Ακαδημαϊκό Έτος 2016-2017

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών

Γεωργιάδης Αχιλλέας: 7415

[ΠΑΡΑΛΛΗΛΑ ΚΑΙ ΔΙΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ – ΕΡΓΑΣΙΑ 2]

Περιγραφή Προβλήματος

Το **game of life** είναι ένα κυτταρικό αυτόματο το οποίο κατασκευάστηκε από τον μαθηματικό John Conway το 1970 . Η εξέλιξη του παιχνιδιού καθορίζεται μόνο από τις αρχικές συνθήκες. Ο παίκτης δημιουργεί την αρχική διάταξη και παρατηρεί πώς αυτή εξελίσσεται.

Στη συγκεκριμένη εργασία έπρεπε να τηρηθούν τα εξής:

- cyclic boundary conditions
- χωρισμό του αρχικού πίνακα σε πλέγμα και να ανάθεση κάθε τμήμα του πλέγματος σε διαφορετικές διεργασίες
- εκτέλεση του παιχνιδιού για τρείς γενιές – επαναλήψεις

Για την υλοποίηση των παραπάνω χρησιμοποιήθηκε MPI (Message Passing Interface) για την επικοινωνία μεταξύ των διεργασιών μέσω μηνυμάτων, και OpenMP για την επιτάχυνση του προγράμματος στο επίπεδο της κάθε διεργασίας.

Περιγραφή Μεθόδων

Η εργασία αυτή ουσιαστικά μπορεί να χωριστεί σε δύο τμήματα. Το πρώτο αποτελεί την «άμεση» επιτάχυνση της σειριακής υλοποίησης σε επίπεδο μίας διεργασίας, μέσω παραλληλοποίησης του προγράμματος με χρήση OpenMP. Το δεύτερο αποτελεί την επιπλέον «έμμεση» επιτάχυνση της παράλληλης υλοποίησης, μέσω της κατανομής του προβλήματος σε πολλαπλά μηχανήματα, με χρήση MPI.

Βασικός λόγος χρήσης του MPI είναι η επίλυση του bottleneck της RAM, λόγω του μεγάλου μέγεθος του πίνακα. Εξάλλου σκοπός του MPI είναι η μοντελοποίηση παράλληλου προγραμματισμού σε κατανεμημένα συστήματα μνήμης.

Από τα παραπάνω γίνεται φανερή η διαφορετική φύση των δύο τμημάτων της εργασίας, για αυτό και θα αναπτυχθούν ξεχωριστά.

OpenMP

Η υλοποίηση με openMP πραγματοποιήθηκε στο επίπεδο μίας διεργασίας / μηχανήματος. Η υλοποίηση έγινε με χρήση 8 threads.

Η μορφή των openMP εντολών στα περισσότερα σημεία είναι η παρακάτω:

```
#pragma omp parallel for default(none) shared(board, N1, N2, threshold, tasks_num) private(i ,j, seed) collapse(2)
```

Το στοιχείο που διαφοροποιείται από τις εντολές που χρησιμοποιήθηκαν στην 1η εργασία είναι η χρήση του collapse(2), το οποίο χρησιμοποιείται σε nested for loops τα οποία δεν έχουν εξάρτηση μεταξύ τους.

Ιδιαίτερη μνεία αξίζει να γίνει στη παραλληλοποίηση της generate_table() και πιο συγκεκριμένα στην rand(). Η συνάρτηση rand() δεν είναι thread safe και για αυτό τον λόγο παράγει λανθασμένα αποτελέσματα σε παράλληλο περιβάλλον. Για το λόγο αυτό χρησιμοποιήθηκε η συνάρτηση rand_r().

Επιπλέον να σημειωθεί ότι για seed στην rand δεν δίνουμε απλώς τον αύξοντα αριθμό του κάθε thread, αλλά `time(NULL) ^ omp_get_thread_num()`, ώστε να έχουμε όσο πιο τυχαίους αριθμούς γίνεται. Αν χρησιμοποιούσαμε μόνο τον αριθμό του thread, τότε σε κάθε νέα εκτέλεση του προγράμματος θα είχαμε τον ίδιο ακριβώς πίνακα.

Η χρήση του παραπάνω seed αυξάνει των χρόνο εκτέλεσης κατά περίπου 3 secs, σε σχέση με τον χρόνο εκτέλεσης με seed τον αύξοντα αριθμό thread.

MPI

Η χρήση MPI γίνεται για τη περίπτωση που θέλουμε 2 ή παραπάνω διεργασίες. Στην περίπτωση της μίας διεργασίας, οπού δεν γίνεται χρήση MPI, γίνεται και πάλι initialization του, για μεγαλύτερη γενίκευση του κώδικα του προγράμματος. Κάθε διεργασία είναι και ένα νέο μηχάνημα στο hellasgrid.

Η λογική που ακολουθήθηκε για την υλοποίηση είναι κοινή για 2 ή παραπάνω διεργασίες.

Αρχικά χωρίζεται το board / πίνακας σε έναν αριθμό ισομεγεθών υπό πινάκων, τόσων όσος και ο αριθμός των διεργασιών που θέλουμε. Ο χωρισμός αυτός είναι οριζόντιου προσανατολισμού όπως φαίνεται στο παρακάτω σχήμα:

Rank 0
Rank 1
Rank 2
Rank 3

Η τμηματοποίηση του board γίνεται κατά αυτόν τον τρόπο ώστε να είναι ευκολότερη η επικοινωνία μεταξύ των διεργασιών, μιας και απαιτείται αποστολή στοιχείων μόνο προς τα πάνω ή προς τα κάτω, και όχι προς τα άλλη κατεύθυνση.

Βασική «επέμβαση» στον σειριακό κώδικα είναι προσθήκη δύο επιπλέων γραμμών στον κάθε υποπίνακα που δημιουργούμε. Οι επιπλέον αυτές γραμμές αποτελούν τα links μεταξύ των υποπινάκων και στο εξής θα αναφέρονται ως linking rows.

Πιο συγκεκριμένα έχουμε ένα linking row στην αρχή του υποπίνακα, και ένα στο τέλος. Στις γραμμές αυτές αποθηκεύονται τα γειτονικά στοιχεία των γείτονων υποπινάκων:

- Στο πρώτο linking row (αρχή υποπίνακα) τα στοιχεία της τελευταίας γραμμής που **δεν** είναι linking row, του προηγούμενου υποπίνακα
- Στο τελευταίο linking row (τέλος υποπίνακα) τα στοιχεία της πρώτης γραμμής που **δεν** είναι linking row, του επόμενου υποπίνακα.

Η παραπάνω αρχιτεκτονική διασφαλίζει την τήρηση των cyclic boundary conditions. Το μόνο που χρειάζεται είναι η επικοινωνία του rank0 με το rank3 σύμφωνα με την εξής αλυσίδα επικοινωνίας **rank0 <-> rank1 < -> rank2 < -> rank3 < -> rank0**

Αφού λοιπόν δημιουργηθεί και αρχικοποιηθεί ο κάθε υποπίνακας, κάθε διεργασία στέλνει στους γείτονες της τα στοιχεία που χρειάζονται με τον τρόπο που περιγράψαμε, και περιμένει τα στοιχεία που χρειάζεται η ίδια.

Η επικοινωνία αυτή θέλουμε να είναι ασύγχρονη, ώστε να κρύψουμε τον χρόνο αναμονής. Κατά την αναμονή λοιπόν, κάθε διεργασία υπολογίζει τη νέα κατάσταση όλων των κελιών του υποπίνακα της, εκτός αυτών που εξαρτώνται από τα στοιχεία που αναμένει.

Εφόσον βεβαιωθούμε ότι η αποστολή και λήψη έχει ολοκληρωθεί επιτυχώς, η διεργασία εκτελεί και τον υπολογισμό της νέας κατάστασης των υπόλοιπων (οριακών) στοιχείων.

Για να επιτύχουμε το παραπάνω απαιτείται η χρήση non-blocking επικοινωνίας. Χρησιμοποιούμε λοιπόν τα non-blocking variant της MPI_SEND και MPI_RECV, την MPI_ISEND και MPI_IRECV[1]. Επιπλέον εξασφαλίζουμε την ολοκλήρωση της επικοινωνίας μεταξύ όλων των διεργασιών με την Waitall().

Πειράματα – Σχόλια

Η εκτέλεση του game of life πραγματοποιήθηκε στο hellasgrid, για τις εξής 3 περιπτώσεις μεγεθών του board, για 3 γενιές και 8 threads:

- 40.000 x 40.000 – 1 διεργασία
- 80.000 x 40.000 – 2 διεργασίες
- 80.000 x 80.000 – 4 διεργασίες

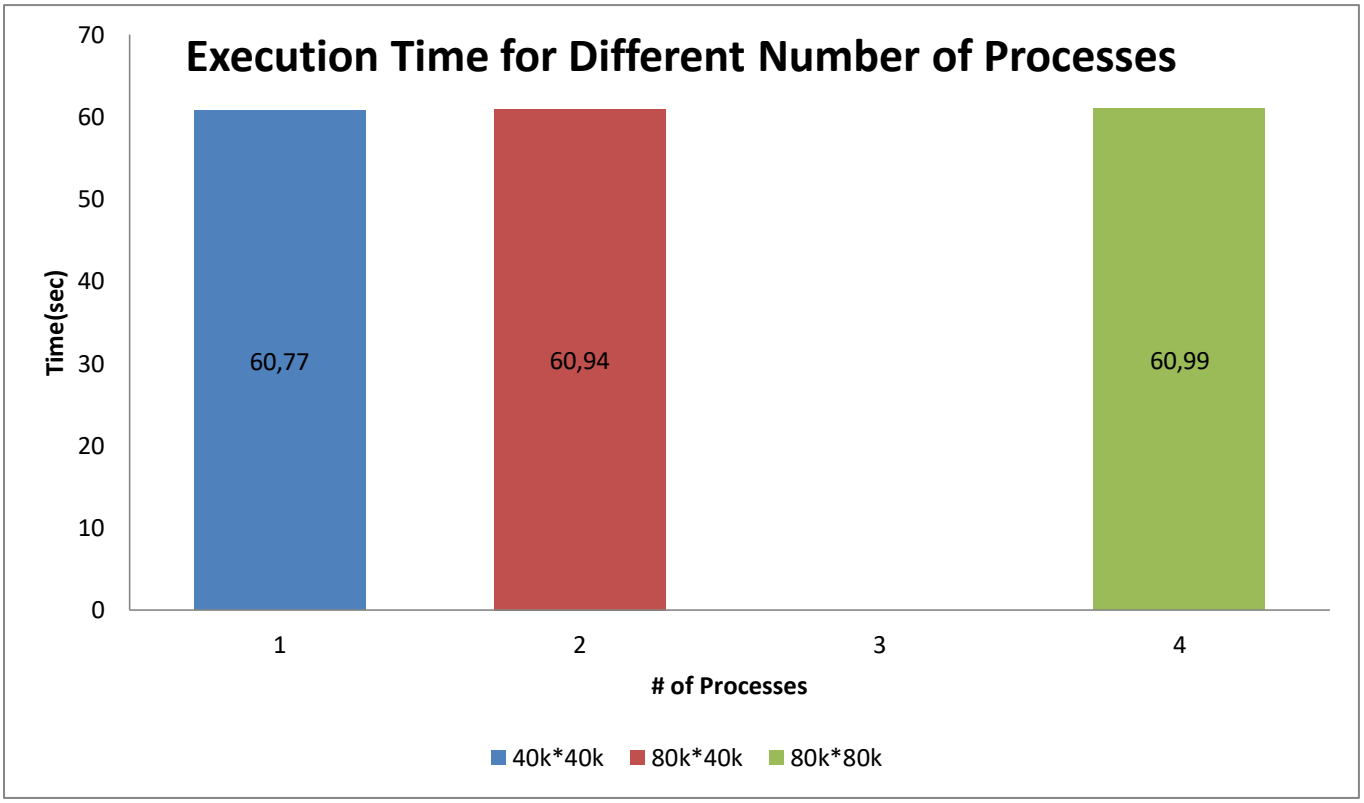
Να σημειωθεί ότι για κάθε διεργασία χρησιμοποιήθηκε και διαφορετικό μηχανημα στο grid.

Στόχος είναι να έχουμε τον ίδιο χρόνο εκτέλεσης και για τα 3 πειράματα, καθώς αν έχει γίνει επιτυχημένο distribution των εργασιών θα πρέπει να ισχύει:

Χρόνος Επίλυσης N μεγέθους προβλήματος με μία διεργασία = Χρόνος Επίλυσης 2*N προβλήματος σε 2 διεργασίας = Χρόνος Επίλυσης 4*N προβλήματος σε 4 διεργασίες κοκ

[1] http://stackoverflow.com/questions/17582900/difference-between-mpi-send-and-mpi-ssend , selected answer

Το παραπάνω γίνεται φανερό ότι ισχύει και από το παρακάτω διάγραμμα.



Ο χρόνος αυτός όπως αναφέρθηκε και στην αρχή είναι κατά 3 sec υψηλότερος σε σχέση με το χρόνο εκτέλεσης του προγράμματος για seed = `omp_get_thread_num()`.

Για περισσότερες πληροφορίες μπορεί κανείς να δει τα log files που περιλαμβάνονται στην αναφορά.