Χειμερινό Εξάμηνο

ΕΡΓΑΣΙΑ ΣΤΑ ΠΛΑΙΣΙΑ ΤΟΥ ΜΑΘΗΜΑΤΟΣ: ΠΑΡΑΛΛΗΛΑ ΣΥΣΤΗΜΑΤΑ

ΕΡΓΑΣΙΑ 2016-17: CONWAY'S GAME OF LIFE

1115201300065 ΚΑΤΗΦΟΡΗΣ ΕΛΕΥΘΕΡΙΟΣ
1115201300177 ΤΟΥΜΑΣΗΣ ΆΓΓΕΛΟΣ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν Πανεπιστήμιον Αθηνών

Περιεχόμενα

Εισαγωγή

Η εργασία καλύπτει όλες τις απαιτήσεις της εκφώνησης και διαθέτει σε καίρια σημεία τον απαραίτητο σχολιασμό.

Αποτελείται από τα παρακάτω αρχεία :

main.c , master.c , worker.c , functions.c, header.h και το ενδεικτικό checker.cpp, με σειριακή υλοποίηση του προβλήματος που χρησιμοποιήθηκε για τη σύγκριση των αποτελεσμάτων του παράλληλου προβλήματος ώστε να ελεγχθούν ότι είναι σωστά τα παραγόμενα αποτελέσματα.

Παρέχεται επιπλέον makefile με τις εντολές για μεταγλώττιση αλλά και τη διαγραφή των αρχείων initial.dat και final.dat που περιέχουν αντίστοιχα το πρόβλημα και τη λύση του, όπως και αρχεία μετρήσεων μετά από κάθε τρέξιμο, για την αποφυγή περιττών στοιχείων.

Για το MPI παράγονται τα:

- “mpi” (εκτελέσιμο για το MPI)
- “mpi_con” (εκτελέσιμο για το MPI με σύγκλιση)

Για το OpenMp παράγονται τα:

- “omp” (εκτελέσιμο για το OpenMp)
- “omp_con” (εκτελέσιμο για το OpenMp με σύγκλιση)

Ενώ για το Cuda παρέχεται ξεχωριστό makefile όπου παράγεται το εκτελέσιμο life_game_cuda.

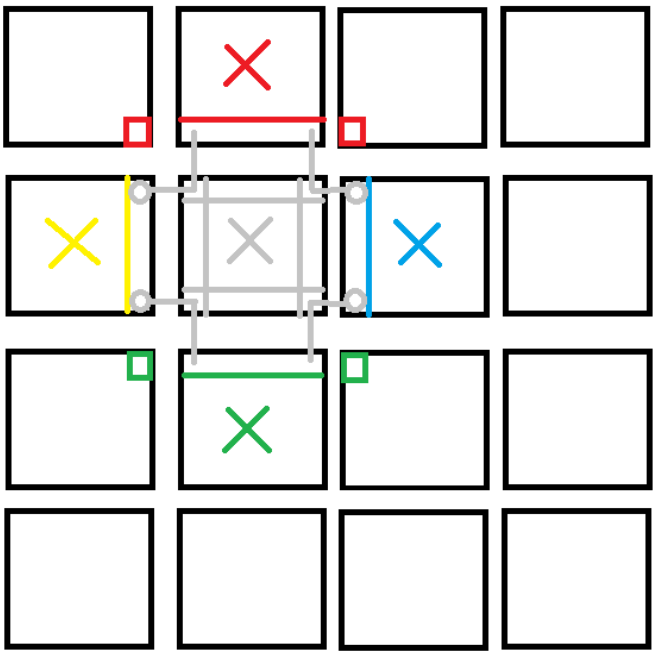
Σχεδιασμός Διαμοιρασμού Δεδομένων

Δημιουργούνται όσες διεργασίες όσες επιλεχθούν από τον χρήστη, εκ των οποίων η πρώτη αναλαμβάνει το ρόλο του MASTER . Ο MASTER διαμοιράζει και αποστέλλει τις πληροφορίες στις υπόλοιπες διεργασίες που έχουν το ρόλο του WORKER. Πραγματοποιείται διαμοιρασμός σε Block και χρήση της καρτεσιανής τοπολογίας (με περιοδικότητα).

Προκειμένου να επιλυθεί το πρόβλημα είναι αναγκαίος ο υπολογισμός των εσωτερικών στοιχείων ενός Block (Independent_Update) , των περιμετρικών (Dependent_Update) και φυσικά των σωστών διαγώνιων (UpdateDiag).

Στο σχήμα δίπλα φαίνεται ο τρόπος επικοινωνίας μεταξύ των Block στα οποία έχουμε διαμοιράσει το πρόβλημα.

Στέλνονται εκτός από τις 4 πλευρές (περιμετρικά) κάθε Block και οι 2 πάνω και οι 2 κάτω διαγώνιοι. Το γκρι Block στο παράδειγμα λαμβάνει τα κυκλωμένα στοιχεία από το κίτρινο και το μπλε και τα προωθεί (Send) αντίστοιχα. Στη συνέχεια μέσα από το πράσινο και το κόκκινο θα μπορέσει να γίνει και η σωστή ενημέρωση των διαγώνιων όπως απαιτείται από το πρόβλημα (8 γείτονες).



Για να αποφύγουμε την επικοινωνία της κάθε διεργασίας με 8 γείτονες αποφασίσαμε για κάθε διεργασία αρχικά να στέλνει στους πάνω και κάτω γείτονές της την πάνω και κάτω γραμμή της αντίστοιχα έτσι ώστε όταν στείλουμε στους δεξιά και αριστερά γείτονες τις αντίστοιχες στήλες να έχουμε στη διάθεσή μας και τις 2 (πάνω και κάτω δεξιά ή αριστερά αντίστοιχα) γωνίες που χρειάζεται το γειτονικό μπλοκ.Έτσι αποφύγαμε την επικοινωνία της κάθε διεργασίας με 8 γείτονες και στην υλοποίησή μας επικοινωνεί μόνο με 4 (πάνω,κάτω,δεξιά και αριστερά) και δε χρειάζεται επικοινωνία με τους διαγώνιους γείτονες.

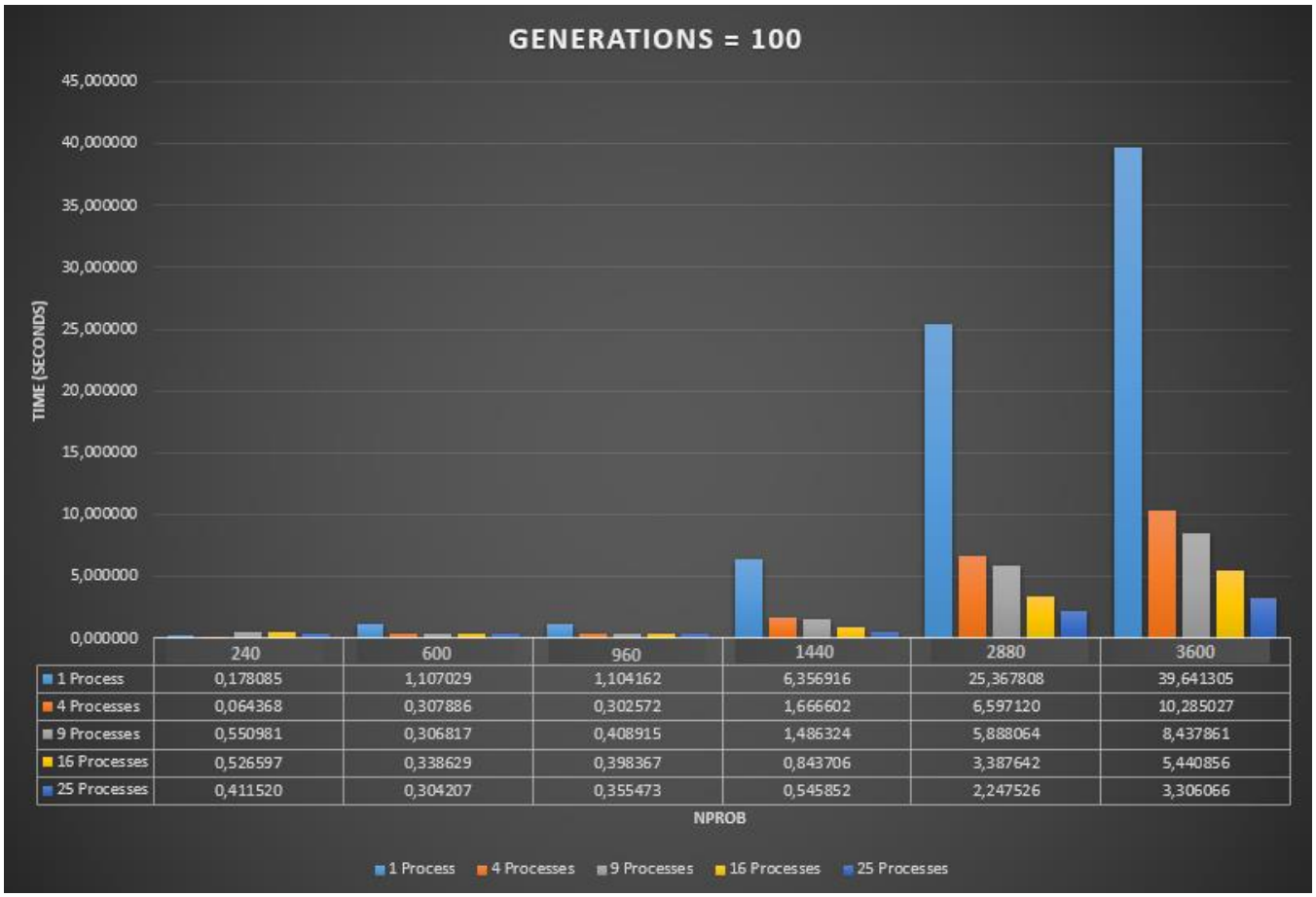
Σχεδιασμός και υλοποίηση MPI κώδικα

Ο MASTER λειτουργεί όπως αναφέρθηκε παραπάνω, ενώ οι WORKERS εκτελούν επαναλήψεις όσες το πλήθος των GENERATIONS και επικοινωνούν ασύγχρονα με τις γειτνιάζουσες διεργασίες για τον υπολογισμό του σωστού αποτελέσματος. Ακόμα γίνεται η χρήση datatypes για την αποφυγή πολλαπλών αντιγραφών που θα επέφερε ιδιαίτερα σημαντικό κόστος στην αποστολή των στηλών των υποπινάκων (subarrays). Η λήψη της πληροφορίας στους WORKERS από τον MASTER γίνεται μέσω της MPI_Irecv, την οποία περιμένουμε να ολοκληρωθεί πριν αρχίσουμε τις πράξεις για κάθε γενιά, καλύπτοντας το συγκεκριμένο χρονικό διάστημα με αρχικοποιήσεις των πινάκων και των request που θα χρησιμοποιήσουμε. Κατά τα GENERATIONS λαμβάνουμε την αριστερή και δεξιά στήλη όσο γίνεται ο υπολογισμός των εσωτερικών στοιχείων (Independent_Update) και στη συνέχεια ετοιμάζουμε ανάλογα τις προωθήσεις των διαγώνιων και συνεχίζουμε με την αποστολή τους. Κάνουμε τον υπολογισμό των περιμετρικών στοιχείων (Dependent_Update) και τέλος αφού λάβουμε τις σωστές διαγώνιες με την ενημέρωση τους (UpdateDiag). Σημαντικό είναι να αναφέρουμε ότι έγινε χρήση του Derived Datatype MPI_CHAR που περιέχει ακέραιους από 0 έως 127 μιας και χρειαζόμαστε μόνο 0 ή 1 για το πρόβλημα μας, οπότε δεσμεύουμε το μικρότερο δυνατό Datatype.

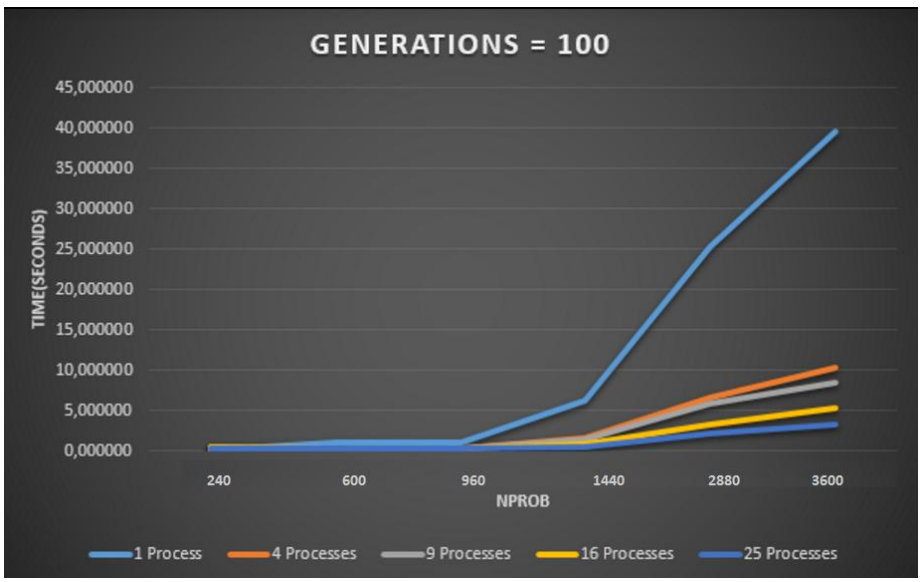
Παρουσίαση αποτελεσμάτων

Σημειώνεται ότι οι παρακάτω μετρήσεις έγιναν στο μηχάνημα linux01 βραδινές ώρες για να υπάρχει περιορισμένη κινητικότητα χρηστών. Επίσης έγιναν το ίδιο βράδυ ώστε να μην υπάρξουν αλλαγές στο αρχείο machines και τα 15 εν λειτουργία μηχανήματα που διέθετε.Τέλος οι μετρήσεις έγιναν πριν το καλοκαίρι όπου ακόμα ήταν αρκετά μηχανήματα ενεργά.

MPI

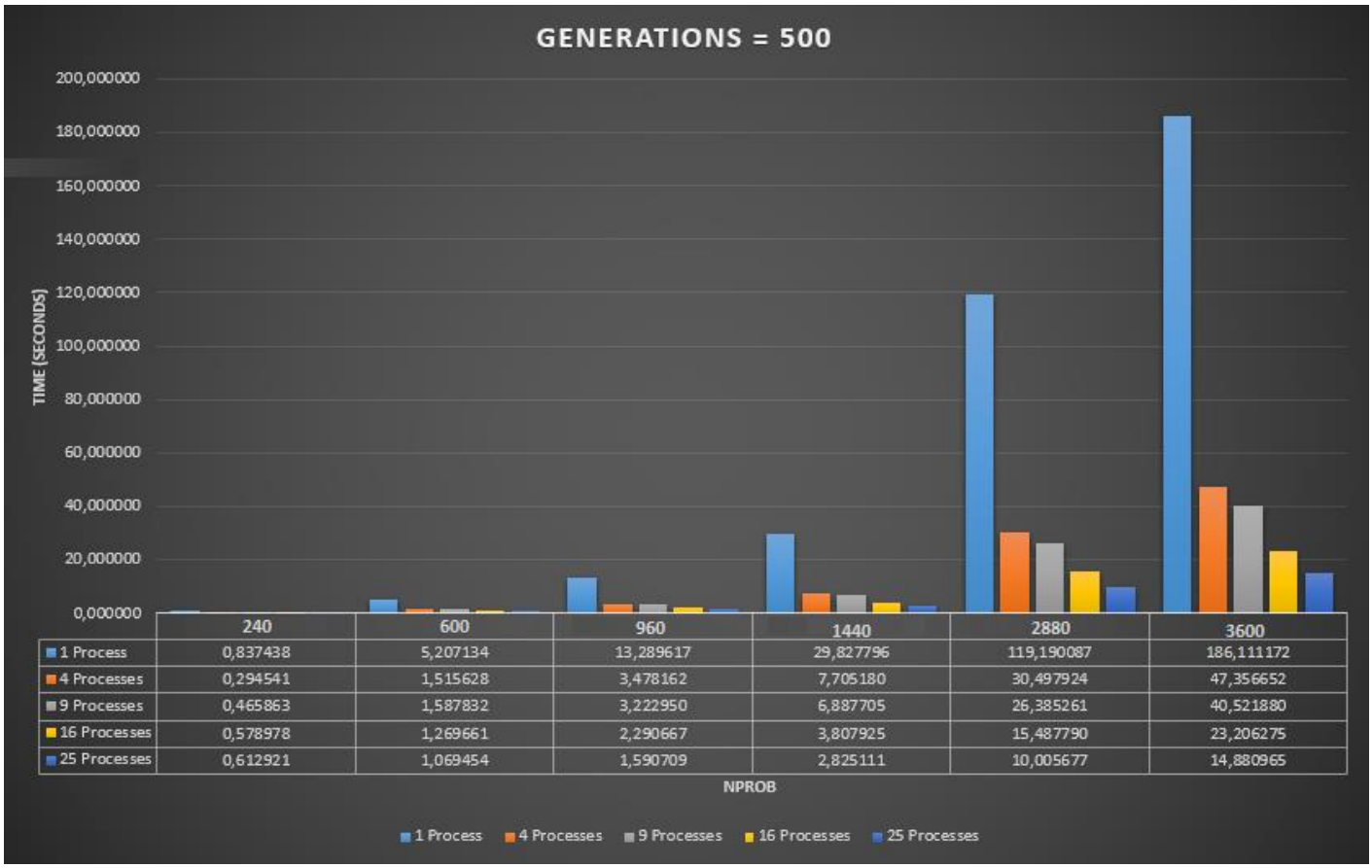


	240	600	960	1440	2880	3600
1 Process	0,178085	1,107029	1,104162	6,356916	25,367808	39,641305
4 Processes	0,064368	0,307886	0,302572	1,666602	6,597120	10,285027
9 Processes	0,550981	0,306817	0,408915	1,486324	5,888064	8,437861
16 Processes	0,526597	0,338629	0,398367	0,843706	3,387642	5,440856
25 Processes	0,411520	0,304207	0,355473	0,545852	2,247526	3,306066

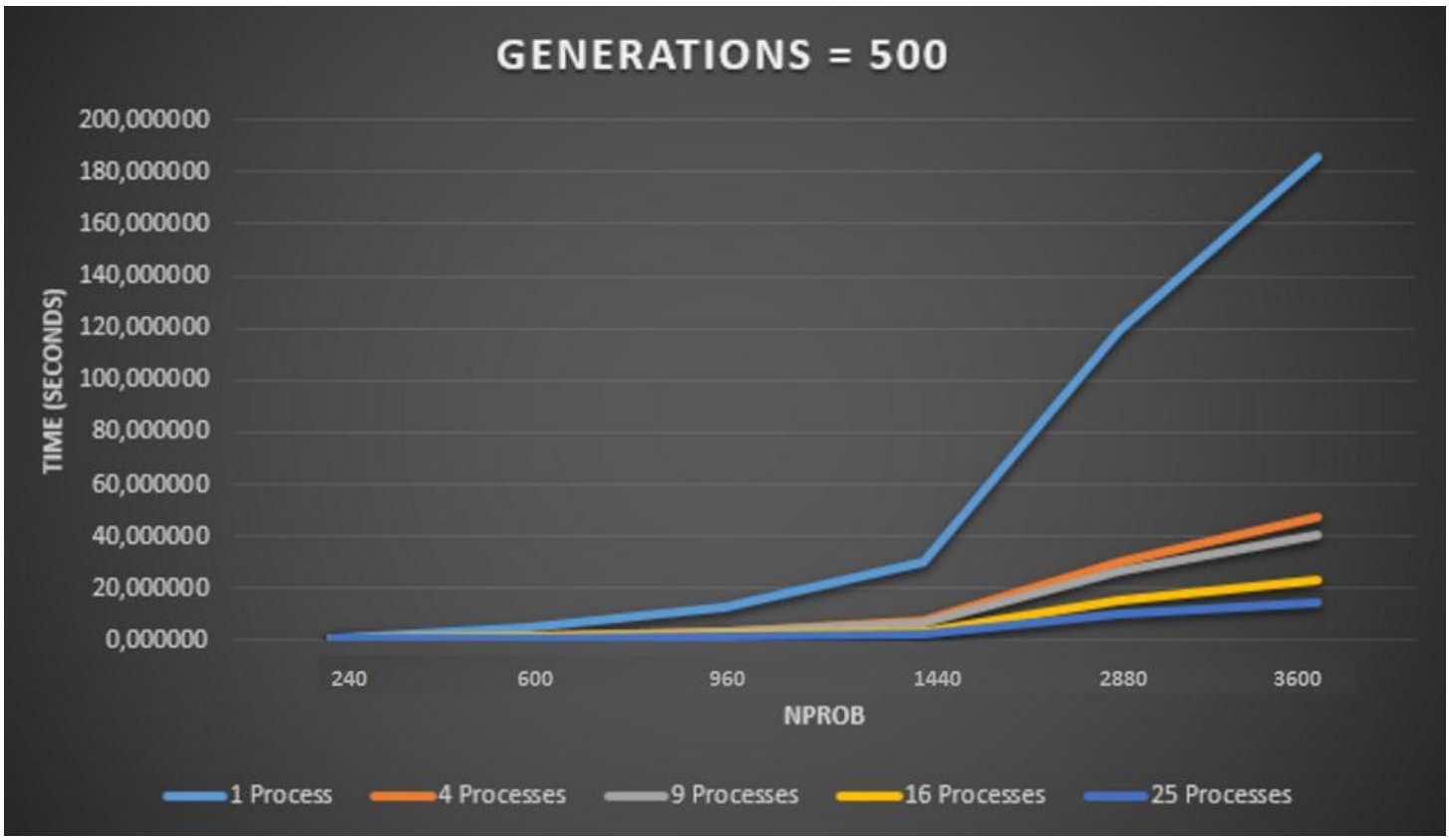


Εύκολα συμπεραίνουμε ότι ειδικά για τα μεγάλα προβλήματα όσο ανεβαίνει ο αριθμός των διεργασιών τόσο πέφτει ο χρόνος εκτέλεσης. Στα πιο μικρά προβλήματα όπως είναι το 240, 600 επειδή και ο αριθμός των Generations είναι μόνο 100 βλέπουμε ότι δεν υπάρχει μεγάλη διαφορά όταν αλλάζουμε τον αριθμό των διεργασιών κάτι που είναι λογικό.

Στο δεύτερο διάγραμμα βλέπουμε τη μεγάλη κλίση της γραμμής της μιας διεργασίας η οποία ανεβαίνει με μεγάλο ρυθμό όσο ανεβαίνει το μέγεθος του προβλήματος. Ενώ για μεγαλύτερο αριθμό διεργασιών η κλίση της γραμμής πέφτει όλο και περισσότερο!

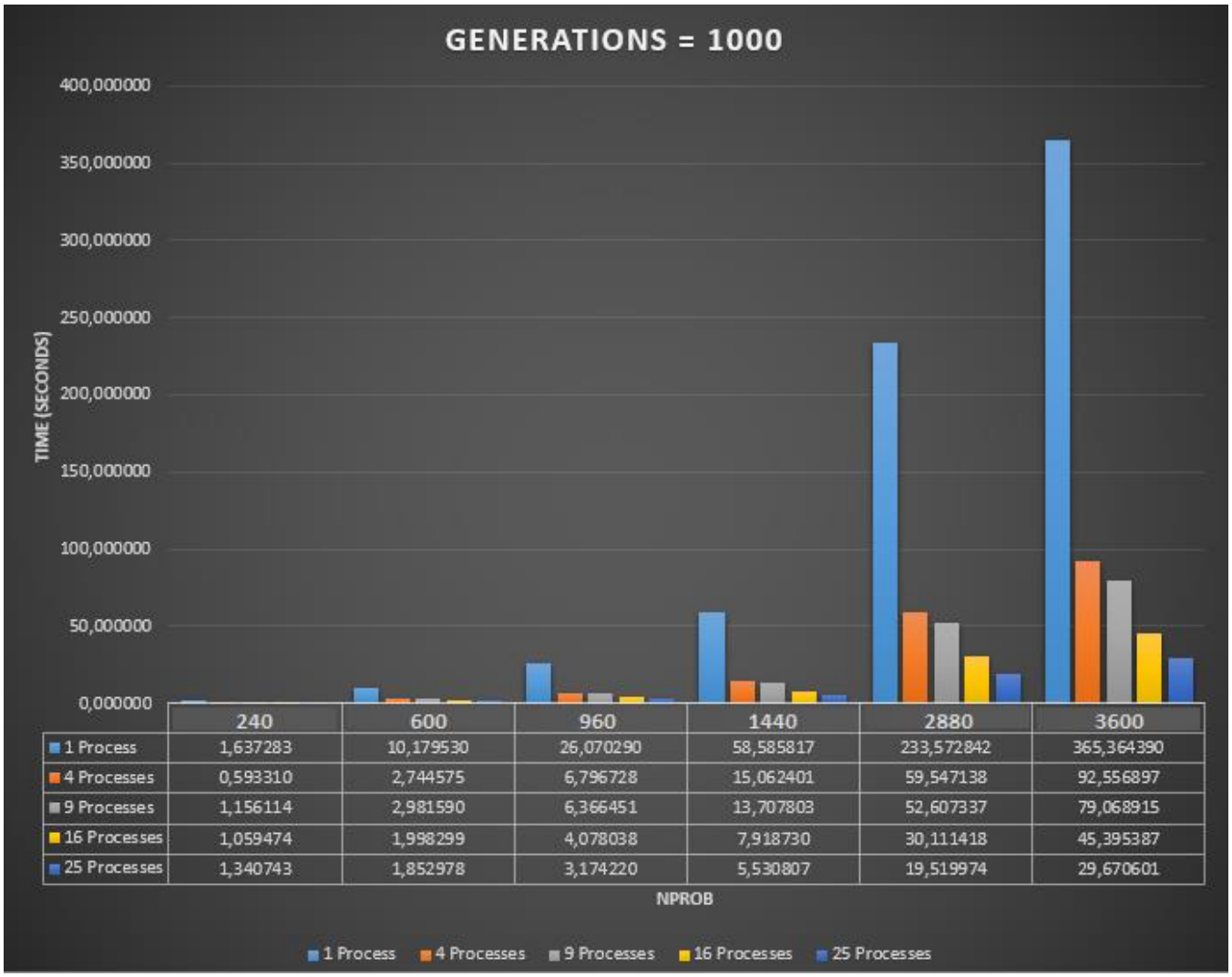


	240	600	960	1440	2880	3600
1 Process	0,837438	5,207134	13,289617	29,827796	119,190087	186,111172
4 Processes	0,294541	1,515628	3,478162	7,705180	30,497924	47,356652
9 Processes	0,465863	1,587832	3,222950	6,887705	26,385261	40,521880
16 Processes	0,578978	1,269661	2,290667	3,807925	15,487790	23,206275
25 Processes	0,612921	1,069454	1,590709	2,825111	10,005677	14,880965

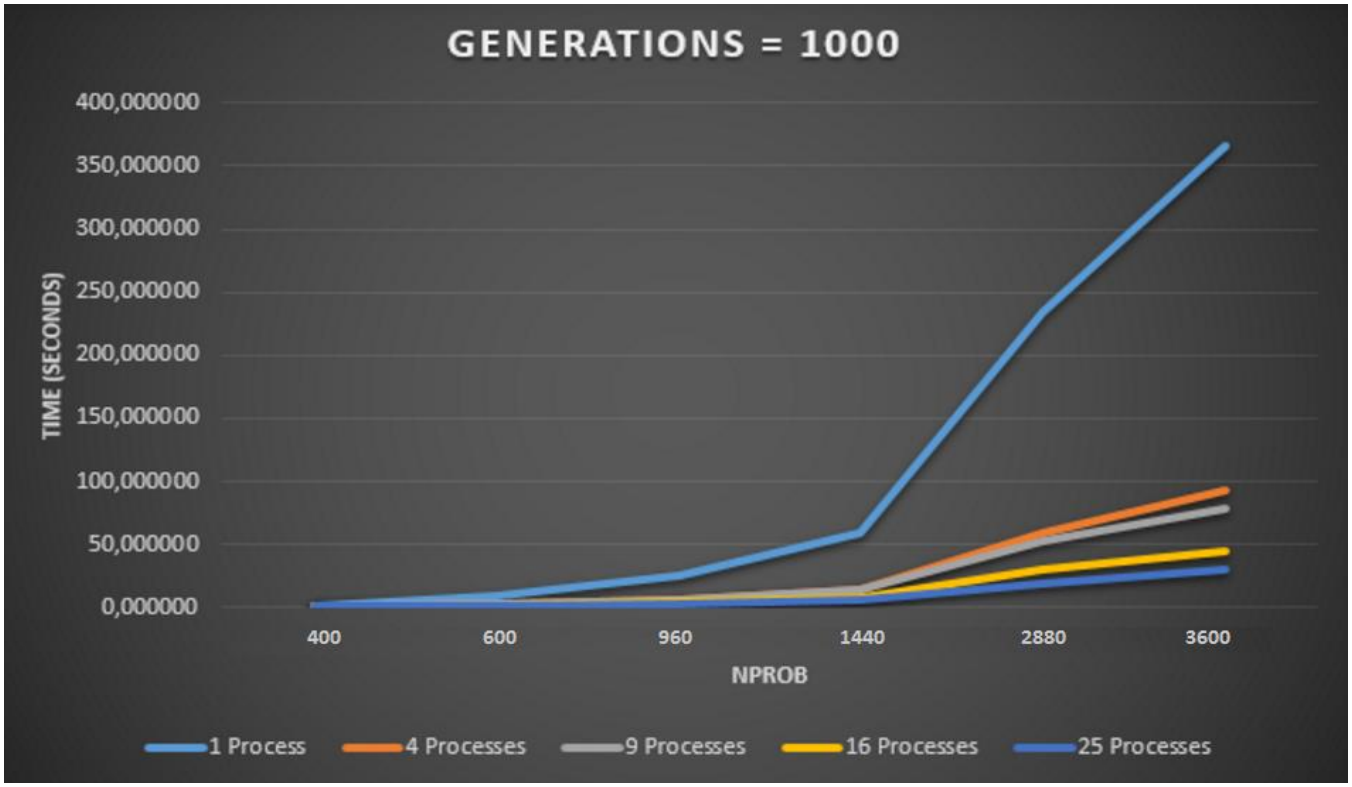


Συνεχίζοντας τις μετρήσεις για 500 Generations έχουμε τα αποτελέσματα που φαίνονται παραπάνω. Βλέπουμε ότι όλες οι τιμές κυμαίνονται σε σχεδόν 5πλάσια μεγέθη με τα αντίστοιχα στα 100 Generations. Τα αποτελέσματα που παίρνουμε είναι τα ίδια μιας και σε ίδιο NPROB όσο αυξάνουμε τις διεργασίες ο χρόνος μειώνεται.

Όσον αφορά τις μετρήσεις για 1000 Generations παρατηρούμε διπλασιασμό των τιμών σε σύγκριση με τα αντίστοιχα αποτελέσματα στα 500 Generations, όπως φαίνεται και παρακάτω:



	240	600	960	1440	2880	3600
1 Process	1,637283	10,179530	26,070290	58,585817	233,572842	365,364390
4 Processes	0,593310	2,744575	6,796728	15,062401	59,547138	92,556897
9 Processes	1,156114	2,981590	6,366451	13,707803	52,607337	79,068915
16 Processes	1,059474	1,998299	4,078038	7,918730	30,111418	45,395387
25 Processes	1,340743	1,852978	3,174220	5,530807	19,519974	29,670601



Για να γίνουν πιο ορατά τα αποτελέσματα υπολογίζονται παρακάτω το Speedup και το Efficiency για 1000 Generations.

Σύγκλιση

Με σύγκλιση στα 500 Generations με μέγεθος προβλήματος 1440 έχουμε:

MPI Con Generations = 500		
Time(secs)	NPROB	1440
Processses		
1		30,676887
4		16,781345
9		20,801518
16		29,595597
25		49,360601

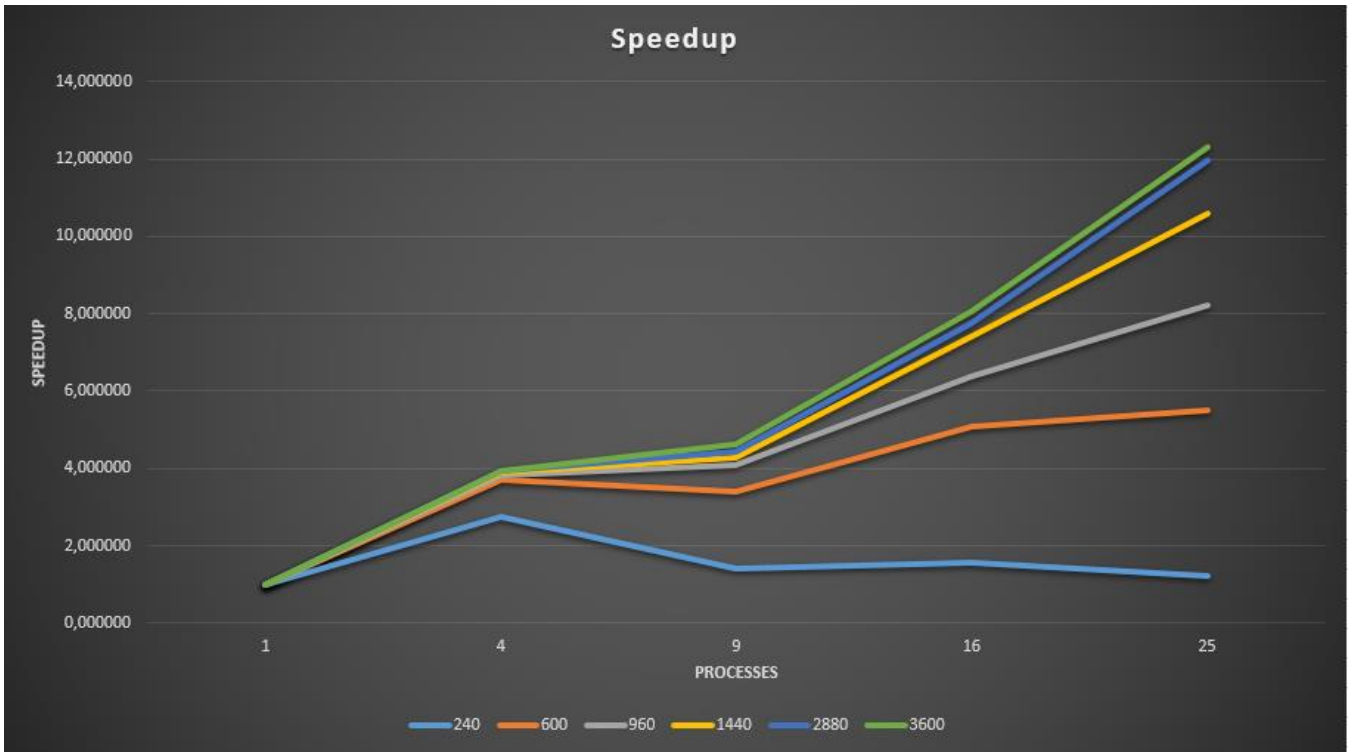
Οι χρόνοι είναι αρκετά αυξημένοι σε αντιστοιχία με τους χρόνους χωρίς σύγκλιση, λόγω των επιπλέον συγκρίσεων ενώ δε παρατηρείται καμία βελτίωση αφού στο life game στις περισσότερες περιπτώσεις το παιχνίδι συνεχίζεται για πάντα.

Speedup

Ως επιτάχυνση Speedup ορίζουμε το λόγο του χρόνου του σειριακού προγράμματος προς το χρόνο του παράλληλου. Με άλλα λόγια Speedup = Tserial / Tparallel , με Tserial ο σειριακός χρόνος και Tparallel ο παράλληλος χρόνος.

Έτσι για τα αντιστοιχα μεγέθη προβλήματος που χρησιμοποιήσαμε έχουμε τα εξής :

Speedup στα 1000 Generations		S=Tserial/Tparal					
Gens=1000	NPROB	240	600	960	1440	2880	3600
Processses							
1		1,000000	1,000000	1,000000	1,000000	1,000000	1,000000
4		2,759574	3,708964	3,835712	3,889540	3,922486	3,947457
9		1,416195	3,414128	4,094949	4,273903	4,439929	4,620835
16		1,545373	5,094098	6,392851	7,398385	7,756953	8,048492
25		1,221176	5,493605	8,213133	10,592634	11,965838	12,314021



Παρατηρούμε ότι με εξαίρεση το μικρό πρόβλημα 240x240 όπου η επικοινωνία μεταξύ των διεργασιών φαίνεται να επηρεάζει την απόδοση αφού για 4 διεργασίες βλέπουμε πολύ καλύτερη απόδοση απ ότι με 9 , 16 και 25 , κατα τα άλλα στα μεγαλύτερα προβήματα η συμπεριφορά του προγράμματος όσο αυξάνουν οι διεργασίες είναι όλο και καλύτερη.

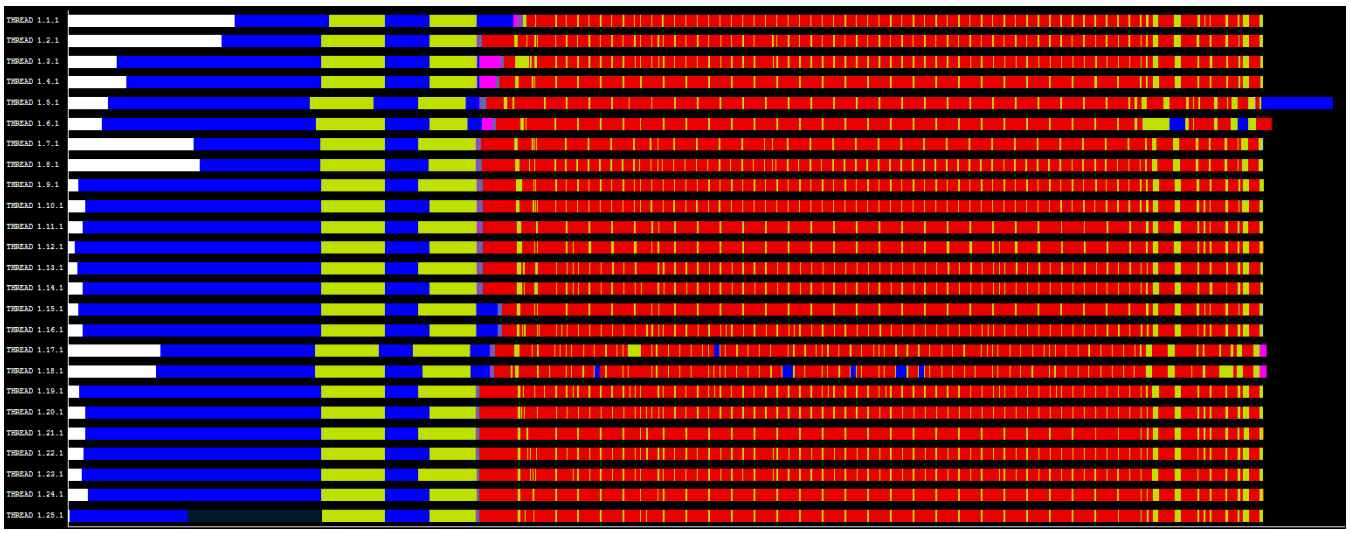
Efficiency

Επειδή το Toverhead (έστω ο παράπλευρος χρόνος), αυξάνεται ανάλογα με το πλήθος των διεργασιών p, περιμένουμε μείωση της επιτάχυνσης S όσο αυξάνεται το πλήθος των πυρήνων p, άρα και ο λόγος S / p μειώνεται. Ως αποτελεσματικότητα ορίζεται ο παραπάνω λόγος της επιτάχυνσης προς το πλήθος των πηρύνων. Συνεπώς E = S / p = (Tserial / Tparallel) / p = Tserial / (p * Tparallel) . Τα άντιστοιχα αποτελέσματα παρουσιάζονται παρακάτω:

Efficiency στα 1000 Generations			E=Tspeedup/Nprocesses				
Gens=1000	NPROB	240	600	960	1440	2880	3600
Processses							
1		1,000000	1,000000	1,000000	1,000000	1,000000	1,000000
4		0,689894	0,927241	0,958928	0,972385	0,980622	0,986864
9		0,157355	0,379348	0,454994	0,474878	0,493325	0,513426
16		0,096586	0,318381	0,399553	0,462399	0,484810	0,503031
25		0,048847	0,219744	0,328525	0,423705	0,478634	0,492561

Παραπάνω βλέπουμε το efficiency να μειώνεται με την αύξηση των διεργασιών, και αυτό οφείλεται στο overhead ενώ παρατηρούμε ότι για τον ίδιο αριθμό διεργασιών αυξάνεται η αποτελεσματικότητα όσο ανεβαίνει το μέγεθος του προβλήματος.

Η χρήση του paraver μας οδηγεί στο παρακάτω διάγραμμα:



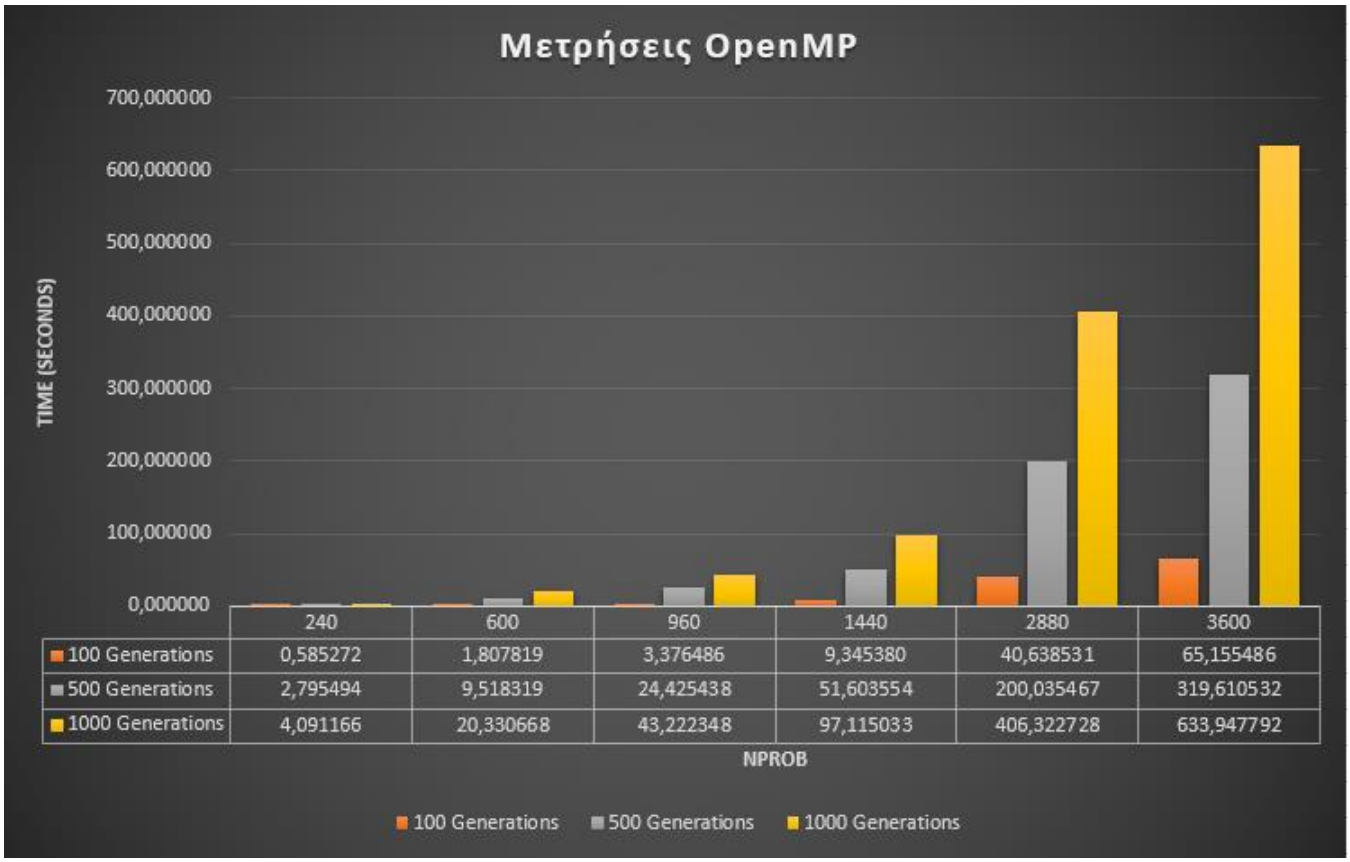
όπου οι χρωματισμοί αντιστοιχίζονται ως εξής:

Από την εκτέλεση του paraver πήραμε τις παρακάτω εικόνες οι οποίες συμφωνούν με την εκτέλεση των προγραμμάτων.
Παρατηρούμε ότι οι κάποιες διεργασίες αργούν πολύ για να δημιουργηθούν (άσπρο χρώμα), ενώ οι αρχικοποιήσεις και δεσμεύσεις μνήμης παρουσιάζονται με μπλε χρώμα. Τα πράσινα σημεία είναι διάφορες συναρτήσεις του MPI όπως MPI_Cart_create, MPI_Type_commit κλπ. Από το σημείο αυτό και έπειτα ξεκινάει η διαδικασία παραλαβής, υπολογισμού και αποστολής των δεδομένων. Το χρονικό διάστημα αυτό βλέπουμε ότι οι διεργασίες αλλάζουν καταστάσεις wait/ running (κόκκινο/μπλε χρώμα) για την επικοινωνία-υπολογισμό, και αν εστιάσουμε σε μικρές

περιοχές φαίνονται και οι αποστολές/παραλαβές των δεδομένων (ροζ/μωβ χρώματα).
Στο τέλος, γίνεται η αποδέσμευση των δομών κάθε διεργασίας (μπλε χρώμα) και γίνεται η εκτύπωση του αρχείου εξόδου (λαδί χρώμα).

Openmp

Συμπεριλήφθηκαν στο πρόγραμμα του MPI, στο αρχείο functions.c τα #pragma omp για να επιτύχουμε τη ζητούμενη παραλληλία. Έτσι διαθέτουμε τα δεδομένα του grid μας που διαχειρίζονται τα thread.



	240	600	960	1440	2880	3600
100 Generations	0,585272	1,807819	3,376486	9,345380	40,638531	65,155486
500 Generations	2,795494	9,518319	24,425438	51,603554	200,035467	319,610532
1000 Generations	4,091166	20,330668	43,222348	97,115033	406,322728	633,947792

Cuda

Όπως κανείς περιμένει η δουλεία γίνεται πιο αποτελεσματικά με τη χρήση της υπολογιστικής δύναμης της GPU. Έτσι έχουμε τα ακόλουθα αποτελέσματα :

Cuda					
Time(secs)	Generations	100	500	1000	100000
NPROB					
240		0,001767	0,008664	0,017527	1,733293
600		0,007434	0,036591	0,072489	7,215022
960		0,018129	0,090057	0,179443	17,920060
1440		0,039747	0,198267	0,395824	39,571133
2880		0,154231	0,770277	1,540560	154,026219
3600		0,243256	1,216621	2,433578	243,364028

**Δεν παρουσιάστηκαν σε Chart μιας και οι τιμές ήταν ιδιαίτερα μικρές.

Συμπεράσματα

Αισθητές γίνονται οι διαφορές ανάμεσα στο MPI σε σχέση με Openmp και Cuda αντίστοιχα μέσα από τα παρακάτω διαγράμματα για τα εκαστοτε μεγέθη προβλήματος στα 1000 Generations:

GENERATIONS = 1000	
NPROB	CUDA/MPI
240	0,013072602
600	0,03912027
960	0,056531368
1440	0,071567133
2880	0,078922
3600	0,082019842
GENERATIONS = 1000	
NPROB	OPENMP/MPI
240	3,051417013
600	10,971888
960	13,616683
1440	17,55892639
2880	20,81574125
3600	21,36619316

Βλέπουμε ότι το το MPI υπερτερεί σαφώς του OpenMP όπως είναι λογικό μιας και το μέγεθος του προβλήματος είναι μεγάλο ενώ το πρόγραμμα CUDA είναι μια τάξη μεγέθους γρηγορότερο από το αντίστοιχο MPI.

Βιβλιογραφία

Διαφάνειες και Ηλεκτρονικές Διαλέξεις του μαθήματος

http://mpi.deino.net/mpi_functions

http://www.mpich.org/static/docs/v3.2/www3

http://beige.ucs.indiana.edu/I590/node100.html