Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών
Τομέας Ηλεκτρονικής και Υπολογιστών
Α.Π.Θ.

5/1/2017

Παράλληλα και Κατανεμημένα Συστήματα Υπολογιστών

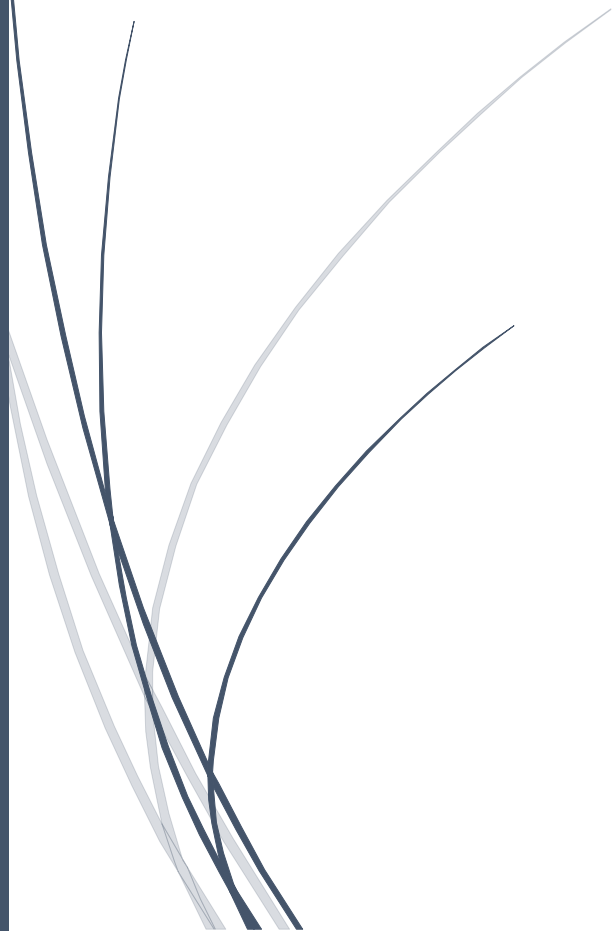
Εργασία 2

Διαμαντή Μαρία 8133

mfdiamanti@ece.auth.gr

Ντζιώνη Δήμητρα 8209

dntzioni@ece.auth.gr



Πίνακας περιεχομένων

1. Εισαγωγή

Η δεύτερη εργασία στο μάθημα των Παράλληλων και Κατανεμημένων Συστημάτων Υπολογιστών αφορά τη διάσπαση ενός προβλήματος σε επιμέρους υποπροβλήματα και την ανάθεση αυτών σε διαφορετικές διεργασίες, προκειμένου να επιλυθεί αυτό στο σύνολό του. Για να επιτευχθεί, όμως, αυτό απαραίτητη είναι η επικοινωνία μεταξύ των διεργασιών και συνεπώς, η χρήση προχωρημένων εντολών MPI-3. Όλα τα παραπάνω εφαρμόζονται στην υλοποίηση του κυψελικού αυτόματου Game of Life, του οποίου ο σειριακός κώδικας μας δίνεται.

Το παιχνίδι Game of Life παίζεται σε ένα ορθογώνιο πλέγμα, όπου το κάθε κελί μπορεί να έχει δύο καταστάσεις, ζωντανό και μη ζωντανό. Οι καταστάσεις των κελιών μεταβάλλονται σε κάθε επανάληψη του παιχνιδιού (γενιά) ταυτόχρονα για όλα τα κελιά. Η εξέλιξη του παιχνιδιού εξαρτάται μόνο από τις αρχικές συνθήκες και από τους τρεις παρακάτω κανόνες:

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

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

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

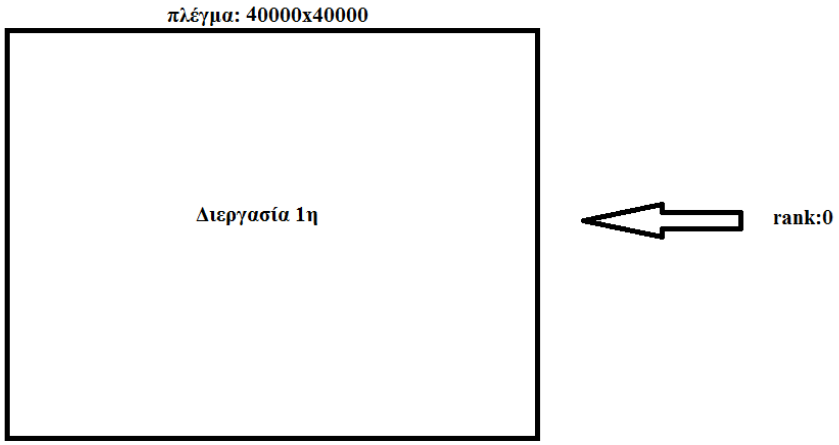
Για τον υπολογισμό των γειτόνων χρησιμοποιούνται κυκλικές οριακές συνθήκες, δηλαδή το κελί που ακολουθεί το n-1 είναι το 0 και αντίστοιχα το προηγούμενο του 0 είναι το n-1.

Στόχος μας είναι λοιπόν, η διάσπαση του αρχικού πλέγματος σε πλέγματα μικρότερων διαστάσεων, για καθένα από τα οποία θα επαρκεί η μνήμη ενός υπολογιστή. Στην εργασία αυτή θα ασχοληθούμε με αρχικά μεγέθη πλεγμάτων 40000*40000, 80000*40000 και 80000*80000, τα οποία θα διασπαστούν κατάλληλα για να αποδοθούν σε μία, δύο και τέσσερις διεργασίες αντίστοιχα. Επιπλέον, σε καθένα υποπρόβλημα θα χρησιμοποιηθεί υλοποίηση σε OpenMP για την παραλληλοποίηση του σειριακού κώδικα. Ακολουθεί αναφορά με την περιγραφή της μεθόδου παραλληλισμού, της επικοινωνίας μεταξύ των διεργασιών, αλλά και τα αποτελέσματα που προέκυψαν από την εκτέλεση του κώδικα στην υπολογιστική συστοιχία του ΑΠΘ.

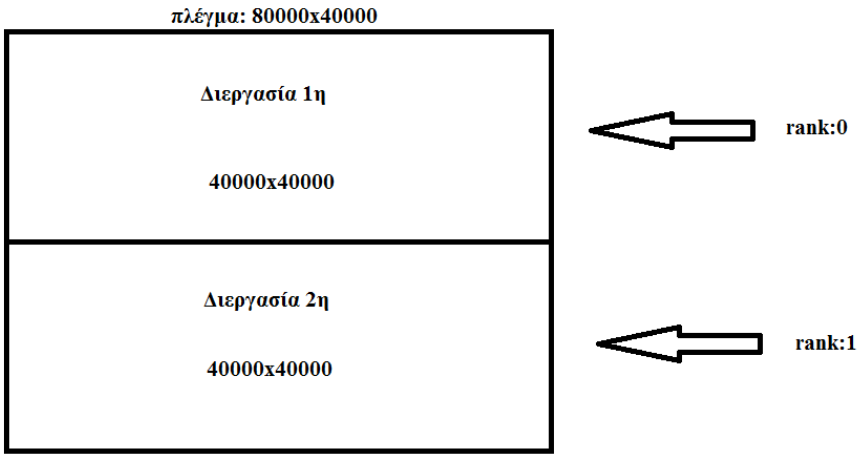
2. MPI

2.1. Καθορισμός μεγέθους πλέγματος διεργασιών

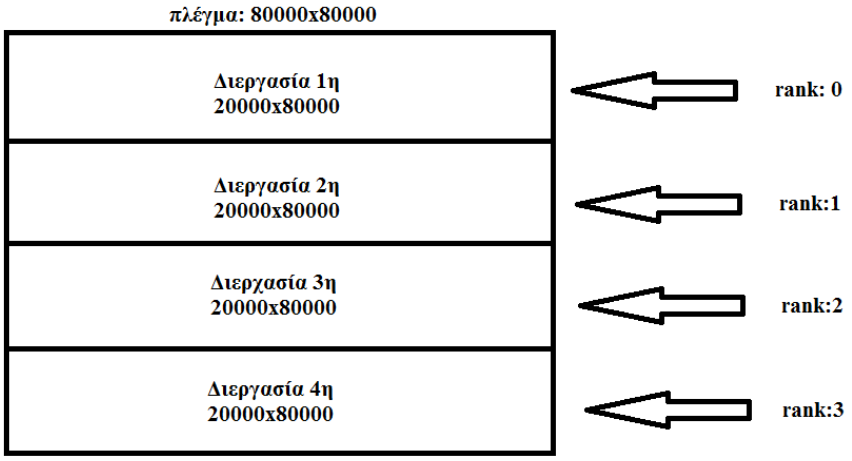
Αρχικά, απαραίτητη για την έναρξη του παιχνιδιού είναι η δημιουργία του πλέγματος. Οι διαστάσεις του πλέγματος εισάγονται ως ορίσματα της συνάρτησης main() από το χρήστη. Επειδή το πλέγμα μπορεί να είναι ορθογώνιο χρησιμοποιούνται οι μεταβλητές M, N για να οριστεί σωστά το μέγεθός του (M*N), προσθέτοντας έτσι ένα έβδομο όρισμα στη συνάρτηση main(). Οι διαστάσεις που μπορεί να έχει το πλέγμα, όπως δίνονται από την εκφώνηση, είναι 40000*40000, 80000*40000 και 80000*80000. Με βάση τις διαστάσεις αυτές ζητήθηκε να διασπαστεί το αρχικό πλέγμα και να ανατεθεί κάθε τμήμα του σε διαφορετικές διεργασίες. Ο τρόπος που επιλέχθηκε για το σκοπό αυτό φαίνεται παρακάτω.



Εικόνα 1. Μέγεθος πλέγματος σε μία διεργασία



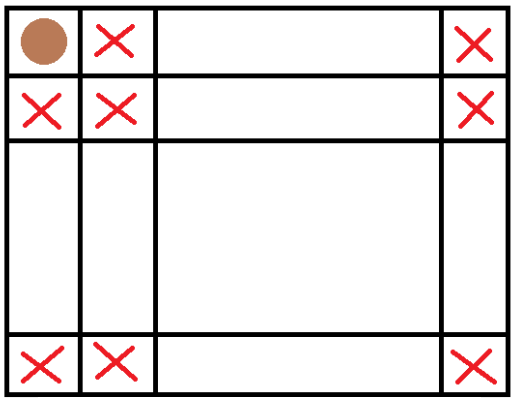
Εικόνα 2. Μέγεθος πλέγματος σε δύο διεργασίες



Εικόνα 3. Μέγεθος πλέγματος σε τέσσερις διεργασίες

Είναι εύκολα κατανοητό ότι ανάλογα με το πλήθος των διεργασιών εκτελείται διαφορετικός κώδικας.

2.2. Επικοινωνία διεργασιών



Εικόνα 4. Γείτονες σημείου

Όπως προαναφέρθηκε, για την εξέλιξη παιχνιδιού κάθε σημείο του πλέγματος πρέπει να ελέγχει τους οκτώ γείτονές του, γεγονός που σημαίνει ότι πρέπει να έχει πρόσβαση στις γραμμές που βρίσκονται πάνω και κάτω από αυτό, πέραν από τη δική του. Στην αριστερή εικόνα φαίνονται οι γείτονες του στοιχείου [0,0] του πλέγματος εφαρμόζοντας κυκλικές οριακές συνθήκες, πράγμα που καταδεικνύει την απαίτηση αυτή.

Ωστόσο, λόγω της διάσπασης του πλέγματος και της ανάθεσης των τμημάτων του σε διαφορετικές διεργασίες, αλλά και των κυκλικών οριακών συνθηκών, τα σημεία της πρώτης και τελευταίας γραμμής κάθε διεργασίας απαιτούν δεδομένα που περιλαμβάνονται σε άλλες διεργασίες. Όσον αφορά, λοιπόν, την επικοινωνία μεταξύ των διεργασιών, χρησιμοποιούμε προχωρημένες εντολές MPI-3, προκειμένου να στέλνει και να λαμβάνει κάθε μία από τις υπόλοιπες τα απαραίτητα δεδομένα. Πιο συγκεκριμένα, μέσω των εντολών:

```
MPI_Init(&argc,&argv);
```

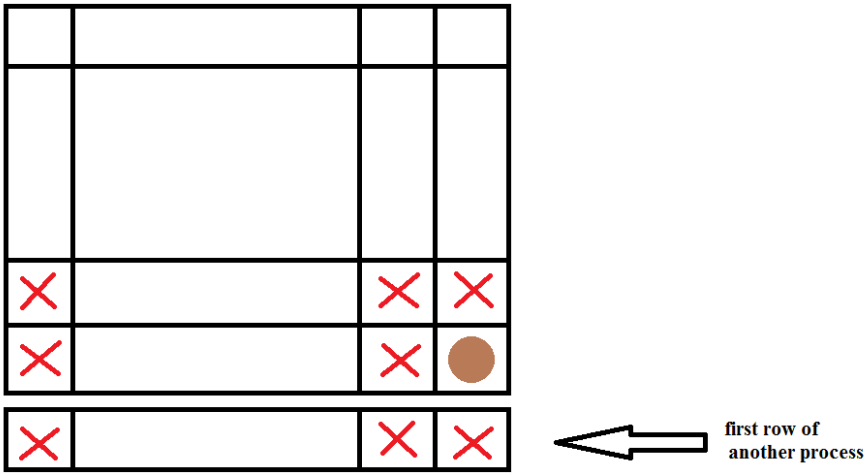
```
MPI_Finalize();
```

ορίζεται η περιοχή αποστολής και λήψης μηνυμάτων.

Επιπρόσθετα, για να ορισθεί το πλήθος των διεργασιών που θα επικοινωνούν και η ταυτότητα της καθεμίας, χρησιμοποιούνται οι εντολές:

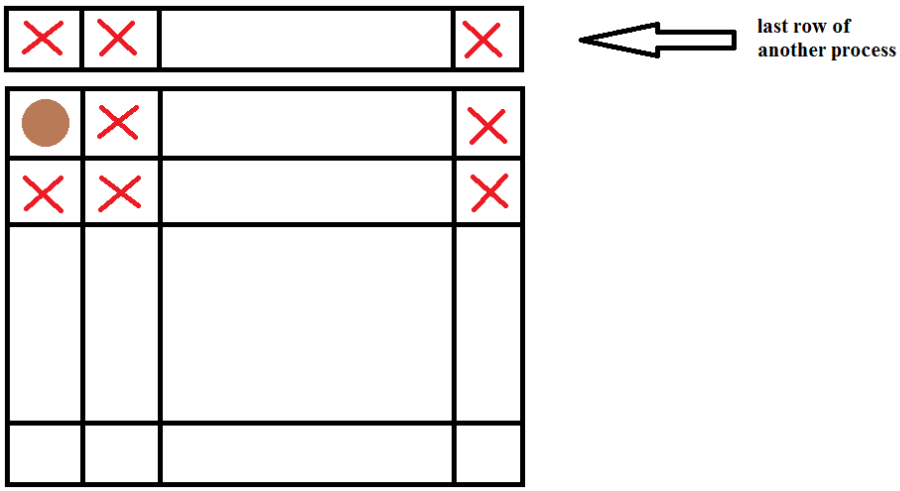
```
MPI_Comm_size(MPI_COMM_WORLD,&numtasks);

MPI_Comm_rank(MPI_COMM_WORLD,&rank);
```

Για την αποστολή και τη λήψη μηνυμάτων επιλέχθηκε να χρησιμοποιηθούν οι ασύγχρονες εντολές MPI_Isend και MPI_Irecv. Οι εντολές αυτές έχουν το πλεονέκτημα ότι δεν καθυστερούν την εκτέλεση του προγράμματος περιμένοντας να πραγματοποιηθεί η ζητούμενη επικοινωνία, αλλά συνεχίζεται κανονικά η ροή εκτέλεσής του. Παρόλα αυτά, προτού χρησιμοποιηθούν τα δεδομένα που προκύπτουν από την επικοινωνία μεταξύ διεργασιών, είναι υποχρεωτικό να ελεγχθεί αν αυτή έχει ολοκληρωθεί. Αν όχι, τότε το πρόγραμμα αναμένει έως ότου πραγματοποιηθούν οι απαραίτητες αποστολές και λήψεις των δεδομένων. Ο έλεγχος αυτός γίνεται μέσω της εντολής MPI_Testall και η αναμονή μέσω της MPI_Waitall. Για την επίτευξη τόσο του ελέγχου όσο και της αναμονής, δεσμεύονται οι πίνακες MPI_Request reqs[4], MPI_Status stats[4]. Το μέγεθος των πινάκων αυτών καθορίζεται από το πλήθος των αποστολών και λήψεων της κάθε διεργασίας.Πιο αναλυτικά, στο αρχείο main.c, έπειτα από την εισαγωγή των διαστάσεων του αρχικού πλέγματος, δεσμεύονται για καθεμία διεργασία δύο πίνακες (board, newboard) που αντιστοιχούν στην τωρινή και την αμέσως επόμενη γενιά. Έτσι, για την περίπτωση που αρκεί μία μόνο διεργασία (40000*40000), εφόσον δε χρειάζεται να πραγματοποιηθεί κάποια επικοινωνία, εκτελείται ο κώδικας που δόθηκε με ορισμένες διαφοροποιήσεις για τη βελτίωση της ταχύτητάς του, που θα αναφερθούν στη συνέχεια. Στην περίπτωση των δύο διεργασιών (80000*40000) αλλά και των τεσσάρων (80000*80000), δεσμεύονται για καθεμία διεργασία δύο επιπλέον πίνακες (first_row, last_row). Κάθε διεργασία στέλνει την πρώτη και τελευταία γραμμή της στις διεργασίες που τις χρειάζονται, οι οποίες αποθηκεύονται στους πίνακες first_row και last_row των παραληπτών. Με αυτόν τον τρόπο, ο πίνακας first_row περιέχει τα δεδομένα που απαιτούνται για τον υπολογισμό των ζωντανών γειτόνων της τελευταίας γραμμής της διεργασίας.



Εικόνα 5. Έλεγχος κατάστασης γειτόνων από την πρώτη γραμμή άλλης διεργασίας.

Αντίστοιχα, ο πίνακας last_row περιέχει τα δεδομένα που απαιτούνται για τον υπολογισμό των ζωντανών γειτόνων της πρώτης γραμμής της διεργασίας.



Εικόνα 6. Έλεγχος κατάστασης γειτόνων από την τελευταία γραμμή άλλης διεργασίας.

2.3. Υπολογισμός ζωντανών γειτόνων

Για να παιχτεί το παιχνίδι για μια γενιά καλείται η συνάρτηση play(), η οποία διαφοροποιήθηκε για μία διεργασία και για δύο ή τέσσερις διεργασίες. Έτσι, στην περίπτωση που ορίζεται μία διεργασία, τότε καλείται η play() που δόθηκε αρχικά, ενώ για δύο ή τέσσερις καλείται η play_processes(), η οποία υλοποιήθηκε στα πλαίσια της εργασίας. Οι συναρτήσεις αυτές έχουν παραλληλοποιηθεί με χρήση της OpenMP, ωστόσο, ο τρόπος παραλληλοποίησής τους θα αναφερθεί στη συνέχεια. Η διαφορά των δύο αυτών συναρτήσεων έγκειται στο εξής γεγονός. Η play() καλεί τη συνάρτηση adjacent_to() για όλα τα σημεία του πλέγματος σε ένα βρόχο επανάληψης, υπολογίζοντας το άθροισμα των ζωντανών τους γειτόνων. Αντίθετα, η συνάρτηση play_processes () υπολογίζει, αρχικά, το άθροισμα των ζωντανών γειτόνων των σημείων του πλέγματος σε ένα βρόχο επανάληψης, χωρίς να συμπεριληφθούν η πρώτη και τελευταία γραμμή αυτού. Το άθροισμα των ζωντανών γειτόνων των σημείων της πρώτης και τελευταίας γραμμής υπολογίζονται ξεχωριστά στη συνέχεια της ίδιας συνάρτησης. Οι συναρτήσεις που καλούνται από την play_processes () είναι οι adjacent_to_less() και adjacent_to_row(), που δημιουργήθηκαν έπειτα, για τις ανάγκες της εργασίας.

Παρατηρήσαμε πως στην play_processes() όταν υπολογίζονται οι γείτονες για τα σημεία του πλέγματος δίχως την πρώτη και τελευταία γραμμή, δεν υπάρχει λόγος να ελέγχεται μέσω της xadd() το ενδεχόμενο να συμβεί i < 0 ή i >= N. Σε αυτό μόνο το σημείο διαφοροποιείται η δοθείσα συνάρτηση adjacent_to() από την adjacent_to_less (), αποφεύγοντας μερικά δις ελέγχων, εξ ‘ου και το όνομα της .

Όσον αφορά την adjacent_to_row(), πρόκειται για μια τροποποιημένη έκδοση της adjacent_to_less(), έτσι ώστε να επεξεργάζονται και τα δεδομένα (οι γείτονες) που λαμβάνονται από άλλη διεργασία. Αξίζει να αναφερθούν σε αυτό το σημείο τα ορίσματα της adjacent_to_row() τα οποία εκτός από αυτά της adjacent_to_less(), είναι τα εξής:

Πρόκειται για έναν πίνακα row στον οποίο περνιέται κάθε φορά είτε ο πίνακας first_row είτε ο last_row ανάλογα με τη σειρά στην οποία βρίσκεται το πρόγραμμα και τους γείτονες των σημείων αυτής τους οποίους επιθυμεί να υπολογίσει. Επιπλέον ως ορίσματα περνιούνται δύο μεταβλητές start και end οι οποίες θα προσδιορίσουν τις τιμές του k που βρίσκεται στον εξωτερικό βρόχο for.

3. OpenMP

Σύμφωνα με το ζητούμενο της εκφώνησης, ο κώδικας πρέπει να παραλληλοποιείται στο τμήμα στο οποίο υλοποιείται η εξέλιξη του παιχνιδιού για μία γενιά. Τα πιθανά αυτά σημεία στον κώδικα είναι δύο, οι συναρτήσεις generate() και play(). Παρόλα αυτά η παραλληλοποίηση της generate() είναι αδύνατη καθώς χρησιμοποιείται η συνάρτηση srand(), η οποία με τη σειρά της δεν υλοποιείται με ορθό τρόπο όταν εκτελείται από πολλά νήματα. Με βάση το παραπάνω παραλληλοποιήθηκε μόνο η συνάρτηση play() και play_processes() αντίστοιχα. Συγκεκριμένα, παραλληλοποιήθηκαν στην play() η εξωτερική των εμφωλευμένων for και στην play_processes() τόσο η εξωτερική των εμφωλευμένων for, όσο και η for που έπεται. Όσον αφορά τις εμφωλευμένες for, προκειμένου να γίνει με ορθό τρόπο η παραλληλοποίηση τροποποιήθηκε ο κώδικας ώστε να ορίζεται για κάθε νήμα εκ νέου η μεταβλητή j του εσωτερικού βρόχου for. Με το ίδιο σκεπτικό, η μεταβλητή που περιέχει το σύνολο των ζωντανών γειτόνων του κάθε σημείου δηλώνεται μέσα στο βρόχο, ώστε να αποφευχθεί πιθανή παράλληλη εγγραφή από πολλά νήματα στην ίδια θέση μνήμης. Οι παράμετροι που χρησιμοποιήθηκαν στην εντολή #pragma omp for είναι οι εξής: schedule(static), καθώς το έργο των νημάτων θα ισομοιραστεί, και nowait επειδή δε χρειάζεται κάποιο νήμα να περιμένει τα υπόλοιπα. Οι ίδιες παράμετροι χρησιμοποιήθηκαν και στην παραλληλοποίηση της επιπλέον for που υπάρχει στη συνάρτηση play_processes().

4. Σύγκριση χρόνων εκτέλεσης

Αρχικά, προκειμένου να μετρηθεί ο συνολικός χρόνος εκτέλεσης του προγράμματος, συμπεριελήφθη το header "sys/time.h" και αφού αρχικοποιήθηκαν οι απαραίτητες μεταβλητές, τοποθετήθηκε η συνάρτηση gettimeofday () στην αρχή και το τέλος της main(). Επιπρόσθετα, δόθηκε πρόσβαση στην υπολογιστική συστοιχία του ΑΠΘ, ώστε να υπάρχει η δυνατότητα τόσο της δέσμευσης μνήμης που απαιτήθηκε από την εκφώνηση, όσο και η ταχύτερη εκτέλεση του προγράμματος.

Ωστόσο, προκειμένου να εκτελεστεί το πρόγραμμα, κύριο μέλημα ήταν η συγγραφή ενός script το οποίο περιέχει απαραίτητες πληροφορίες όπως ο αριθμός των υπολογιστών και των πυρήνων που είναι επιθυμητό να δεσμευτούν, όπως επίσης το πλήθος των διεργασιών ανά υπολογιστή και οι τιμές που αντιστοιχούν στα ορίσματα της main(). Αναλυτικά, για κάθε μία διεργασία δεσμεύονται τόσα nodes όσο και το πλήθος των διεργασιών, αναθέτοντας μία διεργασία στο κάθε ένα, 8 ppn, 50% πιθανότητα ζωντανών σημείων στο αρχικό πλέγμα , 3 γενιές και 8 νήματα.

Στη συνέχεια, παρουσιάζονται οι χρόνοι εκτέλεσης των διεργασιών, σε δύο διαφορετικές χρονικές στιγμές.

Πλήθος διεργασιών	Rank 0	Rank 1	Rank 2	Rank 3
1	74,441224	-	-	-
2	54,309054	54,015266	-	-
4	54,281145	53,997785	54,017088	54,019728

Πίνακας 1. Χρόνοι Εκτέλεσης Διεργασιών(s)

Πλήθος διεργασιών	Rank 0	Rank 1	Rank 2	Rank 3
1	77,640714	-	-	-
2	56,190787	55,915859	-	-
4	56,313915	56,058539	55,937301	56,085065

Πίνακας 2. Χρόνοι Εκτέλεσης Διεργασιών(s)

Όπως είναι φανερό, η παραλληλοποίηση με τη χρήση εντολών MPI για περιπτώσεις που το πλέγμα είναι αρκετά μεγαλύτερο από το μέγεθος που μπορεί να εκτελέσει μια διεργασία, οδηγεί σε αποτελέσματα σημαντικά καλύτερα από την περίπτωση που μια διεργασία αρκεί για να αναλάβει την εκτέλεση ολοκλήρου του πλέγματος. Το αποτέλεσμα αυτό είναι αναμενόμενο καθώς όπως προαναφέρθηκε ο κώδικας που εκτελείται από πολλαπλές διεργασίες έχει βελτιωθεί με τέτοιο τρόπο, ώστε να αποφεύγονται περιττοί έλεγχοι κι επαναλήψεις. Επιπρόσθετα, μέσω της χρήσης των ασύγχρονων εντολών Isend και Irecv, αποφεύγεται τυχόν καθυστέρηση της επικοινωνίας μεταξύ διεργασιών εκτελώντας ταυτόχρονα κάποιο άλλο τμήμα κώδικα.

Τέλος, το συμπέρασμα στο οποίο μπορεί εύκολα να καταλήξει κανείς είναι πως μέσω της χρήσης MPI εντολών δίνεται η δυνατότητα να εκτελεστούν προβλήματα για τα οποία η μνήμη ενός υπολογιστή μόνο δεν αρκεί. Μάλιστα, παρόλο που υλοποιείται επικοινωνία μεταξύ διαφορετικών υπολογιστών, αυτή είναι ταχύτατη δίνοντας εξαιρετικά αποτελέσματα.

5. Αναφορές

Παρακάτω αναγράφονται ενδεικτικά ορισμένοι ιστότοποι από όπου συλλέχθηκαν ορισμένα στοιχεία για την διεκπεραίωση της εργασίας αυτής.

http://mpitutorial.com/tutorials/introduction-to-groups-and-communicators/

http://forum.openmp.org/forum/viewtopic.php?f=3&t=1643

http://www.shodor.org/petascale/materials/distributedMemory/code/Life_mpi/

https://docs.oracle.com/cd/E19708-01/821-1319-10/ExecutingPrograms.html

https://computing.llnl.gov/tutorials/mpi/

https://computing.llnl.gov/tutorials/openMP/

https://www.open-mpi.org/doc/v1.6/man1/mpirun.1.php