Παράλληλα Συστήματα

Ιωάννης Λάμπρου
A.M. 1115201400088
Στέφανος Παντούλας
A.M. 1115201400138
Κωνσταντίνος Στεφανίδης - Βοζίκης
A.M. 1115201400192

1 Εισαγωγή

Στην παρούσα εργασία ασχοληθήκαμε με το Game of life. Μέσω της προσπάθειας ανάπτυξης του με παράλληλο πρόγραμμα, πειραματιστήκαμε με τις διάφορες μεθόδους παραλληλίας και ήρθαμε στην πράξη αντιμέτωποι με τα συνηθέστερα θέματα που προκύπτουν σε αυτό το είδος προγραμματισμού. Στον παράλληλο προγραμματισμό καλείται κανείς να συνυπολογίσει παραμέτρους διαφορετικά δεν έχουν σημασία όπως η τοπολογία διεργασιών και ο διαμοιρασμός δεδομένων και αυτό αποδείχθη μια καλή διεύρυνση των δεξιοτήτων μας. Συνολικα η αλλαγή από σειριακό σε παράλληλο προγραμματισμό υπήρξε μια ενδιαφέρουσα και σίγουρα εκπαιδευτική εμπειρία από την οποιά επωφελήθηκε κάθε μέλος της ομάδας.

2 Σχεδιασμός του προγράμματος

2.1 Παραδοχές

Ο κόσμος του Game of life αναπαρίσταται από ένα array. Υπάρχει η δυνατότητα να δωθεί ένας έτοιμο instance του προβλήματος σε αρχείο ειδάλλως δημιουργείται ένα τυχαίο.

Δεχόμαστε οτι η μορφή του έτοιμου input είναι μια ακολουθία απο 0 κ 1 και διαβαζοντας το αρχείο αγνοούμε οτιδήποτε άλλο μεσα σε αυτό (πχ αλλαγή γραμμής). Επίσης δεχόμαστε οτι ο πίνακας θα έιναι τετραγωνικός με πλευρά N. και οτι ο αριθμός διεργασιών που λαμβάνεται ως όρισμα από την γραμμή εντολών θα πρέπει να έχει ακέραια ρίζα ώστε να χωριστεί το πρόβλημα επιτυχώς. Το μέγεθος του προβλήματος ορίζεται από την εντολή #define N που βρίσκεται στην αρχή της main.

2.2 Διαχωρισμός σε block

Για να φτιάξουμε ενα block κοιτάμε τον αριθμό επιθυμητών διεργασιων. Κατόπιν χωρίζουμε τον πίνακα σε τόσα blocks όσα ο αριθμός των διεργασιών και κάθε πλευρά του block αποτελείται από N/ριζα(διεργασιών) στοιχεία.

2.3 Επικοινωνία διεργασιών

Για την επικοινωνία των διεργασιών έχουν υλοποιηθεί 2 datatypes, ένα για την γραμμή και ένα για την στήλη. Ο υπολογισμός των επόμενων γενεών γίνεται μέσω ενός πλήθους επαναλήψεων όπου σε κάθε επανάληψη, το κάθε block ξεκινάει να στείλει στους κατάλληλους γειτονές του τα στοιχεία της περιμέτρου του και αντίστοιχα ζητάει να λαβει τα στοιχεία που χρειάζεται χρησιμοποιώντας της συναρτήσεις Isend Ireceive. Όσο εκτελείται αυτή η διαδικασία, το block προχωράει στον υπολογισμό των εσωτερικών στοιχείων του. Όταν τελειώσει, περιμένει να ολοκληρωθεί η λήψη των γειτονικών του στοιχείων που απαιτούνται για τον υπολογισμό των στοιχείων της περιμέτρου του και κατόπιν τα υπολογίζει.

2.4 Έλεγχος τερματισμού

Για να ελέγξουμε αν το πρόγραμμα πρέπει να τερματιστεί, κάθε 20 επαναλήψεις ελέγχουμε αν το grid έχει παραμείνει το ίδιο με την χρήση της reduce.

2.5 Παράλληλο I/O

Έχει υλοποιηθεί και η λειτουργικότητα του παράλληλου I/O. Στην λειτουργικότητα αυτή, όλες οι διεργασίες βλέπουν σε ένα αρχείο. Η διεργασία 0 γράφει σε αυτό το αρχείο και κατόπιν όλες οι διεργασίες διαβάζουν αυτό το αρχείο με ένα συγγεκριμένο offset η κάθε μια. Όταν και η τελευταία διεργασία τελειώσει το διάβασμα από το παραπάνω αρχείο, αυτό διαγράφεται.

2.6 Υλοποίηση OpenMp

Για το υβριδικό παράλληλο πρόγραμμα MPI + OpenMp χρησιμοποιήθηκε το μοντέλο/στυλ master only. Το πρόγραμμα χωρίζεται όπως και στην υλοποίηση του MPI και οι υπολογισμοί κάθε διαφορετικής διεργασίας MPI παραλληλοποιούνται με την χρήση εντολών OpenMp. Οι επιπλέον εντολές του OpenMp έχουν μπεί μόνο στις επιμέρους συναρτήσεις και όχι στην main.

2.7 Υλοποίηση Cuda

Για την υλοποίηση του προγράμματος σε Cuda ο κόσμος χωρίζεται σε πολλά blocks με τον ίδιο τρόπο με τον οποίο χωρίζεται και στην υλοποίηση του MPI έτσι ώστε κάθε block να αποτελείται το πολύ από έναν max αριθμό στοιχείων (στο δεδομένο πρόγραμμα 1024). Αυτό γίνεται διότι κάθε στοιχείο του block αποτελεί ξεχωριστό thread, δηλαδή κάθε thread εκτελεί μεμονωμένα τους υπολογισμούς για την εύρευση της επόμενης του κατάστασης. Εδώ χρησιμοποιείται επίσης μια υλοποίηση reduction ώστε να ελέγχουμε αν το grid έχει παραμείνει το ίδιο κάθε 20 επαναλήψεις, το οποίο επιτυγχάνεται μέσω κλήσης διαφορετικού kernel.

3 Μετρήσεις

Στις μετρήσεις εξετάζουμε πάντα 200 generations.
Για κάθε πίνακα που θα παρουσιαστεί, θα μετράμε χρόνο εκτέλεσης με διάφορους συνδυασμούς μεγέθους πίνακα (γραμμές) και αριθμό διεργασιών (στήλες).

3.1 Μετρήσεις MPI

3.1.1 Μετρήσεις με έλεγχο τερματισμού

	1	4	9	16
360	18.762023	13.504018	14.027986	13.546089
720	81.731262	25.335949	12.587978	12.063991
1440	320.287434	94.167862	45.676282	25.707971
2880	1258.040491	340.951981	156.875937	92.979428

Speedup

	4	9	16
360	1.389	1.337	1.385
720	3,225	6,492	6,774
1440	3,401	7,016	12,458
2880	3,689	8,019	13,530

Efficiency

	4	9	16
360	0,35	0,14	0,08
720	0,8	0,72	0,42
1440	0,85	0,77	0,77
2880	0,92	0,89	0,84

3.1.2 Μετρήσεις χωρίς έλεγχο τερματισμού

	1	4	9	16
360	18.012767	10.488107	10.883664	10.931848
720	83.330438	24.816697	13.399497	11.659678
1440	301.702551	86.030471	45.313897	27.348655
2880	1285.869037	368.934777	178.480189	98.574544

Speedup

	4	9	16
360	1.717	1.655	1.648
720	3.358	6.219	7.147
1440	3.507	6.658	11.032
2880	3.485	7.204	13.045

Efficiency

	4	9	16
360	0.43	0.18	0.1
720	0.84	0.7	0.45
1440	0.88	0.74	0.69
2880	0.87	0.8	0.81

3.2 Μετρήσεις MPI + OpenMp

	1	4	9	16
360	17.170922	5.419934	10.383929	9.191981
720	75.419711	20.059146	11.151903	10.703941
1440	273.824692	74.943304	39.706794	24.519189
2880	1148.954700	284.141625	115.864191	6.531976

4 Συμπεράσματα μετρήσεων - Μελέτη Κλιμάκωσης

Όταν έγιναν οι μετρήσεις, τα διαθέσιμα pc ήταν 8 και κάθε pc έχει 2 threads. Συνεπώς περιμένουμε η μέγιστη απόδοση να είναι όταν τρέχουμε με πλήθος διεργασιών $n = 16$.

Καταρχήν παρατηρούμε οτι για πολυ μικρές τιμές του N το πρόγραμμα δεν κλιμακώνει πολύ καλά. Για παράδειγμα για $N = 360$ η επιτάχυνση μεταξύ της 1 και των 16 διεργασιών είναι μόλις 1.385 ενώ κατά μέσο όρο οι 4 διεργασίες έκαναν πιο γρήγορη εκτέλεση από τις 16. Για αυτό ευθύνεται οτι το μέγεθος το προβλήματος είναι αρκετά μικρό και κατα συνέπεια το overhead της επικοινωνίας των 16 διεργασιών επηρεάζει την απόδοση σε τέτοιο βαθμό ώστε οδηγούμαστε στο παραπάνω παράδοξο.

Όσο μεγαλώνει το μέγεθος του προβλήματος παρατηρούμε οτι η κλιμάκωση είναι πολύ καλύτερη. Συγκρίνοντας έναν συγκεκριμένο αριθμό διεργασιών για

διαφορετικά μεγέθη προβλήματος διαπιστώνουμε οτι στα μεγαλύτερα μεγέθη πετυχαίνουμε τόσο μεγαλύτερη επιτάχυνση όσο και αποδοτικότητα.

Επίσης για το ίδιο μέγεθος προβλήματος, το πρόγραμμα γίνεται ταχύτερο και αποδοτικότερο όσο αυξάνουμε τον αριθμό των διεργασιών. Σημειώνεται όμως οτι για $n > 16$ το πρόγραμμα θα αρχίσει να χάνει λίγο από την απόδοση του καθώς οι διεργασίες θα γίνονται περισσότερες από τις διαθέσιμες CPU.

Όσον αφορά τον συνδυασμό MPI + OpenMp, εκτελέσαμε το πρόγραμμα αναθέτοντας 1 MPI process ανα μηχανή του cluster αφήνοντας όλα τους υπόλοιπους πόρους να χρησιμοποιηθούν από το MPI + OpenMp και παρατηρούμε οτι αυτός ο συνδυασμός είναι αισθητά πιο γρήγορο από το σκέτο MPI. Σημειώνονται επίσης κάποια παράδοξα όπως superlinear speedup τα οποία οφείλονται σε ξαφνική αποχώρηση πολλών χρηστών από τα μηχανήματα.

Δυστυχώς οι μετρήσεις για το Cuda δεν πραγματοποιήθηκαν λόγω τεχνικών προβλημάτων.

5 Extra μετρήσεις

Για βαθύτερη κατανόηση της συμπεριφοράς του παράλληλου προγραμματισμού, πραγματοποιήσαμε μια σειρά μετρήσεων σε μια επιπλέον CPU δικού μας μηχανήματος. Η CPU η οποία χρησιμοποιήσαμε είναι η CPU: Intel i7 4700MQ μια CPU η οποία αποτελείται από 4 cores.

5.1 Μετρήσεις με έλεγχο τερματισμού

	1	4	9	16	25	36
360	0.358026	0.124393	0.229261	0.243987	0.250476	0.325942
720	1.486048	0.669465	0.916780	0.953203	1.155503	1.201235
1440	5.581889	1.561678	2.283734	1.788274	1.973669	1.958452
2880	22.563305	6.233173	8.419352	6.856979	8.335352	9.752237

5.2 Μετρήσεις χωρίς έλεγχο τερματισμού

	1	4	9	16	25	36
360	0.356983	0.121136	0.193105	0.209974	0.235780	0.250916
720	1.435622	0.441621	0.587944	0.608373	0.688056	0.730334
1440	5.512047	1.672708	2.279074	1.940809	2.101026	2.123812
2880	22.925416	7.387190	8.551683	8.054605	9.766192	9.892658

5.3 Συμπεράσματα από τις extra μετρήσεις

Οι extra μετρήσεις παρήγαγαν μερικά πολύ ενδιαφέροντα αποτελέσματα. Καταρχήν παρατηρούμε οτι αυτή η CPU είναι πολύ πιο αποδοτική από όλα τα μηχανήματα της σχολής στα οποία δοκιμάσαμε προηγουμένως! Αυτό οφείλεται εν μερει στο οτι CPU είναι πολύ καλύτερης τεχνολογίας από της επιμέρους CPU στις οποίες δοκιμάζαμε προηγουμένως αλλά και στο γεγονός οτι στις προηγούμενες CPU υπήρχαν πολλά άτομα τα οποία τις απασχολούσαν ταυτοχρόνως ενώ τώρα απασχολούμε την CPU μας μόνο εμείς.

Επίσης παρατηρούμε οτι το πρόγραμμα είναι μέγιστα αποδοτικό για αριθμό διεργασιών $n = 4$. Αυτό ήταν αναμενόμενο διότι η CPU μας έχει 4 cores.