Παράλληλα & Κατανεμημένα συστήματα Υπολογιστών

Εργασία 2

John Conway's Game of Life

προσομοίωση του παιχνιδιού με συνδυασμό μεθόδων:

OpenMP & MPI

Κασπαρίδης Παναγιώτης, ΑΕΜ:8157

Περιγραφή μεθόδων Παραλληλισμού

Περιγραφή του παιχνιδιού: Το παιχνίδι δημιουργεί ένα ταμπλό (Board) το οποίο γεμίζει με 0 ή 1. Παίρνει τιμές ανάλογα με μια πιθανότητα που έχουμε θέσει (propability of alive cell) μέσα από μια γεννήτρια ψευδό-τυχαίων αριθμών. Έπειτα καλείται η play και δημιουργείται το νέο ταμπλό, η διαδικασία με την οποία γίνεται αυτό περιγράφεται στην εκφώνηση της άσκησης. Η διαδικασία αυτή επαναλαμβάνεται για όσα generations έχουμε ορίσει στην αρχή της εκτέλεσής μας.

Υλοποίηση (MPI)

Για την συγκεκριμένη άσκηση η δυσκολία έγκειται στο γεγονός ότι δεν έχουμε τόση μνήμη για να αποθηκεύσουμε τον πίνακά μας. Οι διαστάσεις που ζητούνταν είναι 40000x40000 για μια διεργασία (MPI process), 80000x40000 για δυο και 80000x80000 για 4. Οπότε η λογική που ακολούθησα ήταν να χωρίσω το ταμπλό και να το κάνω generate ξεχωριστά σε κάθε process αφού συμπληρώνεται από τυχαίους αριθμούς και δεν επηρεάζεται από τίποτα άλλο.

Makefile : Στο Makefile άλλαξα τον compiler σε mpicc και έβαλα και flag για την OpenMP -fopenmp

Running on Grid : Για την εκτέλεση του προγράμματος έτρεξα λίγο αλλαγμένο το example που υπήρχε ανεβασμένο στο ethmmy. Δήλωνα πάντα ppn = 8 για να δεσμεύω 8 πυρήνες από το σύστημα και στην εκτέλεση έτρεχα με mpiexec -np $PBS_NUM_NODES -ppn 1 όπου $PBS_NUM_NODES γυρνάει τον αριθμό των nodes που έχω θέσει και το flag -ppn δηλώνει πόσα process να δημιουργηθούν ανά node.
Χωρισμός του ταμπλό στην MPI : Ο χωρισμός του ταμπλό με την χρήση του mpi γίνεται ως εξής :

Αρχικά προσθέτω άλλο ένα όρισμα στον κώδικα, το M, που είναι ο αριθμός των στηλών. Η άσκηση ήταν δοσμένη για NxN πίνακα. N είναι ο αριθμός των γραμμών. Η MPI_Comm_size επιστρέφει στην nproc τον συνολικό αριθμό των διεργασιών MPI που έχουν δημιουργηθεί. Ο χωρισμός του πίνακα ουσιαστικά γίνεται απλά με το “σπάσιμο” των γραμμών (N=N/nproc). Έτσι σε κάθε διεργασία θα δημιουργηθεί ένας πίνακας (N/nproc)xM ο συνολικός οπότε πίνακας θα είναι NxM.

```
rc=MPI_Init(&argc,&argv);
 if (rc != MPI_SUCCESS) {
  printf("Error starting MPI program.\nTerminating.\n");
  MPI_Abort(MPI_COMM_WORLD, rc);
 }
 MPI_Comm_rank(MPI_COMM_WORLD,&rank);
 MPI_Comm_size(MPI_COMM_WORLD,&nproc);
 MPI_Get_processor_name(name,&len);
 printf("Hi from process :%d from processor : %s\n",rank,name);
 omp_set_num_threads(NTHREADS);
//Input command line arguments
 int M=atoi(argv[5]);
 int N = atoi(argv[1]);       // Array size
 double thres = atof(argv[2]); // Propability of life cell
 int t = atoi(argv[3]);       // Number of generations
 int disp = atoi(argv[4]);    // Display output?
 N=N/nproc;
```

Μέγεθος Πινάκων: Μετά από αποτυχημένες προσπάθειες να τρέξω για 1, 2 και 4 processes για τα μεγέθη που ζητούσε η άσκηση αποφάσισα να μειώσω το μέγεθος των πινάκων καθώς πιστεύω ότι ήταν πολύ μεγάλοι και δεν χωρούσαν. Για παράδειγμα για 40000x40000 έχουμε δύο τέτοιους πίνακες board & newboard συνολικά σαν integer θέλουν κοντά στα 12 gb ram. Ωστόσο φορτώνονται με τιμές ή 0 ή 1.Ετσι δεσμεύεται αχρησιμοποίητος χώρος. Άλλαξα όλους του πίνακες σε int8_t

δηλαδή 1 byte integer ο οποίος μειώνει δραματικά τον χώρο που χρειάζομαι για αποθήκευση των δεδομένων.

Έπειτα υπήρχαν δύο συναρτήσεις για την αρχικοποίηση του πίνακα, οι initialize_board και generate_board. Η πρώτη ήταν περιττή και την έσβησα. Στο seed της generate_board βάζω επιπλέον το rank του process.

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
srand((rank+1)*time(NULL));
```

- ***Συνάρτηση Play:***

  Σε αυτήν την συνάρτηση γίνονται στην ουσία όλες οι επικοινωνίες που χρειάζονται ώστε να παίζεται το παιχνίδι σαν ένα ενιαίο ταμπλό. Εδώ έχουμε και τους 2 διαφορετικούς τρόπους υλοποίησης: με MPI (Point to Point Communication) και MPI3 shared memory – RMA.

  1) MPI (Point to Point Communication)

     Στην υλοποίηση μου, οι επικοινωνίες γίνονται με τους γείτονες, που είναι το προηγούμενο και το επόμενο process. Ξεχωρίζω τη περίπτωση όπου έχω 1 μόνο process οπότε απλά δεν γίνονται επικοινωνίες.

     Οι μεταβλητές next,prev χρησιμοποιούνται στις συναρτήσεις MPI_Isend και MPI_Irecv όπως αντίστοιχα και οι buffers. Τα MPI_Request χρησιμοποιούνται στις κλήσεις της MPI. Μερικές ιδιαιτερότητες: αν είμαι το process 0 (το 1ο δηλαδή) θέτω το prev= nproc-1 ,η τελευταία διεργασία δηλαδή, και αν είμαι στη τελευταία διεργασία θέτω το next=0 η πρώτη διεργασία δηλαδή.

```
int next,prev,nproc,rank;
 int8_t *recv_buffer1,*recv_buffer2,*send_buffer1,*send_buffer2;
 recv_buffer1=(int8_t *)malloc(M*sizeof(int8_t));
 recv_buffer2=(int8_t *)malloc(M*sizeof(int8_t));
 send_buffer1=(int8_t *)malloc(M*sizeof(int8_t));
 send_buffer2=(int8_t *)malloc(M*sizeof(int8_t));
 MPI_Comm_rank(MPI_COMM_WORLD, &rank);
 MPI_Comm_size(MPI_COMM_WORLD, &nproc);
 send_buffer1=&Board(0,0);
 send_buffer2=&Board(N-1,0);
 MPI_Request send_req[2],recv_req[2];
 prev=rank-1;
 next=rank+1;
 //MPI Communication
 if(rank==0) prev=nproc-1;
 if(rank==(nproc-1)) next=0;
```

```
MPI_Irecv(recv_buffer1,M,MPI_INT8_T,prev,prev,MPI_COMM_WORLD,&recv_req[0]);
MPI_Irecv(recv_buffer2,M,MPI_INT8_T,next,next,MPI_COMM_WORLD,&recv_req[1]);
```

Η επικοινωνία γίνεται με την κλήση των ασύγχρονων συναρτήσεων MPI_Isend & MPI_Irecv. Εξήγηση των ετικετών: Στέλνω στο προηγούμενο process με tag το id μου και στο επόμενο process με tag το id μου. Λαμβάνω από το προηγούμενο process με tag το id του προηγούμενου process και το ίδιο για το επόμενο.

Μέχρι να ληφθούν και σταλούν οι πληροφορίες κάνω την δουλεία στα ενδιάμεσα κελιά του πίνακα που δεν χρειάζονται γνώση της πρώτης ή τελευταίας γραμμής. Όταν τελειώσω την δουλεία με τα ενδιάμεσα στοιχεία του πίνακα κάνω MPI_Waitall και για τα recv_req και για τα send_req, έπειτα υπολογίζω και τις νέες καταστάσεις του board για την πρώτη και τελευταία γραμμή. Ο κώδικας για τον υπολογισμό είναι εύκολος κάνω χρήση δυο

προσωρινών πινάκων temp και temp2 3xM όπου η μεσαία γραμμή είναι αυτή που θέλουμε να μάθουμε την νέα της κατάσταση.

2) MPI3 - RMA shared Memory

Εδώ οι επικοινωνίες γίνονται μέσω ενός παραθύρου shared memory. Κάθε process κάνει share την πρώτη και τελευταία γραμμή.

```
int8_t first_row[M],last_row[M],local_fr[M],local_lr[M];
MPI_Win frwin,lrwin;
MPI_Win_create(first_row,M,sizeof(int8_t),MPI_INFO_NULL,MPI_COMM_WORLD,&frwin);
MPI_Win_create(last_row,M,sizeof(int8_t),MPI_INFO_NULL,MPI_COMM_WORLD,&lrwin);
```

Έπειτα γεμίζω την first_row και last_row με τα στοιχεία της πρώτης και τελευταίας γραμμής και κάνω MPI_Win_fence(0,frwin) MPI_Win_fence(0,lrwin). Έπειτα κάνω όπως και πριν τις πράξεις για τις ενδιάμεσες τιμές που δεν χρειάζονται παραπάνω πληροφορίες. Μόλις τελειώσω με αυτές, με τη MPI_Get παίρνω τις τιμές της πρώτης και τελευταίας γραμμής και βρίσκω τις νέες τιμές της 1ης και τελευταίας γραμμής, όμοια με την MPI(Point to Point).

Μια ακόμη αλλαγή που έκανα είναι ότι αφαίρεσα την διπλή for για το γέμισμα του πίνακα board με τις νέες τιμές. Πλέον αλλάζω τις διευθύνσεις των πινάκων board και newboard στην main μετά την κλήση της συνάρτησης play.

Παραλληλοποίηση με openMP

Σχετικά με την OpenMP, έχει απλές παραλληλοποιήσεις κάποιων for χωρίς ιδιαίτερες δυσκολίες αφού όλες οι πράξεις είναι ανεξάρτητες.

Συνάρτηση generate_board : Η συνάρτηση αυτή αρχικοποιεί το board δίνοντάς του τυχαίους αριθμούς. Εδώ δεν γίνεται να εφαρμόσω openMP γιατί η συνάρτηση rand() δεν είναι thread safe. Δοκίμασα να βρω άλλες συναρτήσεις thread safe για να πετύχω καλύτερους χρόνους. Οι περισσότερες μου δίναν ίδιους χρόνους με το σειριακό ή και χειρότερους. Υπήρχαν κι άλλες που έδιναν μισό χρόνο όπως η rand_r() και για seed χρησιμοποίησα το process id επί του omp_get_num_thread() επί το time(NULL). Έδινε καλά νούμερα αλλά μου έδινε πάντα ίδια 1η γραμμή σε όλα τα processes. Σε ένα μεγάλο πίνακα από τυχαίους αριθμούς είναι λογικό να εμφανιστεί 2 ή και παραπάνω φορές ίδια γραμμή αλλά επειδή ήταν σε συγκεκριμένες θέσεις δεν την κράτησα σαν υλοποίηση. Οι διαφορές του χρόνου ήταν γύρω στα 4-5 sec κάτω από τον σειριακό. Η αλλαγή που έκανα στην σειριακή rand() ήταν να προσθέσω απλά στο seed το process id ώστε να μη παίρνει για διαφορετικά processes ίδιες τιμές ο πίνακας.

```
MPI_Comm_rank(MPI_COMM_WORLD, &rank);
srand((rank+1)*time(NULL));
```

Συνάρτηση play : Σε αυτήν το δύσκολο κομμάτι ήταν η διπλή for για τον υπολογισμό των τιμών του newboard. Η παραλληλοποίησή της είναι η εξής:

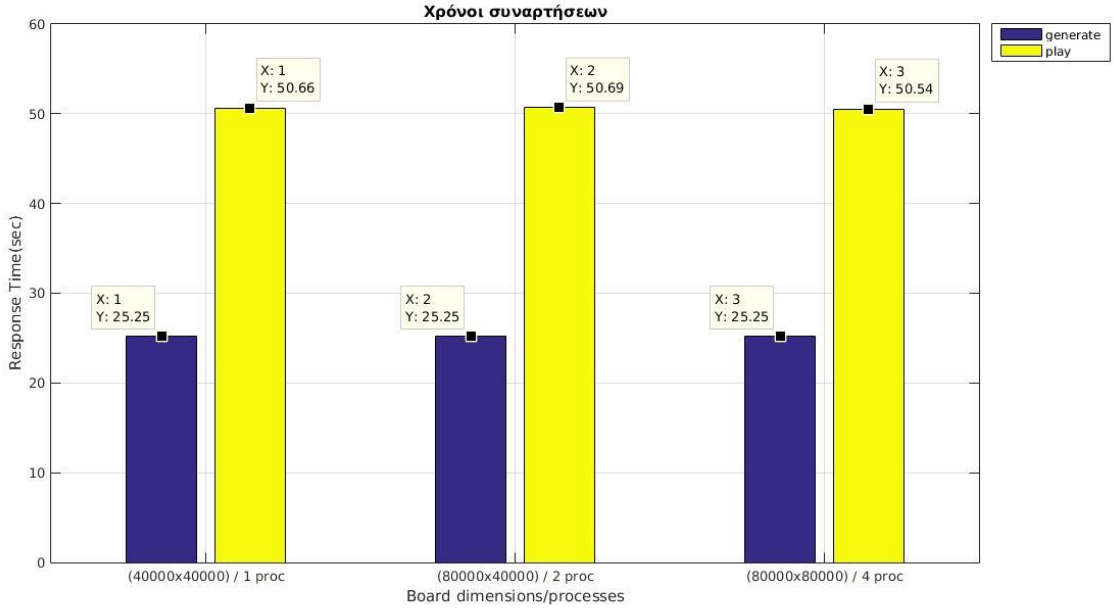
```
#pragma omp parallel for schedule(dynamic) private(i,j,a) shared(newboard,board,N,M) num_threads(NTHREADS)
```

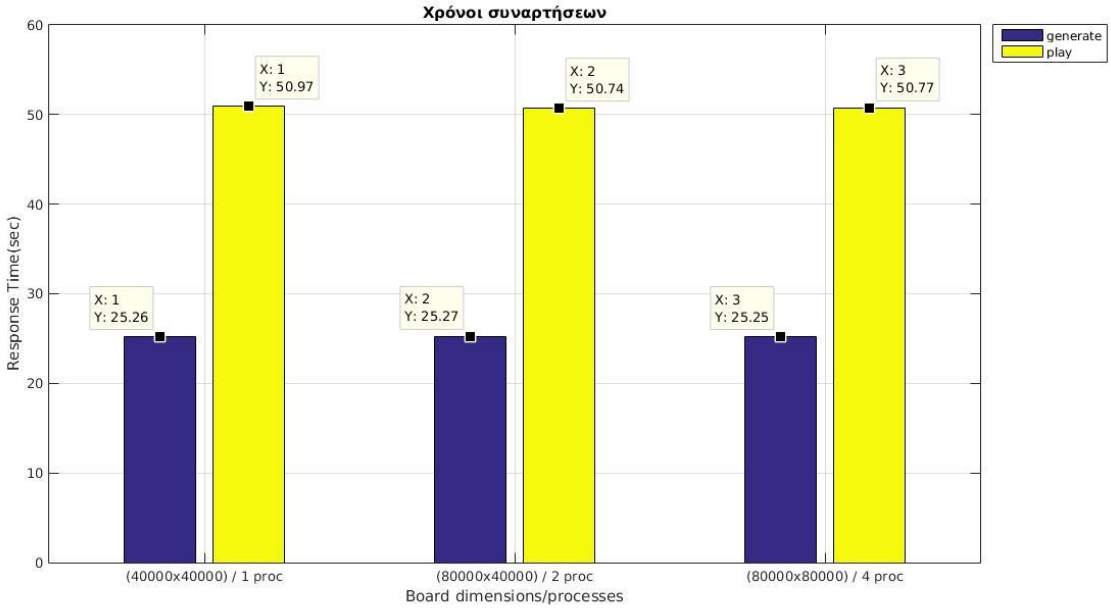
Η μεταβλητή NTHREADS είναι define στην αρχή και ίση με 8. Άλλες δυσκολίες δεν είχα στην openMP οι υπόλοιπες for που ήταν για παραλληλοποίηση είναι απλές και δεν έχουν καμία δυσκολία.

Έχω κάνει μικροαλλαγές και στις άλλες συναρτήσεις όπως το header game-of-life.h στο οποίο άλλαξα το define του Board και τα ορίσματα των συναρτήσεων και στις συναρτήσεις της io_functions.c helpers.c . Λόγω των μεγάλων πινάκων(40000x40000 80000x40000 80000x80000) με την io_functions.c δεν έκανα μετρήσεις για την αποτελεσματικότητα των παραλληλοποίησεων μου. Στην ουσία δεν την χρησιμοποίησα καθόλου.

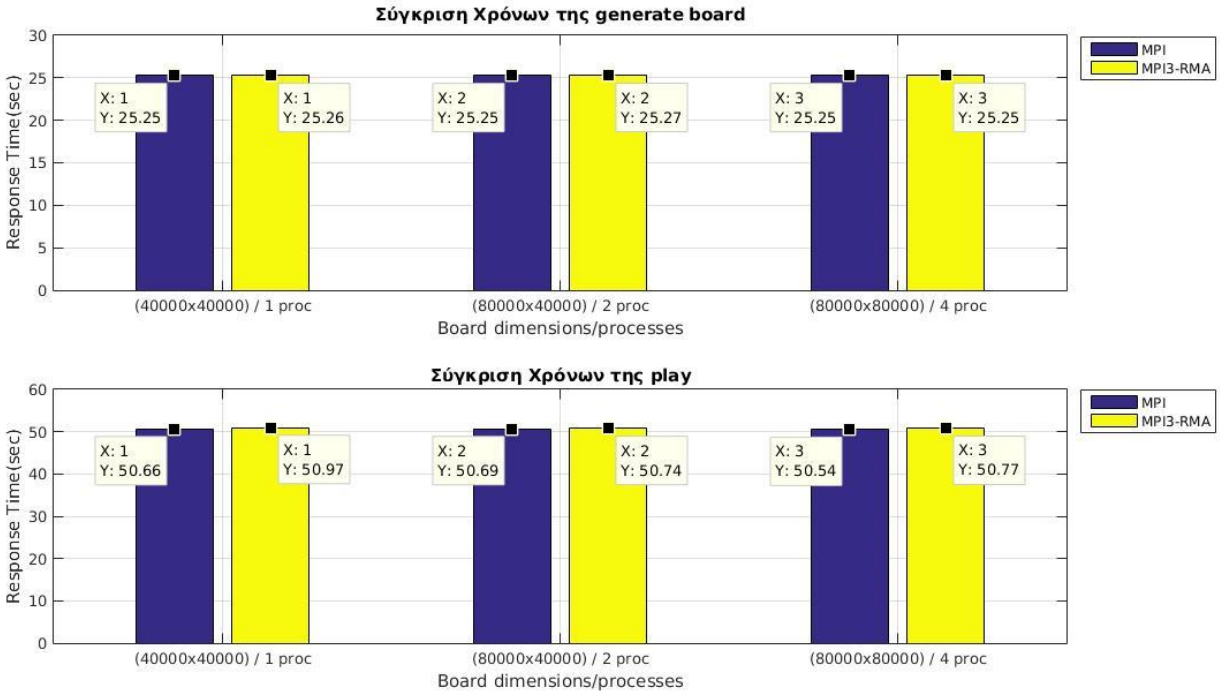
Σύγκριση αποτελεσμάτων MPI και MPI-3 RMA

Στα δύο πρώτα διαγράμματα φαίνονται οι χρόνοι των συναρτήσεων generate_board και play. Πρέπει να σημειωθεί ότι δεν χρησιμοποίησα MPI_Barrier για την μέτρηση της generate. Ο χρόνος είναι αυτός του πρώτου process.

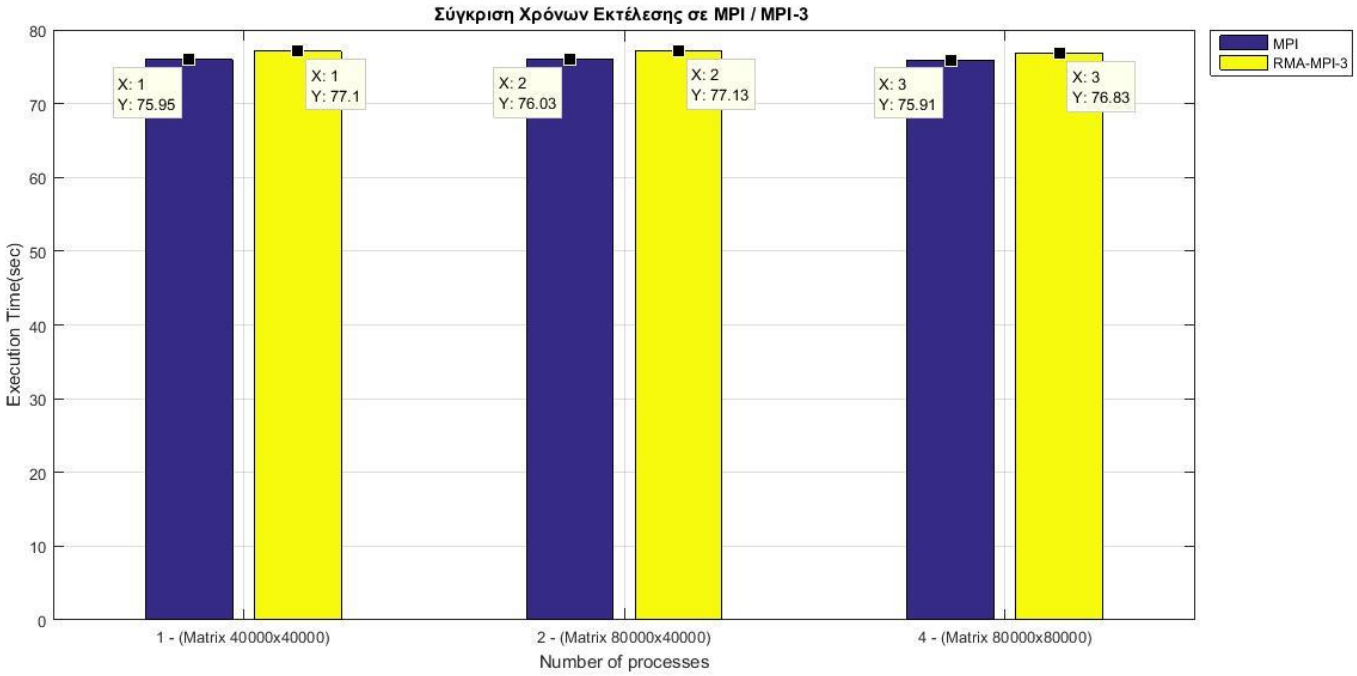




Εδώ συγκρίνω τις δυο υλοποιήσεις στους χρόνους εκτέλεσης. Παρατηρώ ότι είναι περίπου ίδιοι με ελάχιστα πιο αργή η MPI3-RMA υλοποίηση.



Παρατηρώ ότι η υλοποίηση με MPI Point to Point Communication είναι λίγο πιο γρήγορη από ότι η υλοποίηση με τις νέες συναρτήσεις της MPI 3 RMA.



Παρατήρηση: Για την μέτρηση των χρόνων των συναρτήσεων χρειάστηκε να τροποποιήσω τον κώδικα και να μετρήσω τον χρόνο μετά την generate με παρόμοιο τρόπο, όπως φαίνεται για τον συνολικό χρόνο μέσα στο πρόγραμμα(στην main).