Game of Life using MPI

Παράλληλα & Διανεμημένα Συστήματα : 2η εργασία

Φίλιππος Χρήστου

ΑΕΜ : 8276

fchristou@auth.gr

25/12/2016

Περιεχόμενα

Εισαγωγή

Η παρούσα εργασία αποτελεί μια υλοποίηση του Conway's Game of Life[1] . Είναι ένα <<παιχνίδι>> που επινόησε ο μαθηματικός *John Horton Conway*, για το οποίο έχει αναπτυχθεί αρκετή θεωρία από πίσω, όμως σε πρώτο στάδιο είναι αρκετά απλό.

Το παιχνίδι αποτελείτα από ένα τερέν με οκτάγωνα, όπου κάθε οκτάγωνο είναι ένας ζωντανός ή νεκρός οργανισμός. Κάθε οργανισμός έχει οκτώ γείτονες. Τον ανατολικό, τον βορειοανατολικό, τον βόρειο, τον βορειοδυτικό, τον δυτικό, τον νοτιοδυτικό, τον νότιο και τον νοτιοανατολικό. Το παιχνίδι εξελίσσεται σε γενιές , και σε κάθε γενιά άλλοι οργανισμοί πεθαίνουν , άλλοι παραμένουν νεκροί, άλλοι παραμένουν ζωντανοί , και άλλοι γεννιούνται. Οι κανόνες του παιχνιδιού είναι οι εξής :

1. Οι καταστάσεις των κελιών μεταβάλλονται σε κάθε επανάληψη του παιχνιδιού (γενιά)

2. Stasis: Κάθε κελί με ακριβώς δύο ζωντανούς γείτονες παραμένει στην ίδια κατάσταση στην επόμενη γενιά. Αν ήταν ζωντανό, παραμένει ζωντανό και αντίστροφα.

3. Growth: Κάθε κελί με ακριβώς τρεις ζωντανούς γείτονες θα είναι ζωντανό στην επόμενη γενιά, ανεξάρτητα από την τωρινή του κατάσταση.

4. Death: Κάθε κελί με 0, 1, ή 4 − 8 ζωντανούς γείτονες πεθαίνει στην επόμενη γενιά, ανεξάρτητα από την τωρινή του κατάσταση.

5. Οι αλλαγές των καταστάσεων συμβαίνουν ταυτόχρονα για όλα τα κελιά μιας γενιάς.

Ζητείται ο υπολογισμός του συνολικού πίνακα που προκύπτει μέχρι και για 40000x40000, 40000x80000 και 80000x80000 στοιχεία για 3 γενιές χρησιμοποιώντας 1,2 και 4 διεργασίες αντίστοιχα.

Το παραπάνω παιχνίδι είναι ιδανική εφαρμογή για παραλληλοποίηση καθώς κάθε κελί είναι εντελώς ανεξάρτητο από τα άλλα κελιά αφού οι αλλαγές ανά γενιά συμβαίνουν ταυτόχρονα σε όλα. Οι τρόποι παραλληλοποίησης που χρησιμοποιήθηκαν εδώ είναι με openMP και με MPI.

[1] (Conway's Game of Life, n.d.)

MPI

Η MPI (*Message Passing Interface*)[2] είναι βιβλιοθήκη που επιτρέπει την επικοινωνία μεταξύ διαφορετικών υπολογιστικών συστημάτων, καθώς και την μεταφορά δεδομένων μεταξύ των. Αναφέρεται κυρίως σε συστήματα που έχουν αρχιτεκτονική διανεμημένης μνήμης. Αυτό σημαίνει ότι ο κάθε CPU ενός συστήματος μπορεί να δει μόνο την δικιά του μνήμη και για να γράψει ή για να φορτώσει δεδομένα από άλλο σύστημα με άλλον CPU πρέπει πρώτα να ζητήσει από αυτόν άδεια, να γίνει αποδεκτό το αίτημα και έπειτα να λάβει χώρα η μεταφορά δεδομένων.

Μια πιο σύγχρονη τεχνική μεταφοράς δεδομένων σε διανεμημένα συστήματα είναι αυτή του RMA(*Remote Memory Access*). Σύμφωνα με αυτήν, το κάθε σύστημα επιλέγει ένα μέρος της μνήμης (*window*), το οποίο μπορεί να το κάνει κοινό (shared) σε μια ομάδα διεργασιών (*Communicators*). Έτσι το σύστημα μπορεί να γράφει στην τοπική του μνήμη και τα δεδομένα αυτά να είναι κοινώς προσβάσιμα από ένα σύνολο διεργασιών. Η παραπάνω τεχνική είναι πιο γνωστή ως *one-sided Communication.*

Η MPI χρησιμοποιήθηκε επειδή υπήρχε επιτακτική ανάγκη για περαιτέρω χώρο αποθήκευσης. Ζητείται να υπολογισθεί ο τελικός πίνακας του Game Of Life για 40000x40000 στοιχεία και 1 διεργασία, για 80000x40000 και 2 διεργασίες και για 80000x80000 και 4 διεργασίες. Γίνεται φανερό ότι για το τελευταίο πείραμα απαιτείται 80000*80000*sizeof(int) BYTES, δηλαδή περίπου 25GByte να δεσμευτούν από το σύστημα σε συνεχόμενες θέσεις μνήμης. Αυτό είναι σχεδόν ακατόρθωτο ,οπότε σπάμε το πρόβλημα σε 4 διαφορετικά συστήματα , έτσι ώστε το καθένα από αυτά να δεσμεύει περίπου 6 GByte. Το ίδιο συμβαίνει και για το πείραμα των 80000x40000. Η εσωτερική παραλληλοποίηση σε κάθε υποσύστημα πραγματοποιείται με την openMP.

[2] (MPI tutorial, n.d.)

Τρόπος Διάσπασης Προγράμματος

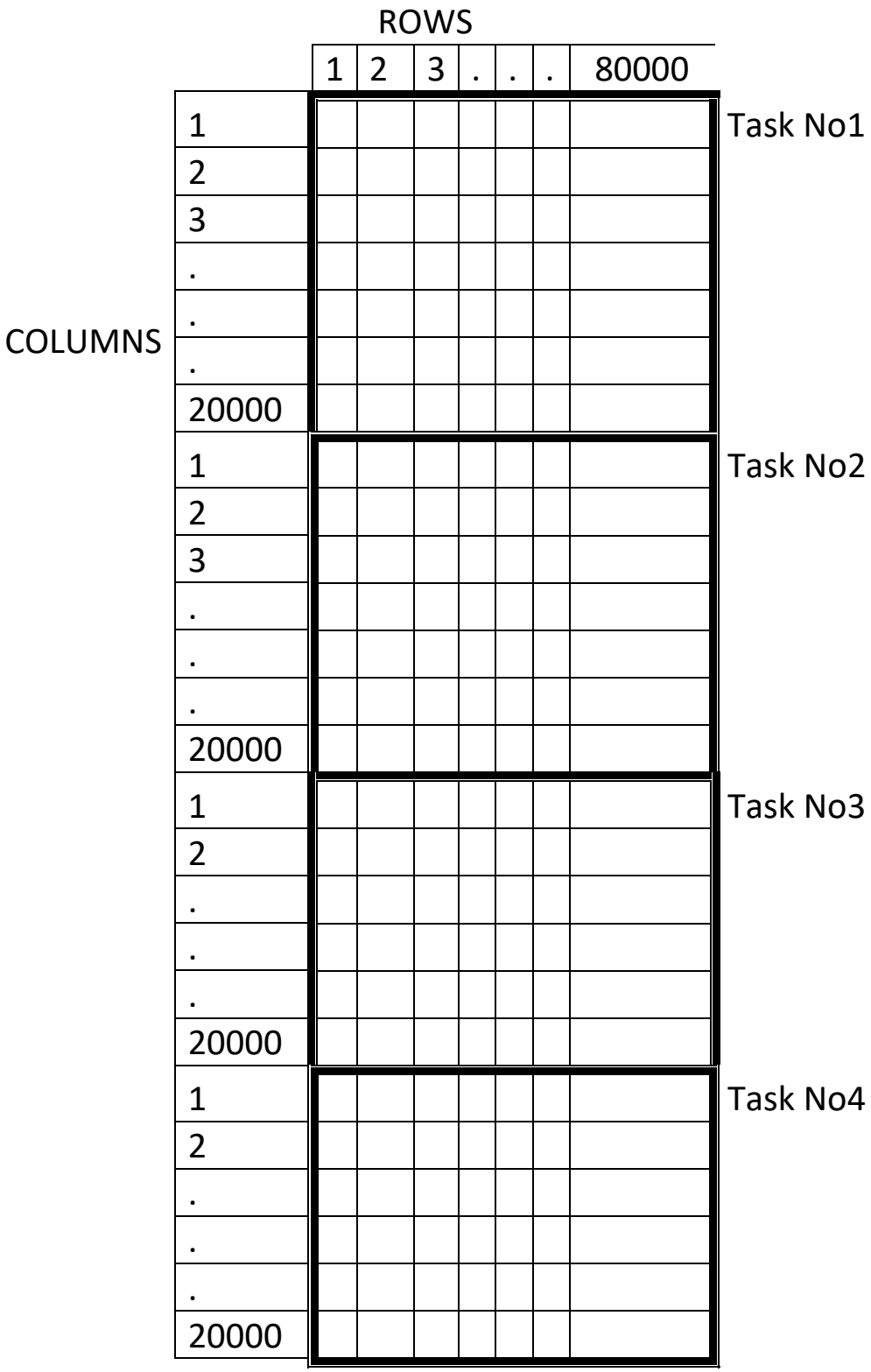
Παρακάτω θα γίνει λόγος για το πώς διαχωρίστηκαν τα δεδομένα έτσι ώστε το πρόγραμμα να τρέξει σε διαφορετικά υπολογιστικά συστήματα. Έστω ότι έχουμε 80000x80000 στοιχεία σε έναν πίνακα. Κάθε στοιχείο μπορεί να είναι είτε 0 είτε 1 , αν ο οργανισμός είναι νεκρός ή ζωντανός αντίστοιχα.

Πίνακας 1

		ROWS						
		1	2	3	.	.	.	80000
COLUMNS	1							
	2							
	3							
	.							
	.							
	.							
	80000							

Στον *Πίνακα 1* βλέπουμε το αρχικό μέγεθος του πίνακα που πρέπει να δεσμευτεί σε συνεχόμενες θέσεις μνήμης. Καθώς είπαμε πως αυτό δεν είναι δυνατόν, θα τον σπάσουμε σε επιμέρους πίνακες που θα διανέμουμε σε ξεχωριστά συστήματα. Κάθε υπολογιστικό σύστημα θα λάβει από 20000x80000 στοιχεία όπως φαίνεται και στον *Πίνακα 2.* Η διαμοίραση των στοιχείων γίνεται έτσι ώστε κάθε υπολογιστικό σύστημα να λάβει το κατά δυνατόν περισσότερο συνεχόμενες θέσεις μνήμης. Καθώς η μνήμη δεσμεύεται καρά στήλη (δηλαδή, η επόμενη θέση μνήμης δείχνει την επόμενη στήλη της ίδιας γραμμής, ή -εφόσον έχουν τελειώσει οι διαθέσιμες στήλες της σειράς - το πρώτο στοιχείο της επόμενης γραμμής). Αυτό σημαίνει ότι για να έχουμε γρήγορη προσπέλαση στην ιδιωτική μνήμη κάθε διεργασίας θα πρέπει τα δεδομένα να αποθηκευτούν κατά γραμμές. Έτσι ο επεξεργαστής τα άλματα που θα χρειάζεται να κάνει ο επεξεργαστής σε θέσεις μνήμης θα είναι ελεγχόμενα και μειωμένα.

Πίνακας 2



Παρόμοια χωρίστηκε η δουλειά και για πίνακα 40000x80000 σε δύο διεργασίες των 20000x80000.

Σχεδιάζοντας το πρόβλημα όπως παραπάνω γίνεται σαφές ότι η επκοινωνία μεταξύ των διεργασιών χρειάζεται μόνο για την $1^{η}$ και τελευταία (δηλαδή $20000^{η}$) γραμμή της κάθε διεργασίας.

Παραλληλοποίηση με MPI

Η τεχνική που χρησιμοποιείται είναι κάθε διεργασία ζητάει την κάτω γραμμή της διεργασίας πάνω από αυτήν και την πάνω γραμμή της διεργασίας που είναι κάτω από αυτήν. Επίσης δίνει την κάτω γραμμή της στην διεργασία πάνω από αυτήν και την κάτω γραμμή της στην διεργασία κάτω από αυτήν. Η επικοινωνία αυτή συμβαίνει με τις *non-blocking* εντολές *Irecv & Isend*. Αφού πρώτα γίνουν αυτές η διεργασία προχωράει στον υπολογισμό όλων των υπόλοιπων 19998 γραμμών ,για τα οποία έχει όλα τα δεδομένα τοπικά στην μνήμη της. Αφού τελειώσει η επεξεργασία αυτών κατά πάσαν πιθανότητα θα έχει συμβεί και η μεταφορά δεδομένων (ακολουθεί και μια *MPI_Wait()* για την επιβεβαίωση). Με αυτόν τον τρόπο <<κρύβεται>> ο χρόνος επικοινωνίας μεταξύ των διεργασιών

Στο Grid η εργασία έτρεξε έτσι ώστε κάθε διεργασία να έχει το δικό της node και 8 επεξεργαστές. Δηλαδή nodes = 4 και ppn = 8. Κάθε διεργασία έπρεπε να έχει το δικό της node διότι έπρεπε να έχει τον δικό της αποθηκευτικό χώρο χώρο.

Παραλληλοποίηση με openMP

Έγινε παραλληλοποίηση με openMP στο υπολογιστικό υποσύστημα. Η παραλληλοποίηση έγινε τόσο στην *initialize_board()* όσο και στην *generate_table()*. Η παραλληλοποίηση στην πρώτη είναι η προφανής. Στην δεύτερη ώστόσο χρειάστηκε να αντικαταστήσω την συνάρτηση *rand()* που δεν είναι tread safe, με την *drand48_r()* που είναι. Έπρεπε να προσέξω όμως το seed για κάθε *thread* της *openMP* να είναι διαφορετικό έτσι ώστε να μην είναι παρόμοια αρχικοποιημένος ο πίνακας στα διάφορα σημεία που ανατίθενται στα *threads* της *openMP*. Βέβαια ,αν οι διεργασίες είναι περισσότερες από μία θα πρέπει να προσέχω να είναι διαφορετικό το *seed* και για κάθε μια από αυτές. Λαμβάνονται οι απαραίτητες προϋποθέσεις για όλα τα παραπάνω. Τέλος η παραλληλοποίηση στην play έγινε εξίσου με τον προφανή τρόπο που δεν είναι άλλος από μια απλή *#pragma omp parallel for*.

Two-sided Communication

Για την αρχικοποίηση του πίνακα όλων των παρακάτω μετρήσεων δόθηκε πιθανότητα 50% για το αν ο οργανσιμός σε κάθε κελί θα είναι ζωντανός ή νεκρός. Επίσης , τροποποιήθηκε ο κώδικας ώστε οι στήλες να γίνουν γραμμές και οι γραμμές στήλες διότι έτσι φαίνονταν πιο ξεκάθαρο στην σκέψη μου.

Παρακάτω φαίνεται ένα σχεδιάγραμμα που δείχνει τον χρόνο που πήρε για την εκτέλεση ενός Game Of Life Table σε 3 γενιές. Το μέγεθος του πίνακα ήταν 40000x40000 και χρησιμοποιήθηκε μια διεργασία. Οι τρεις *playGen* είναι μια για κάθε γεννεά. Έχει γίνει παραλληλοποίηση με *openMP*.

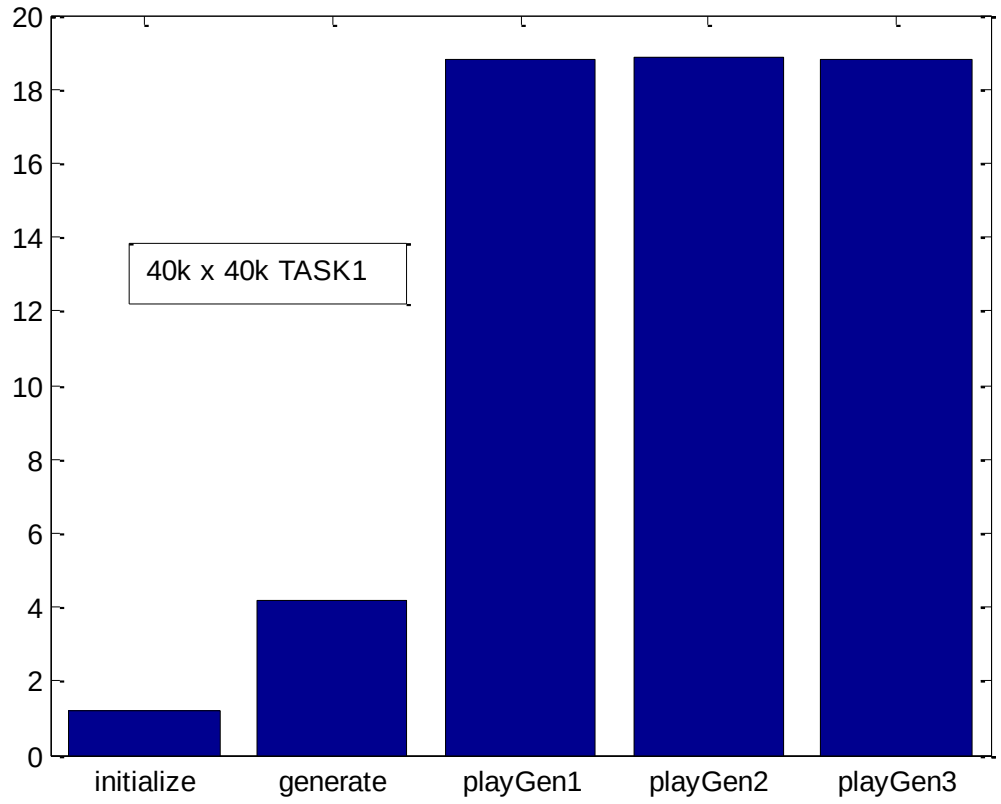


Figure 1

Οι συνολικός χρόνος που έγινε ήταν 1 λεπτό και 2 δευτερόλεπτα : 1’2’’
Αν δεν είχε γίνει εσωτερική παραλληλοποίηση με openMP τότε ο χρόνος αυτός θα ήταν πάνω από 2 λεπτά.

Τα διαγράμματα είναι παρόμοια για τα άλλα δύο πειράματα και οι χρόνοι εξίσου. Αναφέρονται οι χρόνοι και για τα άλλα δύο :

40000x40000 και 1 διεργασία : 1’2’’
40000x80000 και 2 διεργασίες : 1’2’’
80000x80000 και 4 διεργασίες : 1’2’’

Τα αποτελέσματα είναι αναμενόμενα. Κάθε υπολογιστικό σύστημα κάθε φορά κάνει τον υπολογισμό 40000x40000 =1.600.000.000 στοιχείων σταθερά. Το μόνο που αλλάζει είναι ότι όταν οι διεργασίες είναι 2 ή 4 θα πρέπει επίσης να γίνει και μια μεταφορά δεδομένων από το ένα υπολογιστικό σύστημα στο άλλο. Πέραν του ότι αυτό δεν παίρνει πολύ χρόνο συγκριτικά με τα 1,2 δευτερόλεπτα μεριμνούμε έτσι ώστε ο χρόνος επικοινωνίας να <<κρύβεται>>. Αυτό σημαίνει ότι τα υπολογιστικά συστήματα κάνουν αίτηση να πάρουν τα δεδομένα και αντί, να περιμένουν να έρθουν, προχωρούν να κάνουν μια δουλειά που θα γινόταν έτσι και αλλιώς (και δεν εξαρτάται απ’τα δεδομένα τα οποία περιμένουν). Έτσι όταν αυτή η δουλειά τελειώσει κατά πάσαν πιθανότητα θα έχει ολοκληρωθεί και η μεταφορά δεδομένων. Έτσι, ουσιαστικά ο χρόνος επικοινωνίας μεταξύ των συστημάτων είναι σχεδόν μηδενικός.

One-sided Communication

Τα αποτελέσματα και εδώ είναι παρόμοια . Ακολουθείται η ίδια λογική επίσης. Πρώτα η κάθε διεργασία τοποθετεί τις γραμμές που θέλει να δώσει στις άλλες διεργασίες σε μια κοινή μνήμη , έπειτα ζητάει τις γραμμές των άλλων διεργασιών και στη συνέχεια χωρίς να μπλοκάρει (οι εντολές αυτές είναι *non-blocking*), προχωράει στον υπολογισμό των υπόλοιπων 19998 γραμμών. Όταν τελειώσει η επεξεργασία αυτών των γραμμών επιστρέφει να επεξεργαστεί την 1^{η} και τελευταία γραμμή :

40000x40000 και 1 διεργασία : 1’2’’
40000x80000 και 2 διεργασίες : 1’2’’
80000x80000 και 4 διεργασίες : 1’2’’

Ο χρόνος και εδώ είναι ο ίδιος. Κάθε διεργασία αναλαμβάνει 1.600.000.000 στοιχεία. Το μόνο που αλλάζει είναι ο τρόπος επικοινωνίας, ο οποίος όπως βλέπουμε δεν προσδίδει ιδιαίτερη επιτάχυνση ή επιβράδυνση, έναντι του άλλου, στο σύστημα.

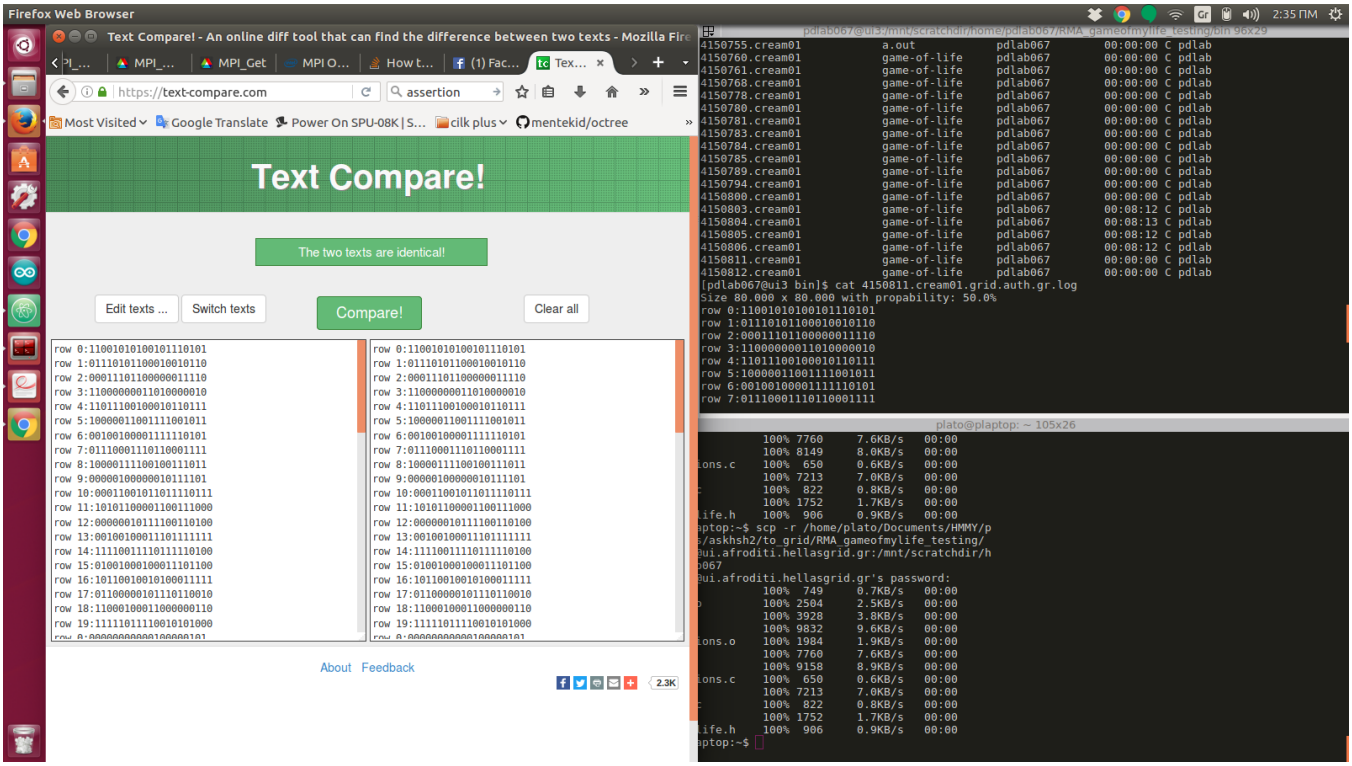
Εγκυρότητα Αποτελεσμάτων

Θέλοντας να αποφευχθούν τυχών λογικά λάθη στον κώδικα (καθώς ήταν αρκετά εύκολο να μην τα παρατηρήσει κανείς) ακολούθησε η εξής διαδικασία :

Τέθηκε ο αριθμός των σημείων να είναι 20x20. Οι οργανισμοί που είχαν ζωή τέθηκαν με μονάδα (έναντι του ‘x’) και οι νεκροί με 0 (ένταντι του κενού χαρακτήτα ΄ ΄). Εκτυπώθηκε το παραπάνω απότέλεσμα (βάζοντας το 3ο όρισμα μονάδα) για τον σειριακό αλγόριθμο τοπικά στον υπολογιστή μου. Το αποτέλεσμα μπήκε σε έναν συγκριτή κειμένου.

Έπειτα έγινε η ίδια διαδικασία για 20x20 στοιχεία τοπικά σε κάθε διεργασία που έπαιζε σε διαφορετικό node. Οι αρχικοποίηση των πινάκων έγινε δίνοντας το ίδιο *seed* και συνεπώς ο αρχικός πίνακας ήταν ο ίδιος. Αφού κάθε διεργασία είχε πίνακα όμοιο με την παραπάνω και παρακάτων διαδικασία, η γραμμή που περίμενε από την πάνω διαδικασία ήταν η δικιά της κάτω γραμμή και αντίστοιχα, η γραμμή που περίμενε από την κάτω διαδικασία ήταν η δικιά της πάνω γραμμή (αφού οι αρχικοί πίνακες ήταν ίδιοι για όλες τις διεργασίες). Έτσι το αποτέλεσμα του τελικού πίνακα κάθε διεργασίας, θα είναι το ίδιο με το να εκτελέστηκε ο ίδιος πίνακας μόνος του. Εκτυπώνω τον πίνακα μιας τυχαίας διεργασίας και συγκρίνω το αποτέλεσμα με το σειριακό πρόγραμμα τις παραπάνω παραγράφου. Το αποτέλεσμα πρέπει να είναι το ίδιο καθώς ο αρχικός πίνακας ήταν ο ίδιος (αφού δόθηκε το ίδιο seed).

Αρχικά οι δύο πίνακες δεν έβγαιναν ίδια αλλά παρόμοιοι. Υπήρχαν κάποια λάθη στον κώδικα . Όταν αυτά βρέθηκαν και διορθώθηκαν, οι δύο πίνακες που προέκυπταν ήταν πανομοιότυποι. Ακολουθεί ένα screenshot της σύγκρισης σε *online* συγκριτή κειμένου:



Εν Κατακλείδι

Βλέπουμε πως η ανάγκη που προέκυψε για αποθηκευτικό χώρο ,λύθηκε . Με την *MPI,* έγινε εφικτή όχι μόνο η επικοινωνία μεταξύ των υποσυστημάτων αλλά και ο παράλληλος υπολογισμός των δεδομένων σε κάθε υποσύστημα. Έτσι εκτός από το πρόβλημα του χώρου καταφέραμε να έχουμε και μεγάλη επιτάχυνση στον χρόνο του προγράμματος.

Μαζί με την αναφορά επισυνάπτονται : οι κώδικες των εργασιών για *two-sided* και *one-sided Communication* , και όλοι οι χρόνοι σε *.log* αρχεία για *1,2,4* διεργασίες για την *initialize, generate , play* για *two-sided* και *one-sided.*

Αναφορές

Conway's Game of Life. (n.d.). Ανάκτηση από https://en.wikipedia.org/wiki/Conway's_Game_of_Life

MPI tutorial. (n.d.). Ανάκτηση από https://computing.llnl.gov/tutorials/mpi/