ΠΑΡΑΛΛΗΛΑ ΚΑΙ ΔΙΑΝΕΜΗΜΕΝΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΙΑ ΙΙ

Παραλληλοποίηση προγράμματος Conway’s Game of Life

ΜΑΣΤΟΡΑΣ ΡΑΦΑΗΛ ΕΥΑΓΓΕΛΟΣ

ΑΕΜ 7918

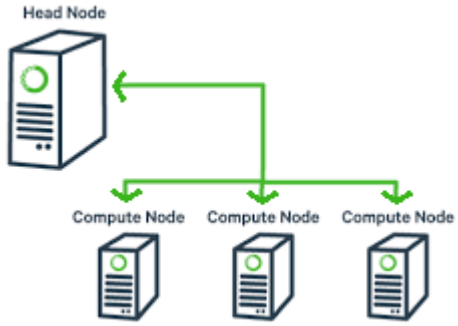
ΔΕΚΕΜΒΡΙΟΣ 2016

Περιγραφή του προγράμματος

Το Game of Life είναι ένα κυτταρικό αυτόματο το οποίο προσομοιώνει τους βασικούς κανόνες τις ζωής. Πρόκειται δηλαδή για μία ομάδα “ατόμων” τα οποία ζευγαρώνουν , γεννούν και πεθαίνουν ανάλογα με το πληθυσμό των γειτονικών ατόμων. Έτσι αν υπάρχει υπερπληθυσμός ή πολύ μικρός πληθυσμός το άτομο πεθαίνει , αν υπάρχουν ιδανικές συνθήκες γεννιέται ένα καινούριο άτομο, αλλιώς παραμένει στην ίδια κατάσταση.

Το πρόγραμμα αρχικά δημιουργεί έναν πίνακα X*Y στοιχείων τα οποία δίνονται από τον χρήστη. Έπειτα γεμίζει τυχαία τον πίνακα βάση της πιθανότητας που εισάγαμε. Για κάθε ένα από τα κελιά του πίνακα υπολογίζει την επόμενη του κατάσταση ανάλογα με το πόσα γειτονικά κελιά είναι “ζωντανά” και τέλος εκτυπώνει τον πίνακα, εφόσον το επιθυμούμε, και επαναλαμβάνει τα προηγούμενα βήματα(υπολογισμός της νέας κατάστασης και μετά) για όσες φορές έχει ζητηθεί.

Παράλληλη υλοποίηση με OpenMp & MPI

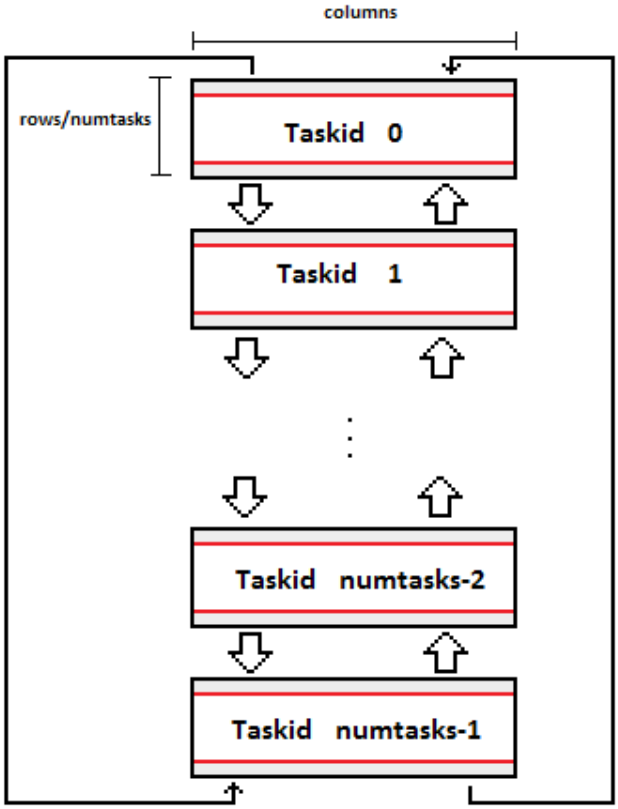


<u>MPI</u> **Με ασύγχρονες εντολές αποστολής & λήψης**

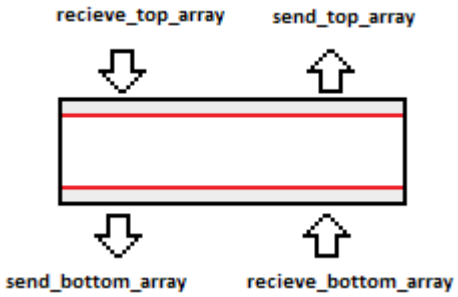
Στην υλοποίηση μου χωρίζω τον πίνακα X*Y , σε μικρότερους πίνακες (X/αριθμός διεργασιών)*Y κάθε ένας από τους οποίους δημιουργείται από μια διεργασία. Έτσι το τελικό αποτέλεσμα θα είναι ένας πίνακας X*Y διαμοιρασμένος στις εκάστοτε διεργασίες. Ο λόγος που το χώρισα σε σειρές αντί για στήλες είναι για να γίνει πιο εύκολα η υλοποίηση της display που τυπώνει σειρά σειρά. Σε αντίθετη περίπτωση θα χρειαζόταν περισσότερες Send και Recv ώστε να τυπωθεί σωστά το board πράγμα που θα κόστιζε σημαντικό χρόνο.

Κάθε διεργασία ,αφού ορίσει τα rows=Nx/numtasks και columns=Ny, καλεί την συνάρτηση generate_table(board,rows,columns,thres) για να γεμίσει τον πίνακα με

τυχαίο τρόπο. Προκειμένου η generate κάθε διεργασίας να παράγει διαφορετικό αποτέλεσμα χρησιμοποίησα την εντολή srand(time(NULL)+rank). Έπειτα κάθε διεργασία καθορίζει τις μεταβλητές prev και next οι οποίες αντιπροσωπεύουν την προηγούμενη και την επόμενη διεργασία αντίστοιχα. Για την διεργασία 0 θεωρώ προηγούμενη την τελευταία διεργασία ενώ για την τελευταία διεργασία θωρώ επόμενη την πρώτη διεργασία προκειμένου να δημιουργήσω κυκλικές συνθήκες στον κάθετο άξονα. Οι κυκλικές συνθήκες στον οριζόντιο άξονα παρέχονται από την συνάρτηση yadd.

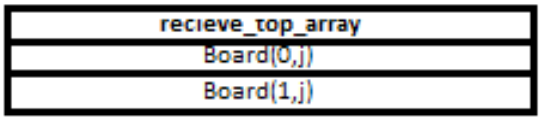


Ύστερα κάθε διεργασία αφού ορίσει τους πίνακες send_bottom_array και send_top_array, στέλνει την πρώτη γραμμή του πίνακα στην προηγούμενη διεργασία και την τελευταία γραμμή στην επόμενη της. Επίσης λαμβάνει την πρώτη γραμμή της επόμενης διεργασίας και την τελευταία γραμμή της προηγούμενης της. Αυτά γίνονται με την χρήση ασύγχρονων εντολών αποστολής και λήψης, Isend και Irecv.

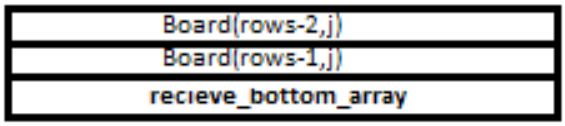


Μέχρι να ολοκληρωθούν οι απαραίτητες αποστολές και λήψεις κάθε διεργασία καλεί την συνάρτηση play(board, newboard, rows,columns). Μέσα στην συνάρτηση αυτή ελέγχεται αν ο αριθμός των διεργασιών είναι μεγαλύτερος από 1 και τότε ορίζονται οι firstrow=1 και lastrow=1 ώστε να χρησιμοποιηθούν στην for (i=firstrow; i<rows-lastrow; i++) προκειμένου να τρέξει μόνο για τις ενδιάμεσες σειρές του πίνακα, διαφορετικά παίρνουν τις τιμές 0 και το πρόγραμμα τρέχει για όλες τις τιμές του πίνακα.

Επιστρέφοντας από την συνάρτηση play κάθε διεργασία περιμένει να ολοκληρωθεί η λήψη της τελευταίας σειράς της προηγούμενης διαδικασίας με την εντολή Wait. Έπειτα δημιουργεί έναν πίνακα temp_board μεγέθους 3*columns και καλεί την συνάρτηση play_temp(board,temp_board,newboard,3,columns,0). Το όρισμα 0 δείχνει σε ποιά θέση του πίνακα newboard θα αποθηκευτεί η νέα τιμή.



Όμοια κάνει μετά την λήψη της πρώτης σειράς της επόμενης διεργασίας.



Η συνάρτηση play_temp είναι διαμορφωμένη ώστε να τρέχει μόνο για την σειρά που μας ενδιαφέρει και να αποθηκεύει την νέα τιμή στην κατάλληλη θέση του πίνακα newboard.

Μετά καλείται η συνάρτηση swap που αλλάζει τις διευθύνσεις των πινάκων board και newboard ώστε να ανανεωθεί ο πίνακας board. Έτσι εξοικονομείται πολύ χρόνος σε σχέση με την διαδοχική αποθήκευση των στοιχείων του newboard, στον board που χρησιμοποιούσε το αρχικό πρόγραμμα που δόθηκε.

Τέλος συγχρονίζονται όλες οι διεργασίες και ελέγχεται αν ζητείται από τον χρήστη η τύπωση του πίνακα. Για την σωστή τύπωση του πίνακα ,όταν υπάρχουν πολλές διεργασίες, χρησιμοποιώ τις εντολές Send και Recv που μπλοκάρουν την εκτέλεση μέχρι να ολοκληρωθούν. Έτσι ξεκινάει και τυπώνει πρώτη η διεργασία και αφού τελειώσει στέλνει ένα μικρό πακέτο (integer) στην επόμενη διεργασία. Όλες οι διεργασίες πλην της πρώτης περιμένουν να κάνουν λήψη και μετά ξεκινάνε να τυπώσουν τον πίνακα, ενώ αφού τον τυπώσουν στέλνουν και αυτές ένα μικρό πακέτο. Στο τέλος η πρώτη διεργασία αφού κάνει λήψη καλεί ένα μικρό delay και καθαρίζει την κονσόλα.

Η παραπάνω διαδικασία επαναλαμβάνεται για όσα Generations έχουν ζητηθεί.

MPI **Με shared memory -RMA**

Η υλοποίηση αυτή είναι παρόμοια με την προηγούμενη με μία διαφορά. Αντί κάθε διεργασία να στέλνει την πρώτη και τελευταία γραμμή, αυτή τη φορά δημιουργεί δύο πίνακες top_row και bottom_row, προσβάσιμους από τις άλλες διεργασίες. Αυτό το πέτυχα με τις εντολές

```
MPI_Alloc_mem(columns*sizeof(int), MPI_INFO_NULL, &top_row);

MPI_Alloc_mem(columns*sizeof(int), MPI_INFO_NULL, &bottom_row);

MPI_Win_create(top_row,(columns)*sizeof(int),sizeof(int),MPI_INFO_NULL,MPI_COMM_WORLD,&top_win);

MPI_Win_create(bottom_row(columns)*sizeof(int),sizeof(int),MPI_INFO_NULL,MPI_COMM_WORLD,&bottom_w
in);
```

όπου top_row η πρώτη γραμμή κάθε διεργασίας, και αντίστοιχα bottom_row η τελευταία. Πρόσβαση στην πρώτη σειρά έχουν οι διεργασίες με την εντολή

```
MPI_Win_fence(0 , top_win);

MPI_Get(get_top_row,columns,MPI_INT,next,0,columns,MPI_INT,top_win);

MPI_Win_fence(0 , top_win);
```

ενώ στην τελευταία με αντίστοιχη εντολή. Πριν από κάθε καινούριο generation χρειάζεται να ανανεώνω τα στοιχεία των top_row και bottom_row.

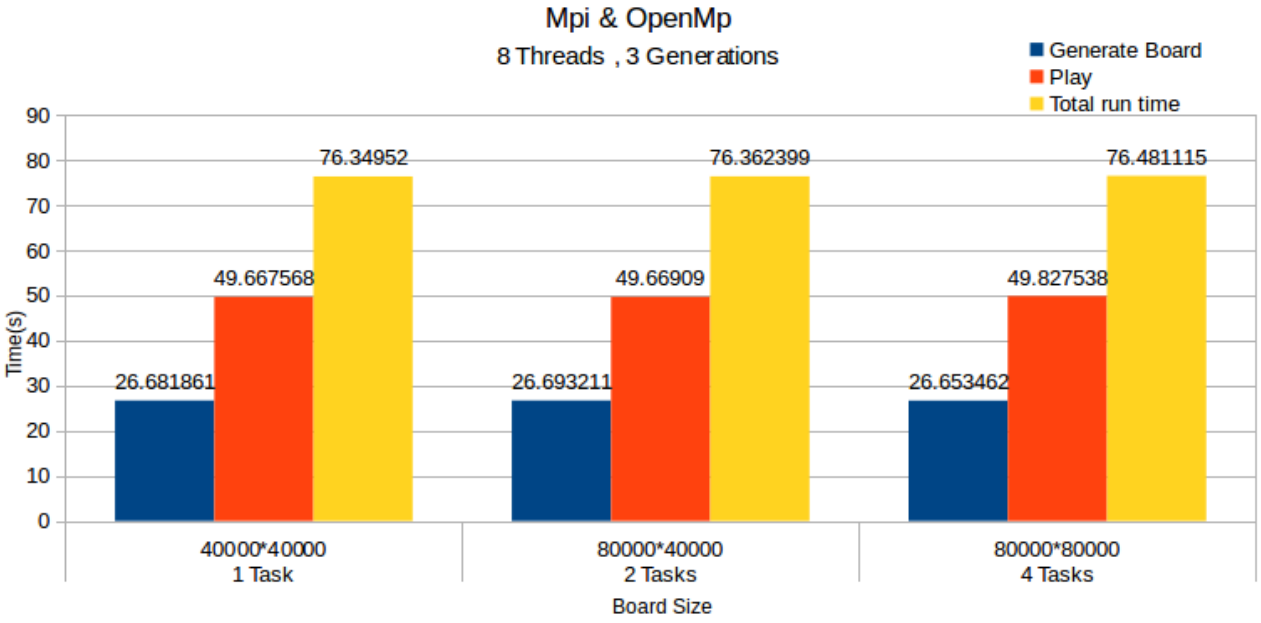
OpenMp

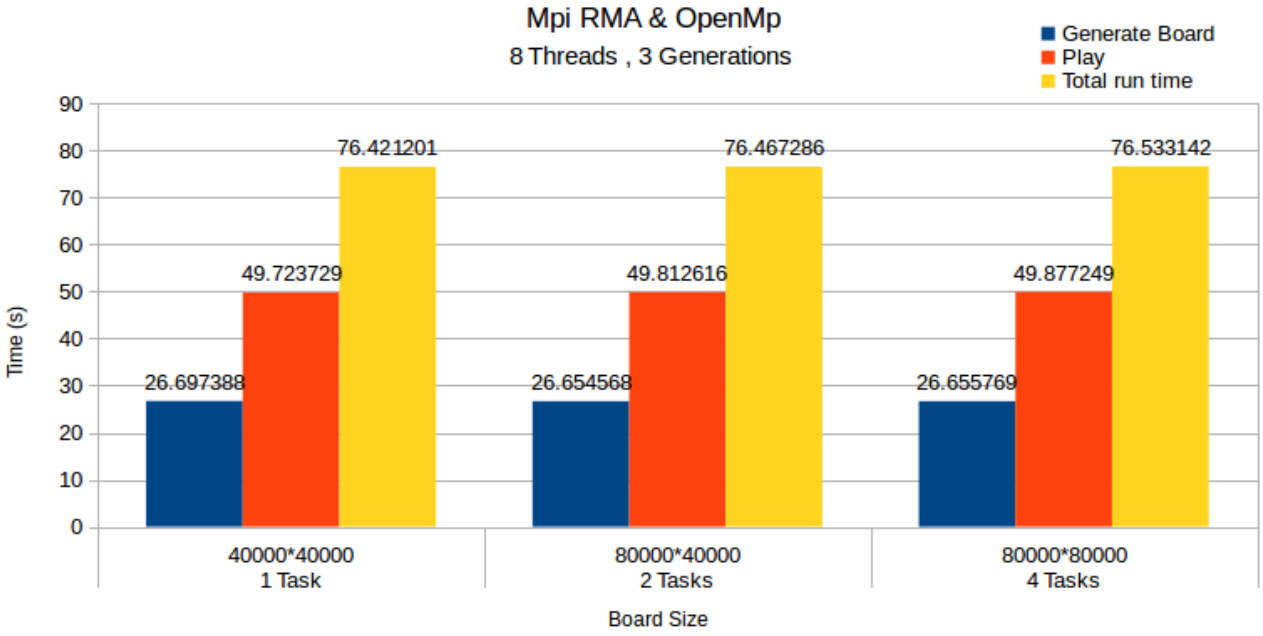
Χρήση του OpenMp έκανα μόνο στην συνάρτηση play και play_temp. Η συνάρτηση generate_board περιέχει την εντολή rand() η οποία δεν είναι thread safe και για αυτό δεν μπόρεσα να την υλοποιήσω σωστά με OpenMp.

Στην συνάρτηση play χρησιμοποίησα την εντολή:
`#pragma omp parallel for private(i) shared(board,newboard) num_threads(NUM_THREADS)`
καθώς και έβαλα την εμφωλευμένη for μέσα σε μία συνάρτηση
`create_new_board(i,board,newboard,rows,columns)` προκειμένου να αποφύγω προβλήματα.

Παρόμοια για στην play_temp χρησιμοποιώ την εντολή :
`#pragma omp parallel for private(j,a) shared(board,newboard) num_threads(NUM_THREADS)`
Η παραπάνω απλή υλοποίηση με OpenMP όπως θα δούμε είναι αρκετή για να επιταχύνει 8 φορές το πρόγραμμα μας.

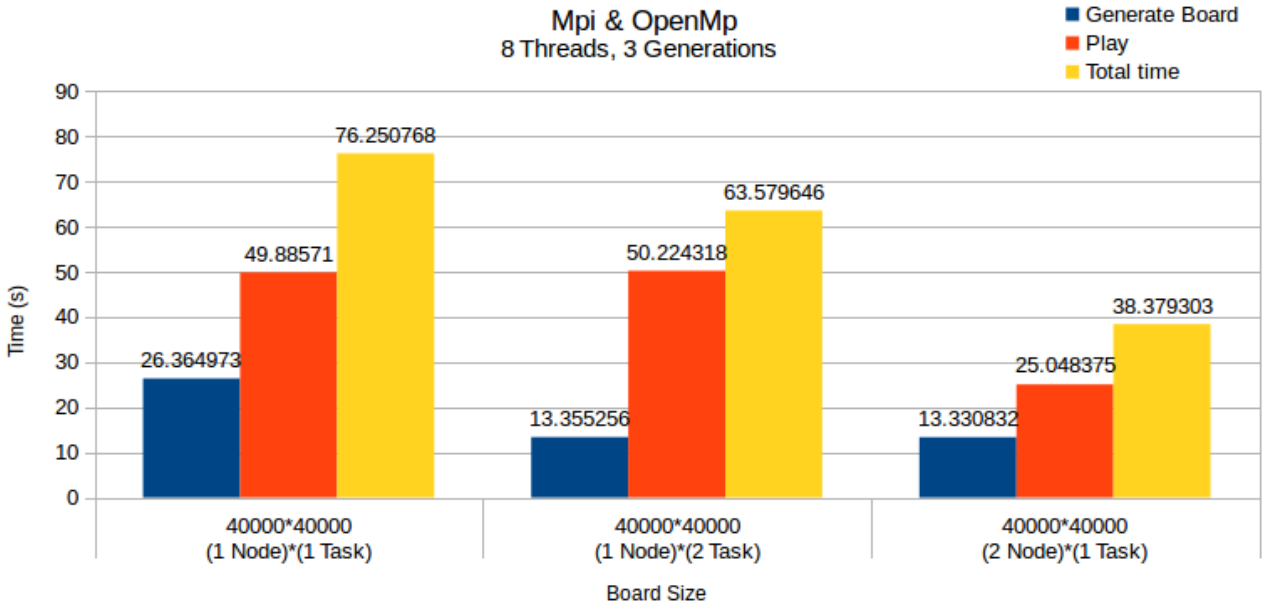
Αποτελέσματα και Σχολιασμός





Παρατηρούμε ότι για τα ζητούμενα από την εργασία αποτελέσματα , υπάρχει σταθερή συμπεριφορά ενώ επίσης η χρονική διαφορά είναι αμελητέα. Καταφέραμε δηλαδή να κρύψουμε όλη την επικοινωνία μεταξύ των διεργασιών χρησιμοποιώντας ασύγχρονες εντολές αποστολής και λήψεις ενώ στο ενδιάμεσο το πρόγραμμα εκμεταλλευόταν χρήσιμα τον χρόνο. Η υλοποίηση με τις ασύγχρονες εντολές αποστολής και λήψεις είναι ελάχιστα πιο γρήγορη από την υλοποίηση με την shared memory αλλά και εδώ η διαφορά είναι αμελητέα.

Για τα παραπάνω αποτελέσματα χρησιμοποίησα διαφορετικά Nodes προκειμένου να δημιουργήσω καινούριο task. Τί γίνεται αν χρησιμοποιήσουμε το ίδιο node όμως;

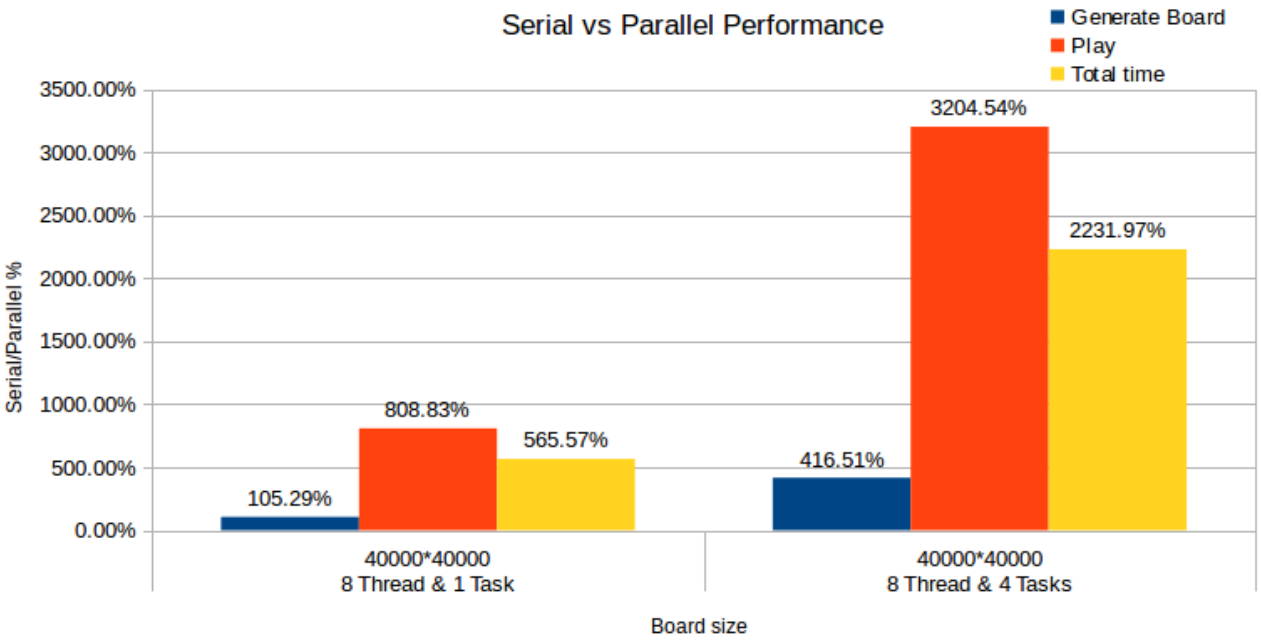


Βλέπουμε ότι ναι μεν μειώνεται ο χρόνος εκτέλεσης αλλά δεν μειώνεται αισθητά όπως όταν χρησιμοποιούμε άλλο node (βλπ. τρίτη στήλη). Αυτό συμβαίνει γιατί η συνάρτηση play χρησιμοποιεί threads, επομένως όταν στο ίδιο μηχάνημα τρέχουμε δύο συναρτήσεις play κάποια threads θα πρέπει να περιμένουν να τελειώσουν τα προηγούμενα καθώς δεν υπάρχουν 16 πυρήνες στο σύστημα μας.

Τώρα ας δούμε τον χρόνο εκτέλεσης του σειριακού προγράμματος, όπως αυτό μας δόθηκε.

	20000*20000	40000*40000
Generate Board	6.940129	27.760516
Play	100.872726	403.490904
Total time	107.812958	431.251832

Δυστυχώς λόγο του περιορισμού στον χρόνο από τον server hellasgrid το πρόγραμμα δεν μπορεί να τρέξει για πάνω από 120 δευτερόλεπτα και έτσι δεν μπορούσαμε να πάρουμε χρόνο για μεγαλύτερες τιμές του σειριακού προγράμματος. Ωστόσο με ασφάλεια μπορεί κανείς να υποθέσει ότι για 40000*40000 ο χρόνος εκτέλεσης θα είναι περίπου 4 φορές μεγαλύτερος από ότι για 20000*20000.



Στο παραπάνω διάγραμμα βλέπουμε την επίδοση της υλοποίησης αυτής σε σχέση με το σειριακό. Παρατηρούμε ότι μόνο με OpenMp επιτυγχάνουμε καλύτερο χρόνο εκτέλεσης της play περίπου 800% ενώ όταν χρησιμοποιούμε και MPI για τέσσερεις διεργασίες περίπου 3200%, πράγμα λογικό αφού το πρόβλημα μοιράζεται σε τέσσερα κομμάτια.

Καταλαβαίνει κανείς την σπουδαιότητα του παράλληλου προγραμματισμού αν αναλογιστεί ότι αυτό το πρόγραμμα κόστιζε χρονικά μόνο μερικά δευτερόλεπτα, υπάρχουν όμως προβλήματα που μπορεί να κοστίζουν πολλές ημέρες. Χρησιμοποιώντας τέτοιες τεχνικές μπορεί να επιτευχθεί τεράστια μείωση στην χρονική απαίτηση εκτέλεσης της εκάστοτε εργασίας.

Όλα τα παραπάνω αποτελέσματα πάρθηκαν στον server Hellasgrid. Επίσης έγινε έλεγχος ορθότητας παρακολουθώντας την εκτύπωση του board στην κονσόλα για πολλά generations.