ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

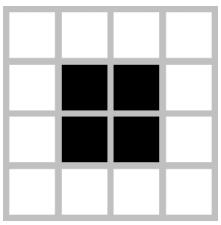
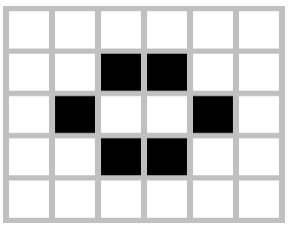
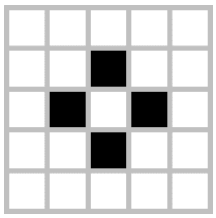
M127

ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

Fall 2019-2020
Project Report

"Game of Life"

Parallelization using MPI, Open MP and visualization using Gnuplot

FEBRUARY 23, 2020
COMPUTER SCIENCE: DATA, INFORMATION AND KNOWLEDGE MANAGEMENT
ΚΟΛΛΙΟΠΟΥΛΟΥ ΜΑΙΡΗ – CS2180021
ΓΙΑΝΝΑΚΗΣ ΣΤΑΥΡΟΣ - CS2180006

Περιεχόμενα

1. Εισαγωγή

Η παρούσα εργασία αποτελεί μέρος του μαθήματος «Μ127 - Παράλληλα Συστήματα», για το μεταπτυχιακό πρόγραμμα σπουδών «Πληροφορική» του τμήματος Πληροφορικής και Τηλεπικοινωνιών του Εθνικού και Καποδιστριακού Πανεπιστημίου Αθηνών.

Η υλοποίηση της έγινε σε περιβάλλον Windows 10, χρησιμοποιώντας το ενσωματωμένο Linux Subsystem for Windows, με distro Ubuntu 18.04. Χρησιμοποιήθηκε το Visual Studio Code για τον προγραμματισμό, οι αρχικές εκτελέσεις του κώδικα έγιναν τοπικά και έπειτα μεταφέρθηκαν στο σύστημα ΑΡΓΩ του πανεπιστημίου για τις μετρήσεις και τα τελικά σχόλια και αποτελέσματα που ακολουθούν στις επόμενες υποενότητες.

Η γλώσσα προγραμματισμού είναι C.

1.1. Παραδοτέα αρχεία

Τα αρχεία που παραδίδονται είναι τα παρακάτω:

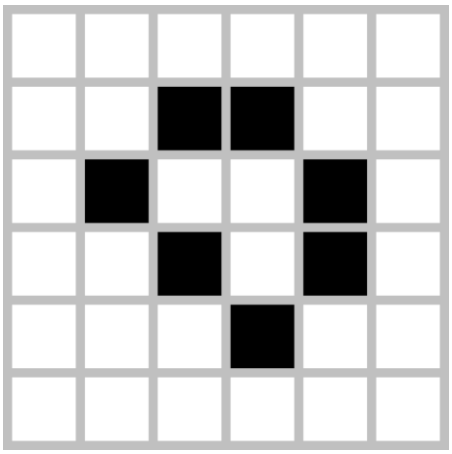
- Το παρόν report,
- Το αρχείο Stats.xlsx με τα στατιστικά και τις μετρήσεις που ακολουθούν,
- Ένας φάκελος “1.MPI” ο οποίος περιέχει τα αρχεία σχετικά με τα MPI προγράμματα που εκτελέστηκαν:
    - Έναν φάκελο “MPI_AllReduce_NonPersistent” με τα mpiP αρχεία των εκτελέσεων και τον πηγαίο κώδικα χωρίς Persistent επικοινωνία και με All Reduce.
    - Έναν φάκελο “MPI_AllReduce_Persistent” με τα mpiP αρχεία των εκτελέσεων και τον πηγαίο κώδικα με Persistent επικοινωνία και All Reduce.
    - Έναν φάκελο “MPI_NoReduce_Persistent” με τα mpiP αρχεία των εκτελέσεων και τον πηγαίο κώδικα με Persistent επικοινωνία και χωρίς All Reduce.
- Έναν φάκελο “2.Hybrid” που περιέχει:
    - Τα mpiP αρχεία των εκτελέσεων του υβριδικού OpenMP + MPI προγράμματος και τον πηγαίο κώδικα.
    - Το bash script myPBSScript.sh το οποίο περιέχει το βέλτιστο ζευγάρι διεργασιών ανά κόμβο και νημάτων.
- Έναν φάκελο “3.Gnuplot” ο οποίος περιέχει:
    - Τον κώδικα παραγωγής .GIF αρχείων “animate.p”
    - Τον πηγαίο MPI κώδικα ο οποίος παράγει τα αρχεία που χρησιμοποιούνται σαν input στο animate.p,
    - 10 .GIF αρχεία που έχουν παραχθεί από το gnuplot, με διαφορετικές διαστάσεις και γενιές.
- Τα .mpiP αρχεία των 3^{ων} καλύτερων επιδόσεων.

1.2. Περιγραφή προβλήματος

Το πρόβλημα με το οποίο ασχολείται η εργασία είναι το «Game of Life» του Άγγλου μαθηματικού John Conway. Είναι ένα κυτταρικό αυτόματο που η εξέλιξή του καθορίζεται μόνο από τις αρχικές συνθήκες. Ο παίκτης δημιουργεί την αρχική διάταξη και έπειτα, με βάση κάποιων κανόνων, η διάταξη αυτή αλλάζει και υπάρχει μια σταθερή εξέλιξη «από γενιά σε γενιά».

Συγκεκριμένα, έχουμε ένα τετραγωνικό πλέγμα NxN διαστάσεων. Κάθε κελί του πλέγματος μπορεί να έχει 2 καταστάσεις, είτε νεκρό (0 / λευκό) είτε ζωντανό (1 / μαύρο). Κάθε κελί επίσης, έχει ακριβώς 8 γείτονες οι οποίοι βρίσκονται κάθετα, οριζόντια και διαγώνια γύρω του. Η κατάσταση των γειτόνων είναι και αυτή που καθορίζει την κατάσταση του κελιού στην επόμενη γενιά. Ειδικότερα:

- Εάν ένα ζωντανό κελί έχει 0 ή 1 ζωντανούς γείτονες, τότε πεθαίνει στην επόμενη γενιά.
- Εάν ένα ζωντανό κελί έχει 2 ή 3 ζωντανούς γείτονες, τότε ζει στην επόμενη γενιά
- Εάν ένα ζωντανό κελί έχει 4 έως 8 ζωντανούς γείτονες, τότε πεθαίνει στην επόμενη γενιά.
- Εάν ένα νεκρό κελί έχει ακριβώς 3 ζωντανούς γείτονες, τότε ζωντανεύει στην επόμενο γενιά.



Εικόνα 1 - Παράδειγμα πλέγματος

Σε κάθε γενιά λοιπόν εφαρμόζονται οι κανόνες αυτοί για να δημιουργηθεί η επόμενη, μέχρι να φτάσουμε στο τέλος των γενιών προς εκτέλεση, μέχρι κάποια γενιά με την επόμενη να μην παρουσιάσει αλλαγές ή εάν έχουν όλα τα στοιχεία του πίνακα πεθάνει.

1.3. Σκοπός της εργασίας

Σκοπός της εργασίας είναι να αποδείξουμε πως με διαχωρισμό του πίνακα σε υποπίνακες και διαμοιρασμό αυτών των υποπινάκων σε πολλούς επεξεργαστές & πυρήνες μέσω διεργασιών, η εκτέλεση του προγράμματος είναι γρηγορότερη και παρουσιάζεται επιτάχυνση με την αύξηση των διεργασιών και την αύξηση του μεγέθους του προβλήματος. Δηλαδή, πως υπάρχει κλιμάκωση.

Η μελέτη γίνεται με χρήση MPI και χρήση υβριδικού προγράμματος MPI + OpenMP. Οι μετρήσεις που παρουσιάζονται βρίσκονται σε ένα .xls αρχείο στον φάκελο του παραδοτέου.

Επίσης, παρουσιάζεται μια γραφική απεικόνιση του προβλήματος που δημιουργήθηκε μέσω Gnuplot.

2. Γενικές Παρατηρήσεις

2.1. Μεθοδολογία Foster

Ο σχεδιασμός ενός παράλληλου προγράμματος από την αρχή χωρίς κάποια λογική μεθοδολογία δεν είναι εύκολος. Η χρήση μιας αποδεδειγμένης μεθοδολογίας που είναι αρκετά γενική και εύκολη αποτελεί μια καλή λύση. Μια τέτοια μεθοδολογία αποτελεί εκείνη του Foster (Μεθοδολογία Foster) που πρόκειται για μια διαδικασία σχεδιασμού τεσσάρων σταδίων. Τα τέσσερα αυτά στάδια είναι:

Partitioning: Η διαδικασία διαίρεσης των υπολογισμών και των δεδομένων σε τεμάχια.

Communication: Η διαδικασία καθορισμού του τρόπου με τον οποίο οι διεργασίες θα επικοινωνούν μεταξύ τους.

Agglomeration: Η διαδικασία ομαδοποίησης των διεργασιών για την βελτίωση της απόδοσης ή την απλούστευση του προγραμματισμού.

Mapping: Η διαδικασία εκχώρησης εργασιών σε φυσικούς επεξεργαστές

Στην παρούσα ενότητα παρουσιάζεται το στάδιο του διαμοιρασμού δεδομένων (partitioning) στα πλαίσια της εργασίας Game of life. Πιο συγκεκριμένα, αξιολογήσαμε δύο τρόπους διαμοιρασμού δεδομένων τον διαμοιρασμό σε σειρές και τον διαμοιρασμό σε blocks.

Σύμφωνα με την εκφώνηση του Game of life, κάθε κελί, δηλαδή κάθε οργανισμός, έχει 8 γείτονες. Επομένως, για τον υπολογισμό της νέας γενιάς κάθε οργανισμός πρέπει να επικοινωνήσει με τους 8 γείτονές του. Η επικοινωνία μεταξύ των οργανισμών απαιτεί χρόνο και συμβολίζεται σαν T_{comm}. Ακόμα, συμβολίζουμε με N×N τις διαστάσεις του πίνακα με *p* τους επεξεργαστές και με *r* και *s* τις σταθερές bandwidth και latency αντίστοιχα. Τις σταθερές bandwidth και latency τις

ανακτούμε χρησιμοποιώντας το mpptest. Για παράδειγμα, εκτελώντας το mpptest για επικοινωνία με 4 διεργασίες πήραμε:

- Latency: 0,226796s
- Bandwidth: 2306,023e+6

Στην περίπτωση του σειριακού προγράμματος ο υπολογισμός κάθε κελιού γίνεται σειριακά επομένως έχουμε:

$T_{serial} = t_c \times N^2$ όπου t_c ο χρόνος υπολογισμού για κάθε κελί.

<u>Διαμοιρασμός σε σειρές:</u>

Κατά τον διαμοιρασμό σε σειρές χρειάζεται να στείλουμε στους 8 γείτονες όλα τα κελιά της σειράς, δηλαδή N κελιά σε κάθε επικοινωνία. Επομένως έχουμε:

$$T_{comm} = 8 \times p(s + r \times N)$$

$$T_{parallel} = \frac{t_c \times N^2}{p} + 8 \times s + 8 \times r \times N$$

$$S = \frac{T_{serial}}{T_{parallel}} = \frac{t_c \times N^2}{\frac{t_c \times N^2}{p} + 8 \times s + 8 \times r \times N} = \frac{t_c \times N^2 \times p}{t_c \times N^2 + 8 \times p \times s + 8 \times r \times N \times p}$$

$$E = \frac{S}{P} = \frac{t_c \times N^2}{t_c \times N^2 + 8 \times p \times s + 8 \times r \times N \times p} \quad (1)$$

<u>Διαμοιρασμός σε blocks:</u>

Κατά τον διαμοιρασμό σε blocks δεν χρειάζεται να στείλουμε στους 8 γείτονες N κελιά αλλά $\frac{N}{\sqrt{p}}$ οπότε έχουμε:

$$T_{comm} = 8 \times p(s + r \times \frac{N}{\sqrt{p}})$$

$$T_{parallel} = \frac{t_c \times N^2}{p} + 8 \times s + 8 \times r \times \frac{N}{\sqrt{p}}$$

$$S = \frac{T_{serial}}{T_{parallel}} = \frac{t_c \times N^2}{\frac{t_c \times N^2}{p} + 8 \times s + 8 \times r \times \frac{N}{\sqrt{p}}} = \frac{t_c \times N^2 \times p}{t_c \times N^2 + 8 \times p \times s + 8 \times r \times \frac{N}{\sqrt{p}}}$$

$$E = \frac{T_{serial}}{T_{parallel}} = \frac{t_c \times N^2}{t_c \times N^2 + 8 \times p \times s + 8 \times r \times \frac{N}{\sqrt{p}}} \quad (2)$$

Παρατηρώντας τις σχέσεις (1) και (2) διαπιστώνουμε ότι οι αριθμητές των δυο κλασμάτων είναι οι ίδιοι. Συγκρίνοντας τους παρονομαστές παρατηρούμε ότι ο παρονομαστής της σχέσης (2) είναι μικρότερος από αυτόν της σχέσης (1) και κατ' επέκταση επιτυγχάνεται μεγαλύτερη αποδοτικότητα. Για τον λόγο αυτό, ο διαμοιρασμός σε blocks είναι προτιμότερος από τον διαμοιρασμό σε σειρές.

2.2. Βελτιώσεις

Οι βελτιώσεις που χρησιμοποιήθηκαν ήταν από αυτές που δόθηκαν στο αρχείο «Οδηγίες σχεδιασμού και ανάπτυξης GOL» που υπάρχει στο eclass του μαθήματος.

Συγκεκριμένα:

MPI πρόγραμμα

- Διαμοιρασμός σε δυο διαστάσεις (block) για την επίτευξη καλύτερης κλιμάκωσης.
- Αποστολή και λήψη ολόκληρων στηλών και σειρών για την μείωση του latency και αποθήκευσή τους τοπικά στα halo points.
- Εφαρμογή επικάλυψης επικοινωνίας με χρήση non-blocking εντολών (Isend/Irecv) για την αποστολή και λήψη στηλών και σειρών έτσι ώστε να υπολογίζονται τα εσωτερικά στοιχεία που δεν εξαρτώνται από τα halo points της διεργασίας, όσο γίνεται η non-blocking μεταβίβαση γραμμών και σειρών.
- Τοποθέτηση εντολών Irecv πριν τις εντολές Isend, ώστε να είναι έτοιμες οι διεργασίες να δεχτούν μηνύματα.
- Χρήση datatypes στα Isend/Irecv για αποστολή/λήψη σειρών και στηλών.
- Υπολογισμός ranks των 8 σταθερών γειτόνων μια φορά έξω από κεντρική επανάληψη.
- Έλεγχος της τοποθέτησης των διεργασιών με το mpiP.
- Λόγω επικοινωνίας με σταθερούς γείτονες εφαρμόστηκε Persistent Communication, MPI_Send_init, MPI_Start και MPI_Recv_init, MPI_Start για να μην επαναϋπολογίζονται οι τιμές παραμέτρων των κλήσεων Isend, Irecv.
- Χρήση δεικτών ούτως ώστε να γίνεται απόδοση της τιμής δείκτη στον πίνακα «πριν» από τον «μετά» αποφεύγοντας την αντιγραφή τιμών πίνακα με διπλό for.
- Μείωση ακολουθιακών υπολογισμών μέσα στην κεντρική επανάληψη.
- Ενσωμάτωση της υλοποίησης για την διαπίστωση μη αλλαγής τιμών ή κενού πλέγματος των πινάκων <<πριν>> και <<μετά>> στις επαναλήψεις για τον υπολογισμό των εσωτερικών και εξωτερικών στοιχείων

Υβριδικό πρόγραμμα

- Παραλληλοποίηση του υπολογισμού των νέων εσωτερικών στοιχείων που δεν εξαρτώνται από την halo με collapse(2)
- Παραλληλοποίηση του υπολογισμού των νέων εξωτερικών στοιχείων μετά την λήψη των δύο σειρών και δύο στηλών της halo.
- Χρήση static scheduling και του καλύτερου chunk_size.
- Χρήση openmp-reduce μεταξύ των νημάτων στην ίδια διεργασία και κατόπιν MPI reduce διεργασίων.

2.3. Σειριακό πρόγραμμα

Όσον αφορά το σειριακό πρόγραμμα, επειδή χρειαζόμαστε μετρήσεις χρόνων διότι θα χρειαστούμε τον χρόνο εκτέλεσης του σειριακού προγράμματος για τον υπολογισμό της επιτάχυνσης, για να αποφύγουμε να συντάξουμε το σειριακό, εκτελέσαμε το παράλληλο πρόγραμμά μας με αριθμό διεργασιών = 1. Έτσι, συμπεριφέρεται σαν σειριακό και έχουμε τον χρόνο εκτέλεσής του για τον υπολογισμό των μετρικών επιτάχυνσης και αποδοτικότητας των MPI & OpenMP.

3. MPI

3.1. Περιγραφή & Σχεδιασμός MPI

Το MPI (Message Passing Interface) είναι τρόπος παράλληλου προγραμματισμού όπου οι διεργασίες επικοινωνούν μεταξύ τους με ανταλλαγή μηνυμάτων. Όταν σταλθούν και ληφθούν τα δεδομένα των διεργασιών, τότε γίνονται οι απαραίτητες ενέργειες που απαιτεί ο αλγόριθμος.

Στην περίπτωσή μας, κάθε διεργασία παίρνει έναν υποπίνακα, τον επεξεργάζεται και επικοινωνεί με τις υπόλοιπες διεργασίες που διαχειρίζονται γειτονικά blocks για να λάβει και να στείλει δεδομένα που χρειάζεται εκείνη ή οι γείτονές της.

Αρχικό μας πρόβλημα ήταν το πως οι διεργασίες θα διαχειρίζονται ίδιο αριθμό δεδομένων. Αυτό και για να έχουμε την καλύτερη δυνατή απόδοση στο πρόγραμμά μας αλλά και για τον λόγο του ότι βοηθάει στον τρόπο προγραμματισμού. Για την εξυπηρέτησή μας, χρησιμοποιήσαμε αριθμό διεργασιών που εξυπηρετούν τις μετρήσεις μας, καθώς και διαστάσεις πινάκων που διαιρούνται ακριβώς από τις ρίζες των διεργασιών. Οι αριθμοί διεργασιών αυτοί είναι αυτοί που έχουν ακέραιες ρίζες λοιπόν: 4, 9, 16, 25, 36, 49, 64 και η κάθε πλευρά του τετραγωνικού πίνακα είναι πολλαπλάσιο του 840, για να έχουμε μετρήσεις με όλους τους συνδυασμούς πλευρών και διεργασιών.

Για την αποστολή και λήψη των δεδομένων υλοποιήθηκαν δύο διαφορετικοί τρόποι επικοινωνίας, Persistent και Non Persistent. Χρησιμοποιήθηκαν οι Non Blocking εντολές MPI_Isend / MPI_Irecv και όχι οι Blocking, διότι θα προκαλούσαν καθυστέρηση και στην συγκεκριμένη υλοποίηση δεν υπάρχει λόγος αναμονής, παρά μόνο συνολικά. Δηλαδή, πρέπει να τελειώσουν όλα την αποστολή τους και όλα τη λήψη τους για να περιμένουν, όχι να λειτουργήσουν μπλοκάροντας το ένα το άλλο. Έγινε υλοποίηση και με τους δύο τρόπους (Persistent / Non-Persistent) διότι θέλαμε να δούμε τις διαφορές μεταξύ των δύο τρόπων υλοποίησης και κατά πόσο το Persistent πρόγραμμά μας έχει καλύτερη απόδοση και επιτάχυνση από το Non-Persisistent. Δημιουργήθηκαν δύο MPI data types για τις εντολές Send και Receive, και για στήλες αλλά και για γραμμές (MPI_Datatype columnDatatype, rowDatatype).

Επίσης, κάθε διεργασία δημιουργεί 2 Contiguous πίνακες. Έναν για την τρέχουσα γενιά και έναν για την μελλοντική. Contiguous σημαίνει πως οι θέσεις μνήμης είναι συνεχόμενες, πράγμα που είναι απαραίτητο για την ανταλλαγή μηνυμάτων μεταξύ των διεργασιών μας. Οι πίνακες αυτοί έχουν 2 έξτρα στήλες και γραμμές, για τα halo points, δηλαδή τα κελία που ανήκουν σε άλλες διεργασίες αλλά είναι γειτονικά με άλλης. Για παράδειγμα, σε έναν πίνακα 9 στοιχείων, το κεντρικό στοιχείο γνωρίζει όλους του τους γείτονες. Το στοιχείο όμως που βρίσκεται στην πάνω γωνιακή σχέση, δεν τους βλέπει, διότι ανήκουν σε υποπίνακα που είναι σε άλλη διεργασία. Έτσι, για τον υπολογισμό των γειτόνων και την απόφαση του εάν στην επόμενη γενιά θα είναι νεκρό ή ζωντανό, το block αυτό θα πρέπει να επικοινωνήσει με γειτονικές διεργασίες και να λάβει την κατάσταση των γειτόνων, την οποία αποθηκεύει στα halo points.

Οι πίνακες αυτοί είναι ανάλογοι με τον αριθμό των διαστάσεων που έχουν δοθεί. Για παράδειγμα, εάν έχουμε 4 διεργασίες (δηλαδή 4 blocks) και πλευρά 20, τότε το συνολικό μέγεθος του πίνακα θα είναι 20x20 = 400. Άρα, κάθε διεργασία θα πάρει έναν τετραγωνικό υποπίνακα των 100 κελιών. Αυτό για να αποφύγουμε την καθυστέρηση της δημιουργίας στην αρχική διεργασία και τον διαμοιρασμό στις υπόλοιπες, το οποίο λόγω ανταλλαγής πολλών μηνυμάτων, ειδικά σε μεγέθη πίνακα με εκατομμύρια κελία θα επιβάρυνε πολύ την εκτέλεση του προγράμματός μας. Οι έξτρα στήλες και γραμμές δεν λαμβάνονται υπόψη μιας και είναι καθαρά για τους υπολογισμούς και δεν αποτελούν μέρος του πλέγματος, είναι «θεωρητικοί». Τα βασικά κελιά γεμίζουν με την συνάρτηση rand() και για να εξασφαλίσουμε τη διαφορετικότητα μεταξύ των αριθμών που παίρνουν, δίνουμε ως seed το time() συν τον αριθμό του rank, για να είναι πάντα διαφορετικό (εάν δεν γεμίζουν από αρχείο).

Απαραίτητη λοιπόν όπως έγινε ξεκάθαρο, είναι η εύρεση του γείτονα κάθε διεργασίας. Αποφύγαμε να δημιουργήσουμε ξεχωριστή συνάρτηση και την αφήσαμε μέσα στην κύρια διαδικασία μας λόγω του κόστους κλήσης.

Κατά την επικοινωνία μεταξύ των διεργασιών, οπού κάθε διεργασία παίρνει τις πληροφορίες που χρειάζεται, γίνεται υπολογισμός των εσωτερικών κελιών. Μετά από τον υπολογισμό αυτόν και αφού έχουν ληφθεί τα υπόλοιπα κελιά, υπολογίζονται και τα εξωτερικά.

Μετά τον υπολογισμό της νέας γενιάς, γίνεται έλεγχος τερματισμού. Όσον αφορά τη συνθήκη τερματισμού, χρησιμοποιείται All Reduce (MPI_AllReduce) για την λήψη της πληροφορίας εάν τα στοιχεία έχουν πεθάνει ή εάν δεν υπάρχει αλλαγή μεταξύ κάποιας γενιάς. Εκτελείται βέβαια το σύνολο των εντολών ακόμη και εάν υπάρξει συνθήκη τερματισμού, όπως ζητήθηκε στην εκφώνηση. Έτσι, ο δείκτης της επόμενης γενιάς δείχνει στην τωρινή, και το ανάποδο, με σκοπό να ξεκινήσει και πάλι η διαδικασία από την αρχή, είτε υπάρχει τερματισμός είτε όχι.

3.2. Εκτέλεση

Το compilation του MPI προγράμματος γίνεται με την εντολή
`mpicc -O3 -g MpiSource.c -L$MPIP_DIR/lib -lmpiP -lbfd -lunwind -lm -o MpiEx.x`
όπου MpiSource.c είναι ο πηγαίος κώδικας και MpiEx.x είναι το εκτελέσιμο.

Η εκτέλεση γίνεται με την εντολή:
`mpirun -np XX MpiEx.x N GEN filepath`
όπου XX είναι ο αριθμός των διεργασιών, N είναι η πλευρά του τετραγωνικού πίνακα, GEN είναι ο αριθμός των γενεών προς εκτέλεση και filepath είναι ένα προαιρετικό όρισμα σε περίπτωση που θέλουμε να έχουμε initial state πίνακα από αρχείο. Το αρχείο πρέπει να είναι της μορφής: 0 ή 1 ανά γραμμή. Σε περίπτωση που δεν δοθεί αρχείο ή το μονοπάτι είναι λάθος, τότε η εκτέλεση γίνεται με random state πίνακα.

3.3. Στατιστικά

Εφόσον όλοι οι πίνακες και οι διεργασίες είναι συμβατά μεγέθη και μπορούν να λειτουργήσουν όλα μεταξύ τους, δεν είχαμε περιορισμούς. Το μόνο που δημιουργήθηκε είναι πως στην περίπτωση του σειριακού προγράμματος, είχαμε Segmentation Fault κατά τη δέσμευση μνήμης για τον πίνακα των 26880 x 26800 θέσεων, πράγμα που δεν μας αφήνει να έχουμε χρόνο εκτέλεσης άρα και επιτάχυνση/αποδοτικότητα και να μελετήσουμε την κλιμάκωση. Ωστόσο κάναμε τις μετρήσεις του χρόνου για 26880 στις περιπτώσεις πολλαπλών διεργασιών για να δούμε κατά πόσο βελτιώνεται ο χρόνος μας. Όσον αφορά το MPI, παρουσιάζουμε στατιστικά για 3 διαφορετικές κατηγορίες εκτελέσεων: Χωρίς Reduce με Persistent Communication, με All Reduce και Persistent Communication και με All Reduce και Non-Persistent communication. Οι μετρήσεις έγιναν για 50 γενιές μετά από δοκιμές, μιας και δοκιμάσαμε 100 και για τα μεγέθη που τρέξαμε είχαμε αρκετές καθυστερήσεις, και λιγότερες από 50 τελείωναν αρκετά γρήγορα και δεν μπορούσαμε να έχουμε «καλά» νούμερα. Όλες οι μετρήσεις έγιναν στο μηχάνημα ΑΡΓΩ του πανεπιστημίου.

Πέρα από τους χρόνους εκτέλεσης, υπολογίζονται και:

- Η επιτάχυνση A, που είναι ο λόγος του χρόνου εκτέλεσης του σειριακού προγράμματος προς τον χρόνο εκτέλεσης του παράλληλου προγράμματος,
- Η αποδοτικότητα E του προγράμματος, που είναι ο λόγος της επιτάχυνσης A προς τον αριθμό των πυρήνων.

3.3.1. No Reduce

GENERATIONS = 50									
Μετρήσεις Χρόνου Εκτέλεσης t (s)									
Number of Processes (p)									
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	0,3677	0,0931	0,0551	0,03002	0,02815	0,02513	0,02784	0,0213
	1680 x 1680	1,4823	0,37109	0,18688	0,1052	0,07881	0,0611	0,0418	0,0358
	3360 x 3360	5,9111	1,5076	0,8215	0,448	0,2762	0,1937	0,1329	0,1059
	6720 x 6720	23,5686	5,9617	3,1633	1,8102	1,1756	0,8282	0,6049	0,4573
	13440 x 13440	94,2158	23,9035	12,4892	7,057	4,5584	3,1886	2,3612	1,8082
	26880 x 26880	SEGM	95,2306	49,7081	28,0139	17,9776	12,515	9,2307	7,0831

Εικόνα 2- Χρόνος εκτέλεσης MPI - No Reduce

GENERATIONS = 50									
Επιτάχυνση A (t/p)									
Number of Processes (p)									
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	1	3,94952	6,67332	12,2485	13,0622	14,6319	13,2076	17,2629
	1680 x 1680	1	3,99445	7,93183	14,0903	18,8085	24,2602	35,4617	41,405
	3360 x 3360	1	3,92087	7,1955	13,1944	21,4015	30,5168	44,4778	55,8178
	6720 x 6720	1	3,95334	7,45064	13,0199	20,0481	28,4576	38,9628	51,5386
	13440 x 13440	1	3,94151	7,54378	13,3507	20,6686	29,5477	39,9017	52,1047
	26880 x 26880	-	-	-	-	-	-	-	-

Εικόνα 3- Επιτάχυνση MPI - No Reduce

GENERATIONS = 50									
Αποδοτικότητα E (A/p)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N	**840 x 840**	1	0,98738	0,74148	0,76553	0,52249	0,40644	0,26954	0,26973
	1680 x 1680	1	0,99861	0,88131	0,88064	0,75234	0,6739	0,72371	0,64695
	3360 x 3360	1	0,98022	0,7995	0,82465	0,85606	0,84769	0,90771	0,87215
	6720 x 6720	1	0,98833	0,82785	0,81374	0,80193	0,79049	0,79516	0,80529
	13440 x 13440	1	0,98538	0,8382	0,83442	0,82674	0,82077	0,81432	0,81414
	26880 x 26880	-	-	-	-	-	-	-	-

Εικόνα 4- Αποδοτικότητα MPI - No Reduce

3.3.2. Persistent – All Reduce

GENERATIONS = 50									
Μετρήσεις Χρόνου Εκτέλεσης t (s)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	0,3717	0,0951	0,0678	0,0371	0,0247	0,0299	0,033	0,0308
	1680 x 1680	1,5075	0,3872	0,1905	0,1103	0,7311	0,6547	0,0476	0,04667
	3360 x 3360	6,0047	1,5333	0,8099	0,46	0,2852	0,1975	0,1429	0,1217
	6720 x 6720	23,9233	6,0765	3,1708	1,8074	1,1818	0,8318	0,6312	0,4616
	13440 x 13440	95,4883	24,1564	12,5021	7,0793	4,5678	3,1994	2,3708	1,8316
	26880 x 26880	SEGM	96,9496	49,7175	28,0225	17,9941	12,5382	9,2545	7,0953

Εικόνα 5-Χρόνος εκτέλεσης MPI - All Reduce - Persistent

GENERATIONS = 50									
Επιτάχυνση A (t(1)/t(p))									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	1	3,908517	5,482301	10,01887	15,04858	12,43144	11,26364	12,06818
	1680 x 1680	1	3,893337	7,913386	13,66727	2,061961	2,302581	31,67017	32,30126
	3360 x 3360	1	3,916194	7,414125	13,0537	21,05435	30,40354	42,02029	49,34018
	6720 x 6720	1	3,93702	7,544878	13,23631	20,2431	28,76088	37,9013	51,82691
	13440 x 13440	1	3,952919	7,637781	13,48838	20,90466	29,84569	40,27683	52,13382
	26880 x 26880	-	-	-	-	-	-	-	

Εικόνα 6 - Επιτάχυνση MPI - All Reduce – Persistent

GENERATIONS = 50									
Αποδοτικότητα E (A/p)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	1	0,977129	0,609145	0,626179	0,601943	0,345318	0,22987	0,188565
	1680 x 1680	1	0,973334	0,879265	0,854204	0,082478	0,063961	0,64633	0,504707
	3360 x 3360	1	0,979048	0,823792	0,815856	0,842174	0,844543	0,857557	0,77094
	6720 x 6720	1	0,984255	0,83832	0,827269	0,809724	0,798913	0,773496	0,809795
	13440 x 13440	1	0,98823	0,848642	0,843024	0,836186	0,829047	0,821976	0,814591
	26880 x 26880	-	-	-	-	-	-	-	-

Εικόνα 7-Αποδοτικότητα MPI - All Reduce – Persistent

3.3.3. Non-Persistent – All Reduce

GENERATIONS = 50									
Μετρήσεις Χρόνου Εκτέλεσης t (s)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	0,3726	0,0964	0,0671	0,0338	0,0306	0,03209	0,03185	0,0279
	1680 x 1680	1,5124	0,3882	0,1898	0,1124	0,08031	0,06406	0,05378	0,0442
	3360 x 3360	6,0215	1,5435	0,8156	0,4516	0,2802	0,1969	0,1473	0,1134
	6720 x 6720	23,9255	6,0835	3,167	1,7989	1,18004	0,8314	0,6137	0,4619
	13440 x 13440	95,725	24,3378	12,5882	7,0923	4,5682	3,1953	2,3833	1,8417
	26880 x 26880	SEGM	97,1521	49,69	28,0285	17,8908	12,5475	9,4532	7,1113

Εικόνα 8-Χρόνος εκτέλεσης MPI - All Reduce - Non-Persistent

GENERATIONS = 50									
Επιτάχυνση A (t/p)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	1	3,865145	5,552906	11,02367	12,17647	11,61109	11,69859	13,35484
	1680 x 1680	1	3,89593	7,968388	13,45552	18,83203	23,60912	28,12198	34,21719
	3360 x 3360	1	3,901199	7,382908	13,3337	21,49001	30,58151	40,87916	53,09965
	6720 x 6720	1	3,932851	7,554626	13,30007	20,27516	28,77736	38,98566	51,79801
	13440 x 13440	1	3,933182	7,604344	13,49703	20,95464	29,95806	40,1649	51,97643
	26880 x 26880	-	-	-	-	-	-	-	-

Εικόνα 9 - Επιτάχυνση MPI - All Reduce - Non-Persistent

GENERATIONS = 50									
Αποδοτικότητα E (A/p)									
		Number of Processes (p)							
		1	**4**	**9**	**16**	**25**	**36**	**49**	**64**
Μέγεθος Πίνακα (N)	**840 x 840**	1	0,966286	0,61699	0,688979	0,487059	0,32253	0,238747	0,208669
	1680 x 1680	1	0,973982	0,885376	0,84097	0,753281	0,655809	0,573918	0,534644
	3360 x 3360	1	0,9753	0,820323	0,833356	0,8596	0,849486	0,834269	0,829682
	6720 x 6720	1	0,983213	0,839403	0,831255	0,811006	0,799371	0,795626	0,809344
	13440 x 13440	1	0,983296	0,844927	0,843564	0,838186	0,832168	0,819692	0,812132
	26880 x 26880	-	-	-	-	-	-	-	-

Εικόνα 10-Αποδοτικότητα MPI - All Reduce - Non-Persistent

3.4. Συγκρίσεις και Παρατηρήσεις

Αρχικά, κάποιες γενικές παρατηρήσεις για όλες τις εκτελέσεις:

- Παρατηρούμε πως όσο μεγαλύτερο το μέγεθος του προβλήματος, τόσο καλύτερη και η επιτάχυνση. Σε μερικές περιπτώσεις όπως στο μέγεθος 3360 που παρουσιάζει καλύτερη επιτάχυνση και αποδοτικότητα, μπορεί να είναι και λόγω αυξημένης κίνησης στο δίκτυο κατά την εκτέλεση των μεγαλύτερων διαστάσεων. Υπάρχει πάντως μια τάση πως όσο μεγαλώνει το μέγεθος του προβλήματός μας, τόσο καλύτερες τιμές Επιτάχυνσης και Αποδοτικότητας παίρνουμε.
- Το αντίθετο συμβαίνει σε μικρά μεγέθη πινάκων. Όσο μικρότερα, όσο αυξάνεται ο αριθμός των διεργασιών, μετά από ένα σημείο παίρνουμε χειρότερα αποτελέσματα, η διαφορά επιτάχυνσης είναι μηδαμινή και η αποδοτικότητα είναι πάρα πολύ χαμηλή λόγω των πόρων που σπαταλούνται.
- Όπως αναφέραμε, για το μέγεθος πλευράς 26880 δεν έχουμε τιμές στην αποδοτικότητα και την επιτάχυνση επειδή δεν έχουμε χρόνο εκτέλεσης σειριακού προγράμματος.

Ειδικότερα, για τις ξεχωριστές εκτελέσεις:

- Όσον αφορά την εκτέλεση των Persistent προγραμμάτων με All Reduce και χωρίς All Reduce, παρατηρούμε πως επειδή δεν τερματίζει το πρόγραμμά μας όταν βρίσκει συνθήκη τερματισμού και εκτελεί όλες τις επαναλήψεις των γενεών, υπάρχει μια μικρή επιβράδυνση στο πρόγραμμα που εκτελεί και

την All Reduce εντολή, λόγω της extra επικοινωνίας που χρειάζεται μεταξύ των προγραμμάτων. Μπορεί στην συγκεκριμένη περίπτωση να μην είναι πολύ μεγάλη η διαφορά για να είναι αισθητή, αλλά σίγουρα σε περιπτώσεις μεγαλύτερων προβλημάτων με χιλιάδες επαναλήψεις, κάθε καθυστέρηση είναι σημαντική.

- Όσον αφορά την σύγκριση μεταξύ Persistent και Non-Persistent επικοινωνίας, παρατηρούμε πως οι χρόνοι της Persistent επικοινωνίας είναι καλύτεροι. Αυτό συμβαίνει λόγω της συνεχόμενος επικοινωνίας μεταξύ των διεργασιών που μειώνει τον επιπρόσθετο overhead χρόνο που υπάρχει στο Non-Persistent communication λόγω της ανάγκης για επικοινωνία κάθε φορά που στέλνεται ή λαμβάνεται κάποιο block δεδομένων. Και εδώ ισχύει το ίδιο, μπορεί οι αριθμοί να μην είναι μεγάλοι μιας και μιλάμε για δέκατα του δευτερολέπτου καλύτερης ταχύτητας εκτέλεσης, αλλά σίγουρα το ότι είναι καλύτεροι είναι σημαντικό.

4. Υβριδική παραλληλοποίηση (MPI – OpenMP)

4.1. Περιγραφή & Σχεδιασμός MPI – OpenMP

Σχετικά με την υβριδική παραλληλοποίση, χρησιμοποιήθηκε το κεντρικό μας MPI πρόγραμμα με το Persistent Communication και το All Reduce και προστέθηκαν τα απαραίτητα κομμάτια του OpenMP. Το OpenMP είναι ένα API που επιτρέπει την χρήση πολλών threads που έχουν πρόσβαση σε ένα κομμάτι κοινής μνήμης. Με την εκτέλεση του προγράμματος, το master thread δημιουργεί όσα Threads του πούμε με την μεταβλητή OMP_NUM_THREADS, τα οποία λειτουργούν σαν slave threads. Επίσης, το Master thread αναλαμβάνει να κάνει και το MPI_AllReduce, δηλαδή τον έλεγχο συνθήκης τερματισμού (εάν είναι όλα νεκρά / εάν είναι ίδιες 2 γενιές).

4.2. Εκτέλεση

Το compilation του MPI προγράμματος γίνεται με την εντολή
`mpicc -O3 -g -fopenmp Hybrid.c -L$MPIP_DIR/lib -lmpiP -lbfd -lunwind -o Hybr.x -lm`
όπου Hybrid.c είναι ο πηγαίος κώδικας και Hybr.x είναι το εκτελέσιμο.

Η εκτέλεση γίνεται με την εντολή:
`mpirun -np XX Hybr.x N GEN filepath`
όπου XX είναι ο αριθμός των διεργασιών, N είναι η πλευρά του τετραγωνικού πίνακα, GEN είναι ο αριθμός των γενεών προς εκτέλεση και filepath είναι ένα προαιρετικό όρισμα σε περίπτωση που θέλουμε να έχουμε initial state πίνακα από αρχείο. Το αρχείο πρέπει να είναι της μορφής: 0 ή 1 ανά γραμμή. Για την αλλαγή

των νημάτων του OpenMP χρησιμοποιείται η εντολή export OMP_NUM_THREADS = X όπου X ο αριθμός των Threads.

4.3. Στατιστικά

Στην περίπτωση του OpenMP έχουμε απλά στατιστικά χρόνων για κάθε κόμβο. Αρχικά, αναζητήσαμε το καλύτερο ζευγάρι νημάτων(OMP_NUM_THREADS) και διεργασιών (pps) για έναν κόμβο (n). Είχαμε τα παρακάτω αποτελέσματα για τα ζευγάρια διεργασιών ανά κόμβο και νημάτων:

Διεργασίες ανά κόμβο (ppn)	**Νήματα (OMP_NUM_THREADS)**	**Time (s)**
1	8	0.3841
2	4	0.4077
2	8	0.4134
4	2	0.0596
4	4	0.4140

Οπότε, παρατηρούμε πως το καλύτερο ζευγάρι είναι το ppn=4, OMP_NUM_THREADS=2.
Οι μετρήσεις μας λοιπόν είναι οι παρακάτω, για διαφορετικό αριθμό κόμβων με σταθερό αριθμό νημάτων και κόμβων ανά διεργασία:

GENERATIONS = 50					
Μετρήσεις Χρόνου Εκτέλεσης t (s)					
Number of Nodes (n)					
		1	**2**	**4**	**8**
Μέγεθος Πίνακα (N)	**840 x 840**	0,0596	0,055	0,2745	-
	1680 x 1680	0,2287	0,7126	0,2738	-
	3360 x 3360	1,871	1,8164	0,838	-
	6720 x 6720	6,4773	5,9769	1,4384	-
	13440 x 13440	14,0017	13,5306	7,0182	-
	26880 x 26880	61,9308	56,9732	23,9489	-

Εικόνα 11- Χρόνοι εκτέλεσης Υβριδικού OpenMP - MPI

4.4. Παρατηρήσεις

Βλέπουμε πως όσο αυξάνονται οι κόμβοι τόσο καλύτεροι είναι οι χρόνοι που επιτυγχάνονται. Πιο συγκεκριμένα, για το μέγεθος πλευράς πίνακα 6720, στους 4 κόμβους (με 4 διεργασίες ανά κόμβο και 2 νήματα, όπως αναφέραμε) επιτυγχάνεται πολύ γρηγορότερος χρόνος εκτέλεσης σε σύγκριση με τον προηγούμενο αριθμό κόμβων (n=2).

Επιπλέον, παρατηρούμε υπεροχή του υβριδικού προγράμματος σε σύγκριση με την υλοποίηση που έχει μόνο MPI καθώς με τον ίδιο αριθμό διεργασιών, στο υβριδικό έχουμε πολύ μικρότερους χρόνους εκτέλεσης εκμεταλλευόμενοι τα threads του OpenMP. Ένα παράδειγμα αποτελεί το εξής:

- Για μέγεθος πίνακα 26880 x 26880 και 16 διεργασίες:
    - MPI: Ο χρόνος εκτέλεσης είναι 28,0225 δευτερόλεπτα.
    - MPI + OpenMP: Ο χρόνος εκτέλεσης είναι 23,9489 δευτερόλεπτα.

Παρατηρούμε πως ο χρόνος στον οποίο ολοκληρώνεται το υβριδικό πρόγραμμα είναι κατά 5 δευτερόλεπτα γρηγορότερος από τον χρόνο του MPI, για τον ίδιο αριθμό διεργασιών.

Όσον αφορά τον αριθμό κόμβων n=8, δεν μπορέσαμε να παράξουμε αποτελέσματα καθώς έχοντας ως καλύτερο ζευγάρι διεργασιών ανά κόμβο και νημάτων το ppn = 4 και threads = 2, ο συνολικός αριθμός των διεργασιών για τον συνδυασμό κόμβων και διεργασιών ανά κόμβο είναι 8 x 4 =32, ο οποίος δεν έχει ακέραια ρίζα. Αυτό σημαίνει πως δεν μπορεί να χρησιμοποιηθεί στην εκτέλεση του προγράμματος μας. Δοκιμάσαμε για n=8 να αλλάξουμε τον αριθμό νημάτων και ppn για να βρούμε έναν αριθμό που βολεύει, όπως για παράδειγμα ppn = 2 και threads = 4 για να έχουμε 16 συνολικές διεργασίες, το οποίο μας επέστρεψε χειρότερα αποτελέσματα από το n=4/ppn=4/T=2 και αποφασίσαμε να μην τα συμπεριλάβουμε καν στο report.

5. Γραφική Απεικόνιση “Game of Life”

Κατ’ επέκταση της αδυναμίας μας για μελέτη κλιμάκωσης ενός CUDA προγράμματος λόγω του ότι δεν είχαμε πρόσβαση σε κάρτα γραφικών Nvidia, αποφασίσαμε να κάνουμε την επέκταση που προτάθηκε στην εκφώνηση που αφορά την γραφική απεικόνιση του αλγορίθμου. Χρησιμοποιήσαμε το Gnuplot (http://www.gnuplot.info/), το οποίο είναι ένα command line tool για την δημιουργία γραφικών παραστάσεων σε πολλά λειτουργικά συστήματα. Χρησιμοποιήσαμε την Linux έκδοση του.

5.1. Περιγραφή υλοποίησης

Κάθε εκτέλεση του προγράμματος δημιουργεί ένα GIF αρχείο με το όνομα “gol.gif”. Οι διαστάσεις του GIF είναι πάντα 1280 x 720 pixels, με τον πίνακα να βρίσκεται στο κέντρο. Πάνω από τον πίνακα υπάρχει ένδειξη της γενιάς που απεικονίζεται.

Όσον αφορά τις πιο τεχνικές πληροφορίες, δίνεται στην μεταβλητή “delay” η τιμή 15, έτσι ώστε να υπάρχει μια ελάχιστη καθυστέρηση μεταξύ των γενιών. Πειραματιστήκαμε με διαφορετικά νούμερα και πιστεύουμε πως το νούμερο αυτό δίνει μια καλή συνέχεια στην εικόνα.

Ο τρόπος με τον οποίο λειτουργεί το Script είναι ο εξής:

Μέσα στο directory που εκτελείται το script πρέπει να υπάρχουν όλα τα αρχεία των γενιών που παράγονται από το «MPIwithGenFiles.c». Τα αρχεία αυτά έχουν όνομα generationX.txt, όπου X είναι ο αριθμός της γενιάς. Η δομή των αρχείων είναι:

- Κάθε block γράφει στο αρχείο τα δικά του κελιά, ένα σε κάθε γραμμή. 0 εάν είναι νεκρό, 1 εάν είναι ζωντανό.
- Το αρχείο γράφεται λοιπόν με σειρά blocks, ανάλογα σε πόσα έχει χωριστεί από την εκτέλεση του αρχείου.

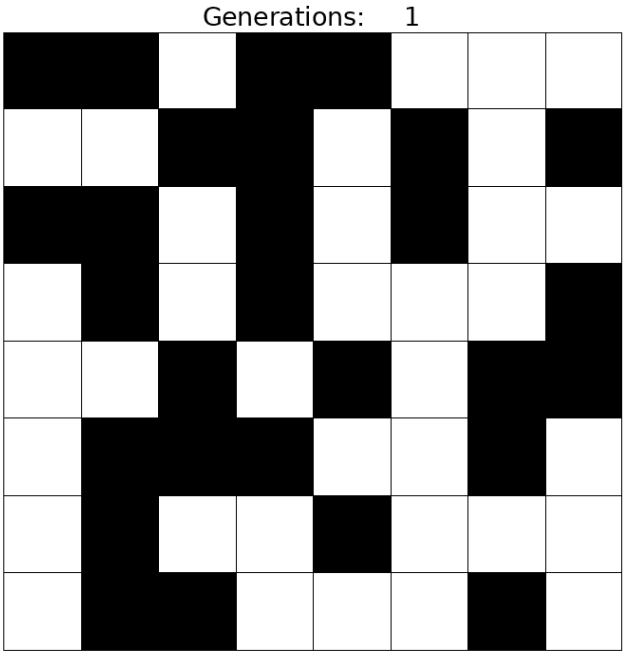
Έπειτα, παίρνουμε το αρχικό generations1.txt και διαβάσουμε με την εντολή STATS τα στατιστικά του, δηλαδή τον αριθμό των γραμμών του αρχείου. Παίρνοντας την ρίζα αυτού, έχουμε τις πλευρές του συνολικού πίνακα. Ο χρήστης δίνει σαν input τον αριθμό των Blocks κατά την εκτέλεση του script (υποενότητα 5.2) για να γνωρίζουμε το block size.

Διαβάζουμε τον αριθμό των αρχείων με την εντολή:

gens = system("find . -name '*.txt' | wc -l"), η οποία εκτελεί την εντολή find του bash για να βρει τα αρχεία με κατάληξη .txt. Έπειτα, αυτά γίνονται pipe στην «wc -l» η οποία μετράει τον αριθμό των αρχείων, που εν τέλει είναι το index number της κεντρικής μας for loop (generations loop).

Γεμίζουμε έναν πίνακα με τιμές που διαβάζουμε από το αρχείο. Επειδή το αρχείο είναι γραμμένο ανα block όπως αναφέρθηκε και εμείς γεμίζουμε τον πίνακα σειριακά, ελέγχουμε τις τιμές που διαβάζουμε για να είναι οι σωστές με την παρακάτω μέθοδο:

- Loop ανά γραμμή (1)
- Loop ανά Block (2)
- Loop ανά στοιχείο του block. (3)
    - Εαν φτάσουμε στο τελευταίο στοιχείο της γραμμής του block, συνεχίζουμε στο επόμενο block.
- Διαβάζουμε τη γραμμή του αρχείου που αντιστοιχεί στο πεδίο αυτό, την οποία:
    - Τη βρίσκουμε με τον δείκτη: m = idx + bl*int(cells/blocks) + j
        - M: ο δείκτης
        - Idx: Δείκτης που ελέγχει εάν η γραμμή μας είναι η αρχή νέου block, ελέγχεται με το εάν ο μετρητής του loop ανά γραμμή (1) διαιρείται ακέραια με τον αριθμό των γραμμών των blocks.
        - Bl: μετρητής της επανάληψης των blocks. (2)
        - J: αριθμός στοιχείου block στη συγκεκριμένη γραμμή.
    - Με τον παραπάνω δείκτη, διαβάζουμε την συγκεκριμένη γραμμή αρχείου με την εντολή "cat generation".g.".txt | awk ''NR==". m ."''" όπου:
        - Cat: η bash εντολή προβολής αρχείου στο τερματικό.
        - g: ο αριθμός του αρχείου γενιάς.
        - Το παραπάνω cat γίνεται pipe στην εντολή awk, όπου για αριθμό γραμμής m, μας επιστρέφει την τιμή του συγκεκριμένου κελιού (0,1)
- Τοποθετούμε την τιμή αυτή στον πίνακα, και ελέγχουμε εάν είναι 0 ή 1.
    - Εάν είναι 0 δημιουργούμε στις θέσεις i, i-1, col, col-1 (όπου col ο αριθμός της στήλης) ένα λευκό κουτί.
    - Εάν είναι 1 δημιουργούμε στις θέσεις i, i-1, col, col-1 (όπου col ο αριθμός της στήλης) ένα μαύρο κουτί.
- Έπειτα, με το πέρας των επαναλήψεων, σχεδιάζουμε τον πίνακα και προχωράμε στην επόμενη γενιά.



Εικόνα 12- Πλέγμα 8x8

5.2. Εγκατάσταση και εκτέλεση

Για την εγκατάσταση του Gnuplot σε Linux περιβάλλον, εκτελείται η εντολή:

```
apt-get install gnuplot.
```

Μετά την εγκατάσταση της βιβλιοθήκης, για την εκτέλεση του script “animate.p” απαιτείται η παρακάτω εντολή:

```
gnuplot -e “blocks=x” animate.p
```

Το -e δηλώνει πως ακολουθεί όρισμα. Στο blocks = x, θα πρέπει στην θέση του X να τοποθετηθεί ο αριθμός των blocks στα οποία έχει χωριστεί το αρχείο κατά την εκτέλεση του MPIwithGenFiles.c.

Παραδείγματα εκτέλεσης υπάρχουν στον παραδοτέο φάκελο για διαφορετικά μεγέθη πινάκων και γενεών.