34	9.59	13.45	18.28	23.51
3360	3.83	5.99	6.44	9.98	14.22	19.42	25.14
6720	3.73	5.88	6.26	9.79	13.7	18.8	24.48
13440	3.48	5.67	6.79	9.51	13.56	18.56	24.26

Efficiency

	4	**9**	**16**	**25**	**36**	**49**	**64**
840	0.83	0.49	0.35	0.34	0.32	0.31	0.3
1680	0.93	0.68	0.4	0.38	0.37	0.37	0.37
3360	0.96	0.67	0.4	0.4	0.39	0.4	0.39
6720	0.93	0.65	0.39	0.39	0.38	0.38	0.38
13440	0.87	0.63	0.42	0.38	0.38	0.38	0.38

3.2 Σχολιασμός Αποτελεσμάτων

Καταρχάς και στις 3 υλοποιήσεις είχαν μεγάλη διαφορά οι χρόνοι ανάλογα με τον διαμοιρασμό των proccesses στους κόμβους. Αν για παράδειγμα στα 36 proccesses έτρεχα 6 κόμβους με 6 procs ο καθένας είχα μεγαλύτερους χρόνους απ' τους 4 κόμβους και 9 procs ο καθένας. Συνεπώς έπειτα από αρκετές δοκιμές κατέληξα στους εξής βέλτιστους συνδυασμούς:
4 proccesses : Έτρεξα 1 κόμβο με 4 proccesses
9 proccesses : Έτρεξα 3 κόμβους με 3 proccesses ο κάθε κόμβος
16 proccesses : Έτρεξα 4 κόμβους με 4 proccesses ο κάθε κόμβος
25 proccesses : Έτρεξα 5 κόμβους με 5 proccesses ο κάθε κόμβος
36 proccesses : Έτρεξα 6 κόμβους με 6 proccesses ο κάθε κόμβος
49 proccesses : Έτρεξα 7 κόμβους με 7 proccesses ο κάθε κόμβος
64 proccesses : Έτρεξα 8 κόμβους με 8 proccesses ο κάθε κόμβο

Όπως ήταν αναμενόμενο με τη χρήση του MPI παρατηρώ ότι όσο αυξάνουμε τα proccesses έχουμε μείωση του χρόνου δηλαδή αύξηση του speedup. Αξιοσημείωτο είναι ότι σε κανένα απ' τα 5 μέγεθη που το έτρεξα δεν έφτασε στο σημείο να δίνω περισσότερα proccceses και ο χρόνος να αυξάνεται, γεγονός το οποίο δεν περίμενα να συμβεί, ιδιαίτερα στα μικρότερα μεγέθη καθώς πίστευα ότι ο παράπλευρος χρόνος για την επικοινωνία των διεργασιών θα ήταν μεγαλύτερος απ' τον υπολογιστικό χρόνο που γλυτώνω με την παραλληλοποίηση.

Με τη χρήση MPI + AllReduce η μόνη διαφορά που υπήρξε ήταν μία μικρή αύξηση των χρόνων. Αυτό είναι κάτι που περιμέναμε καθώς είναι ο ίδιος κώδικας με την προσθήκη του ελέγχου μη αλλαγής του πίνακα ο οποίος αυξάνει τον χρόνο εκτέλεσης εφόσον ακόμα και αν εντοπιστεί το κριτήριο τερματισμού η επανάληψη θα ολοκληρώσει και τα 100 βήματα και θα συνεχίσει να κάνει τον έλεγχο σε κάθε βήμα.

Σε ότι αφορά το MPI + AllReduce + OpenMP έπειτα από διάφορες δοκιμές κατέληξα στο συμπέρασμα ότι για όλα τα proccesses το καλύτερο speedup το πετυχαίνω με 4 threads. Συνεπώς οι παραπάνω μετρήσεις έχουν γίνει όλες με 4 threads. Η βασική διαφορά του από το απλό MPI ήταν πως όλοι οι χρόνοι ήταν μειωμένοι γεγονός το οποίο ήταν αναμενόμενο και επιθυμητό καθώς πολλές επαναλήψεις του αλγορίθμου παραλληλοποιούνται σε 4 threads.

Και για τις 3 παραπάνω υλοποιήσεις τα συμπεράσματα για speedup και efficiency είναι τα ίδια. Συγκεκριμένα όσο αυξάνεται το μέγεθος του προβλήματος αλλά κυρίως και το πλήθος των proccesses το speedup αυξάνεται. Το καλύτερο speedup παρατηρείται στα 64 proccesses στα 3360 και 6720.
Σε ότι αφορά το efficiency παρατηρώ ότι για όλα τα μεγέθη είναι καλύτερο στα 4 proccesses και όσο αυξάνουμε τις διεργασίες μειώνεται, με αποκορύφωμα στις 64 φτάνει ακόμα και στο 0.3. Ενδιαφέρον είναι ότι στα μεγαλύτερα μεγέθη η πτώση του efficiency καταλήγει να είναι αμελητέα , δηλαδή μικρότερη από 0.01.

Όπως περιμέναμε τα αποτελέσματα συμφωνούν με τον νόμο του Amdahl καθώς η με έναν πολλαπλασιασμό χ των διεργασιών έχουμε υποπολλαπλασιασμό του χρόνου που είναι μικρότερος από χ. Συγκεκριμένα στο OpenMP για N=1680 έχουμε:
Με P=1 8.794 δευτερόλεπτα ενώ με P=4 έχουμε 2.365 δευτερόλεπτα
Όπου ενώ οι διεργασίες τετραπλασιάστηκαν η επιτάχυνση δεν αν 4 αλλά 3,72. Αυτό όπως συμβαίνει επείδη υπάρχει ένα κομμάτι του πργράμματος που δεν παραλληλοποιείται και διότι επιδρά και ο παράπλευρος χρόνος.

4. Παραδοχές

1) Ο αλγόριθμος ξεκινά υποχρεωτικά από μία τυχαία κατάσταση και δεν μπορεί να διαβάσει μία αρχική κατάσταση

2) Κάνει ακριβώς 100 επαναλήψεις

3) Οι μετρήσεις έγιναν με 1 ,4 ,9 ,25, 36, 49 και 64 proccesses αλλά όχι με 80

4) Οι μετρήσεις για το Open Mp έπειτα από διάφορες δοκιμές αποφάσισα πως έχουν καλύτερο speedup με 4 threads

5) Το μέγεθος του προβλήματος το αυξάνω με διπλασιασμό ξεκινώντας από 840. Φτάνω μέχρι το 13440 καθώς οι σειριακοί χρόνοι από κει και πέρα ξέφευγαν, δηλαδή θα ξεπερνούσαν τα 10 λεπτά

6) Δεν υλοποίησα το πρόγραμμα σε CUDA