ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Εξεταστική Περίοδος Σεπτεμβρίου
2013 - 2014

Εργασία Παράλληλων Συστημάτων

ΟΝΟΜΑ: Γαλούνη Κωνσταντίνα Α.Μ.: 1115201000034
ΟΝΟΜΑ: Γιαννακέλος Κωνσταντίνος Α.Μ.: 1115201000029

ΔΙΔΑΣΚΩΝ: Κοτρώνης Ιωάννης

ΑΘΗΝΑ – 2014

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

Υλοποίηση

Η εργασία αναπτύχθηκε σύμφωνα με τα απαιτούμενα της εκφώνησης. Συγκεκριμένα, επιτυγχάνεται παραλληλία μέσω MPI, μέσω OpenMP και μέσω CUDA.

MPI: Σε αυτήν την περίπτωση η παραλληλία επιτυγχάνεται μέσω της επικοινωνίας διεργασιών οι οποίες συνδέονται με καρτεσιανή τοπολογία. Συγκεκριμένα, κάθε διεργασία αρχικοποιεί ξεχωριστά μόνη της το μέρος των δεδομένων που πρόκειται να επεξεργαστεί, δημιουργώντας υποπίνακες, συνεπώς καταργείται η έννοια master – slaves. Έπειτα, στέλνει/λαμβάνει μέσω MPI_send και MPI_receive τα οριακά δεδομένα στους γείτονές της, πάνω, κάτω, δεξιά, αριστερά όταν αυτοί υπάρχουν. Προκειμένου να αποφευχθεί η αντιγραφή δεδομένων που είναι δαπανηρή στην αποστολή στηλών του υποπίνακα, χρησιμοποιούνται datatypes. Τέλος, για περαιτέρω βελτίωση η συνάρτηση update χωρίζεται σε δύο τμήματα, την inner_update και την outer_update. Η πρώτη καλείται πριν τα MPI_wait εφόσον επεξεργάζεται τα εσωτερικά στοιχεία του πίνακα που δεν επηρεάζονται από τα γειτονικά δεδομένα, ενώ η δεύτερη καλείται μετά την ολοκλήρωση των επικοινωνιών.

OpenMP: Με αυτόν τον τρόπο επιτυγχάνουμε παραλληλία στην update μέσω των #pragma omp parallel και #pragma omp for. Πλέον δεν απαιτείται επικοινωνία διεργασιών, καθώς επίσης η update παραμένει μία ολοκληρωμένη. Τέλος, δεν έχουμε χωρισμό σε υποπίνακες, αλλά ένα μεγάλο πλέγμα του οποίου τα δεδομένα διαχειρίζονται τα νήματα.

CUDA: Σε αυτήν την υλοποίηση η παραλληλία επιτυγχάνεται και πάλι στο εσωτερικό της update. Για τη χρήση της κάρτας γραφικών η δέσμευση μνήμης μέσω cudaMalloc και η μεταφορά δεδομένων γίνεται μέσω cudaMemcpyHostToDevice και cudaMemcpyHostToDevice. Το πρόβλημα χωρίζεται σε υποπροβλήματα και δημιουργούνται NYPROB-2 blocks με NXPOB-2 threads σε κάθε block, ώστε κάθε thread επεξεργάζεται ένα τετράγωνο από το πλέγμα.

Μεταγλώττιση

MPI: mpicc –o mpi_exec mpi_heat2D.c –lm

Σημειώνεται ότι για την εκτέλεση πρέπει να εκκινήσουμε δαίμονες στα μηχανήματα Linux του Τμήματος χρησιμοποιώντας την εντολή:
mpdboot -v -c -n <NumberOfHosts> -r rsh -f mpd.hosts –ncpus=2
όπου mpd.hosts είναι ένα αρχείο που περιέχει τα ονόματα των μηχανημάτων.
Για την εκτέλεση του προγράμματος αξιοποιώντας το δακτύλιο των δαιμόνων χρησιμοποιούμε την εντολή:
mpiexec –machinefile machines –np <NumberOfProcesses> mpi_exec
όπου machines είναι ένα άλλο αρχείο που περιέχει τα ίδια ονόματα των μηχανών του mpd.hosts ακολουθούμενα από <:2> που δηλώνει ότι θα ξεκινούν 2 διεργασίες σε κάθε μηχανή (έχουν 2 πυρήνες).
Για τον τερματισμό όλων των δαιμόνων χρησιμοποιούμε την εντολή:
mpdallexit

OpenMP: gcc -o openmp omp.c –fopenmp

Στην περίπτωση της σειρακής εκτέλεσης απλά παραλείπουμε το σχετικό με το OpenMP flag, -fopenmp και χρησιμοποιούμε την εντολή:
gcc –o serial omp.c

CUDA: make

Σημειώνεται πως η εντολή make μπορεί να χρησιμοποιηθεί εφόσον έχουμε δημιουργήσει makefile, ωστόσο πρέπει να βρισκόμαστε εντός του φακέλου. Επίσης ο φάκελος common πρέπει να βρίσκεται μαζί με τα υπόλοιπα αρχεία στον ίδιο φάκελο λόγω του makefile. Για την εκτέλεση έχει υλοποιηθεί και κατάλληλο script, το οποίο τρέχει το εκτελέσιμο από το φάκελο cuda μέσω της εντολής:
qsub script.sh

Αποτελέσματα Εκτελέσεων

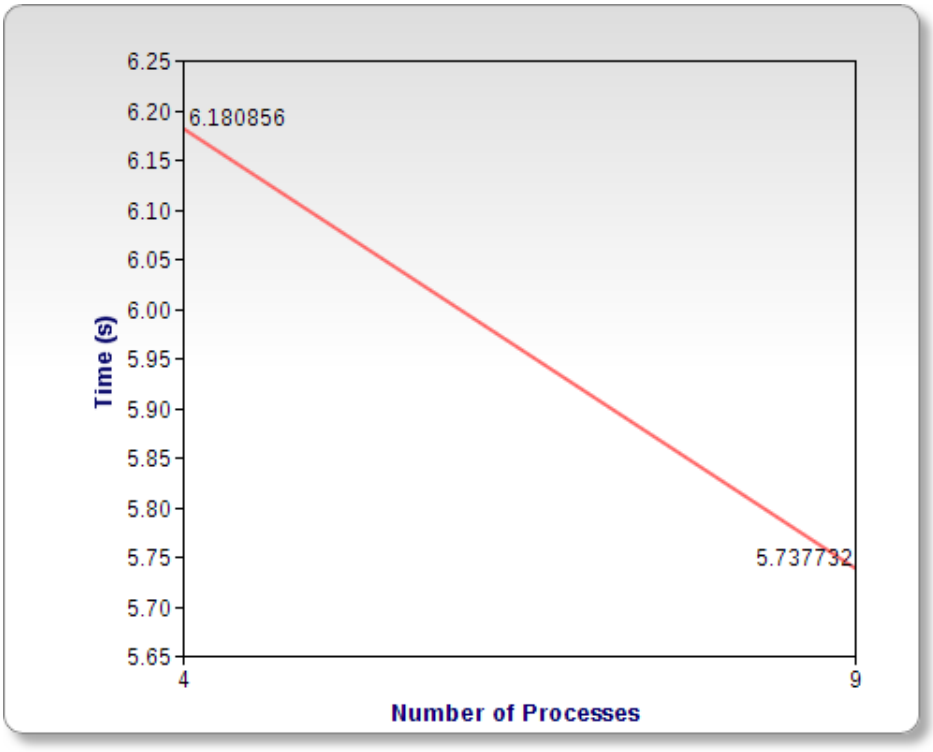
Αρχικό Πρόγραμμα: Οι διεργασίες που εξετάστηκαν είναι 4 και 9, δεδομένου του αρχικού περιορισμού για τον ελάχιστο και το μέγιστο αριθμό.

STEPS = 100

	SIZE	240	600	960
PROCESS	**secs**			
4		0.446556	2.517721	6.180856
9		0.438181	2.345009	5.737732

Το πρόγραμμα που δόθηκε έχει εμφανώς χειρότερους χρόνους σε σχέση με το πρόγραμμα MPI Που αναπτύχθηκε στα πλαίσια της άσκησης. Αυτό συμβαίνει λόγω της ανάπτυξης, αφού εδώ δεν έχουμε πλήρη επικοινωνία διεργασιών, ούτε τοπολογία, καθώς επίσης έχουμε την έννοια του master process, όπου μία διεργασία αναλαμβάνει την αρχικοποίηση και το διαμοιρασμό των δεδομένων, όπως και τη συγκέντρωση των αποτελεσμάτων, ενέργειες αρκετά δαπανηρές. Παρακάτω παρατίθεται ένα διάγραμμα για διάσταση 960 για τη σχέση χρόνου-διεργασιών:

Size = 960

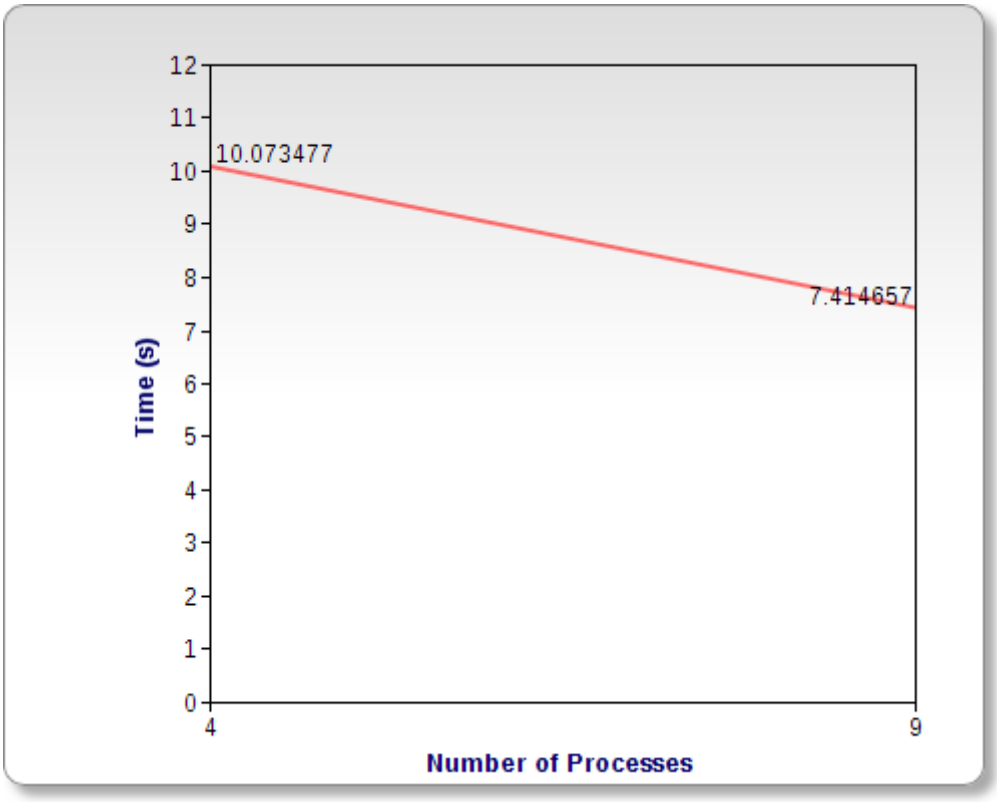


STEPS = 500

	SIZE	240	600	960
PROCESS	**secs**			
4		0.785575	4.094646	10.073477
9		0.635038	3.079757	7.414657

Για 500 επαναλήψεις, ο χρόνος είναι και πάλι αυξημένος. Όσο αυξάνονται οι διεργασίες ο χρόνος βελτιώνεται, ενώ όσο αυξάνεται το μέγεθος του προβλήματος, ο χρόνος εκτέλεσης αυξάνεται. Ωστόσο, τα αποτελέσματα δεν είναι 5 φορές μεγαλύτερα σε σχέση με τα 100 βήματα. Αυτό οφείλεται και πάλι στην κατασκευή του προβλήματος, αφού οι διεργασίες απαιτούν μεν περισσότερο χρόνο για την επικοινωνία τους και καθυστερούν να στείλουν το τελικό αποτέλεσμα, αλλά η αποστολή αυτή γίνεται μόνο μία φορά. Παρακάτω παρατίθεται ένα διάγραμμα για διάσταση 960 για τη σχέση χρόνου-διεργασιών:

Size = 960

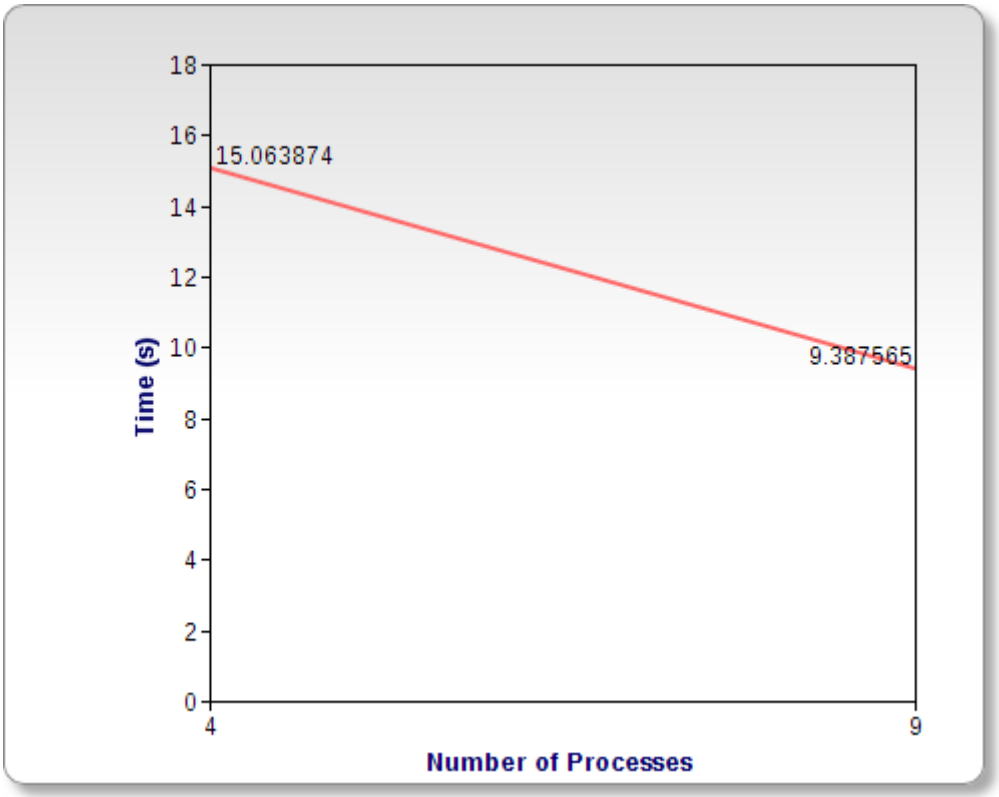


STEPS = 1000

	SIZE	240	600	960
PROCESS	**Secs**			
4		1.156911	6.154844	15.063874
9		0.884147	3.977072	9.387565

Αντίστοιχη συμπεριφορά με αυτήν που επεξηγήθηκε και παραπάνω για 100 και για 500 βήματα παρουσιάζει το πρόγραμμα και για 1000 βήματα. Και πάλι οι χρόνοι είναι αυξημένοι, αλλά όχι διπλάσιοι. Επίσης, όπως ήδη αναφέρθηκε για αύξηση των διεργασιών ο χρόνος μειώνεται, ενώ για αύξηση του μεγέθους πλέγματος ο χρόνος αυξάνεται. Παρακάτω παρατίθεται ένα διάγραμμα για διάσταση 960 για τη σχέση χρόνου-διεργασιών:

Size = 960



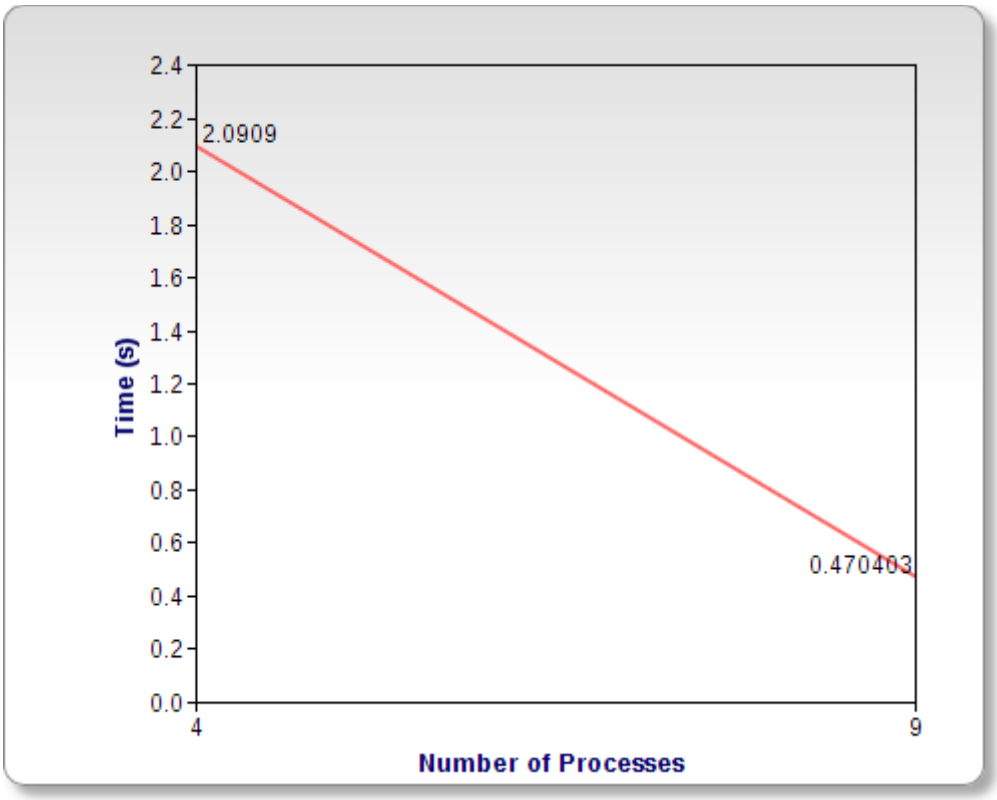
MPI: Ο μέγιστος αριθμός διεργασιών που εξετάσαμε είναι 36, καθώς τα διαθέσιμα μηχανήματα Linux ήταν 19 και καθένα από αυτά υποστηρίζει 2 διεργασίες.

STEPS = 100

	SIZE	240	600	960	1440	1920	2880	3600
PROCESS	**Secs**							
1		0.249488	1.569727	4.013262	9.031954	9.031808	36.087689	56.452353
4		0.070142	0.404114	2.090900	2.317835	2.317572	9.137623	14.270561
9		0.042941	0.190662	0.470403	1.054457	1.056758	4.093983	6.370399
16		0.036198	0.119076	0.276674	0.611926	0.634448	2.332392	3.626058
25		0.031330	0.083835	0.184558	0.389722	0.598859	1.513570	2.336483
36		0.035812	0.070062	0.137823	0.280567	0.283930	1.074319	3.229596

Παρατηρούμε από τα αποτελέσματα ότι για κάθε σταθερή τιμή διάστασης αυξάνοντας τον αριθμό των διεργασιών έχουμε μείωση στο χρόνο εκτέλεσης. Αυτό είναι δικαιολογημένο, αφού έτσι αυξάνεται η παραλληλία, την οποία εκμεταλλευόμαστε μέσω της επικοινωνίας και της τοπολογίας διεργασιών. Στην περίπτωση που κρατήσουμε τον αριθμό των διεργασιών σταθερό και αυξήσουμε τη διάσταση του πλέγματος, ο χρόνος γίνεται πολύ μεγαλύτερος, αφού τα δεδομένα προς επεξεργασία αυξάνονται. Ο χρόνος είναι μικρός, δεδομένου ότι ο αριθμός των επαναλήψεων είναι μόλις 100. Παρακάτω παρατίθεται ένα διάγραμμα για διάσταση 960 για τη σχέση χρόνου-διεργασιών:

Size = 960

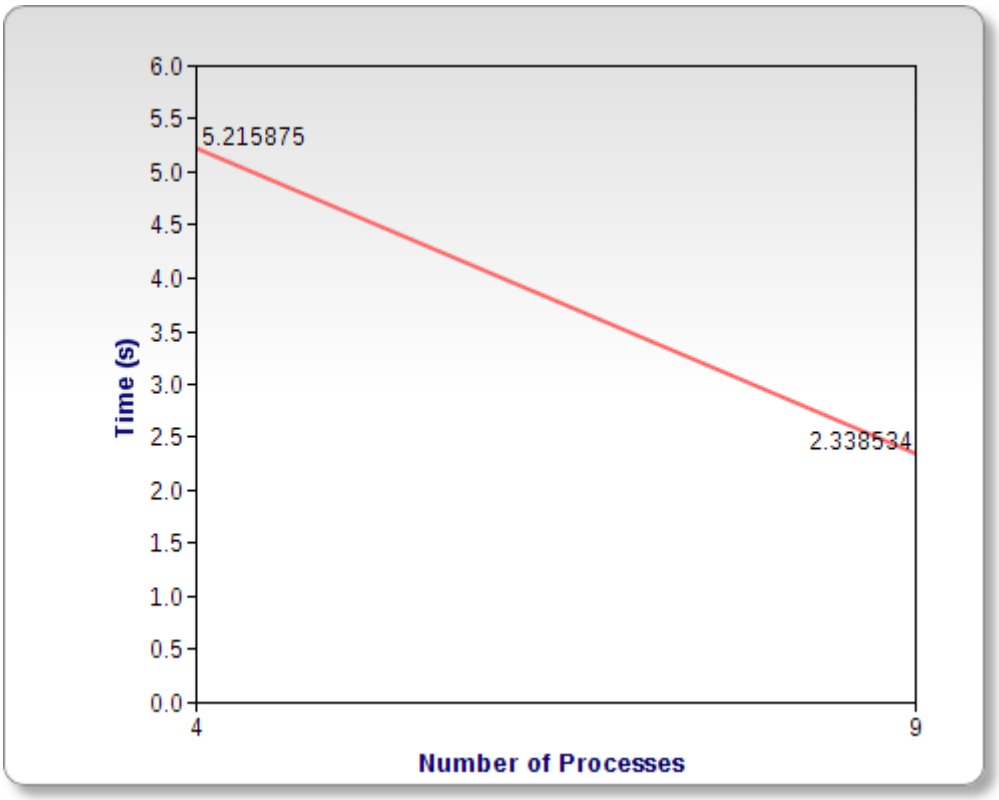


STEPS = 500

	SIZE	240	600	960	1440	1920	2880	3600
PROCESS	**Secs**							
1		1.251072	7.886765	20.081190	45.227720	80.266512	180.665020	282.535250
4		0.344268	2.020231	5.215875	11.483457	20.347032	45.526928	71.057375
9		0.201182	0.934797	2.338534	5.250179	9.185002	20.474208	31.854259
16		0.143211	0.561464	1.346673	3.015414	5.266189	11.611970	18.006926
25		0.127980	0.389355	0.884059	1.914377	3.434478	7.550055	11.641613
36		0.192198	0.568247	0.641436	1.352451	2.359492	5.295695	8.155729

Αυξάνοντας τον αριθμό των επαναλήψεων σε 500 όπως φαίνεται και από τον παραπάνω πίνακα οι χρόνοι εκτέλεσης σχεδόν πενταπλασιάζονται. Αυτό είναι αναμενόμενο, αφού οι διεργασίες εκτελούν ακριβώς τις ίδιες εντολές για περισσότερα όμως βήματα. Η συμπεριφορά του χρόνου είναι ίδια με πριν, αφού αυξάνοντας τις διεργασίες για σταθερό μέγεθος προβλήματος η παραλληλία αυξάνεται και συνεπώς ο χρόνος μειώνεται. Αντίστοιχα, για σταθερό αριθμό διεργασιών με αυξανόμενο μέγεθος πλέγματος, το πλήθος δεδομένων προς επεξεργασία αυξάνεται, άρα ο χρόνος που απαιτείται είναι πολύ περισσότερος. Παρακάτω παρατίθεται ένα διάγραμμα για διάσταση 960 για τη σχέση χρόνου-διεργασιών:

Size = 960

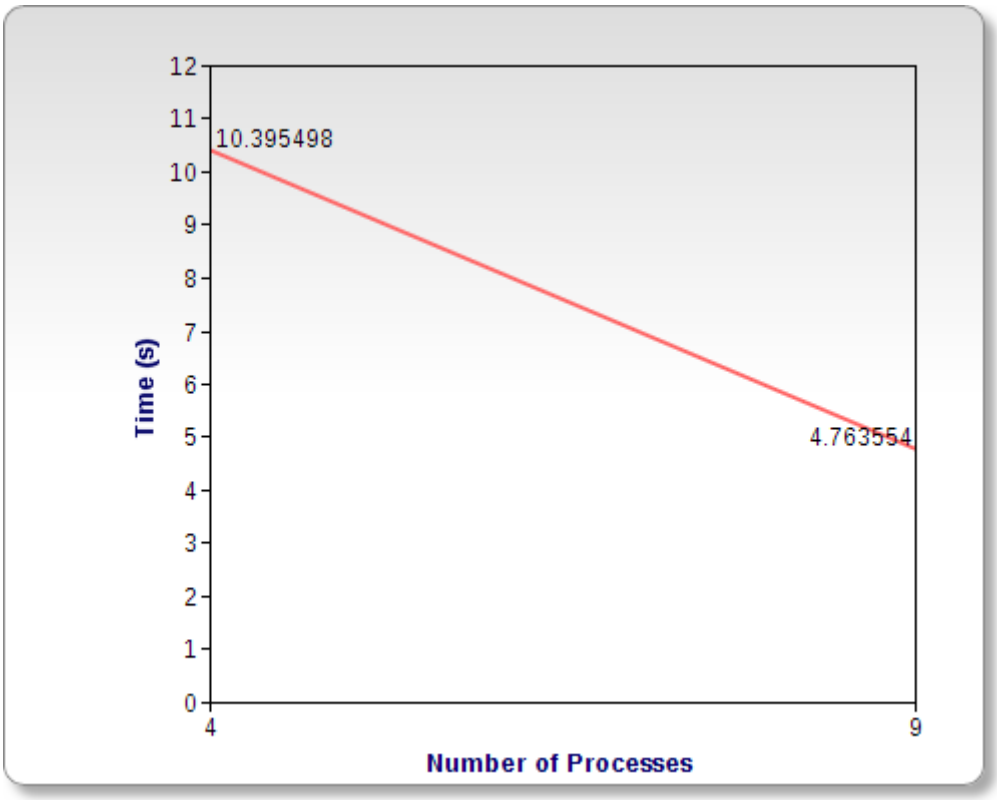


STEPS = 1000

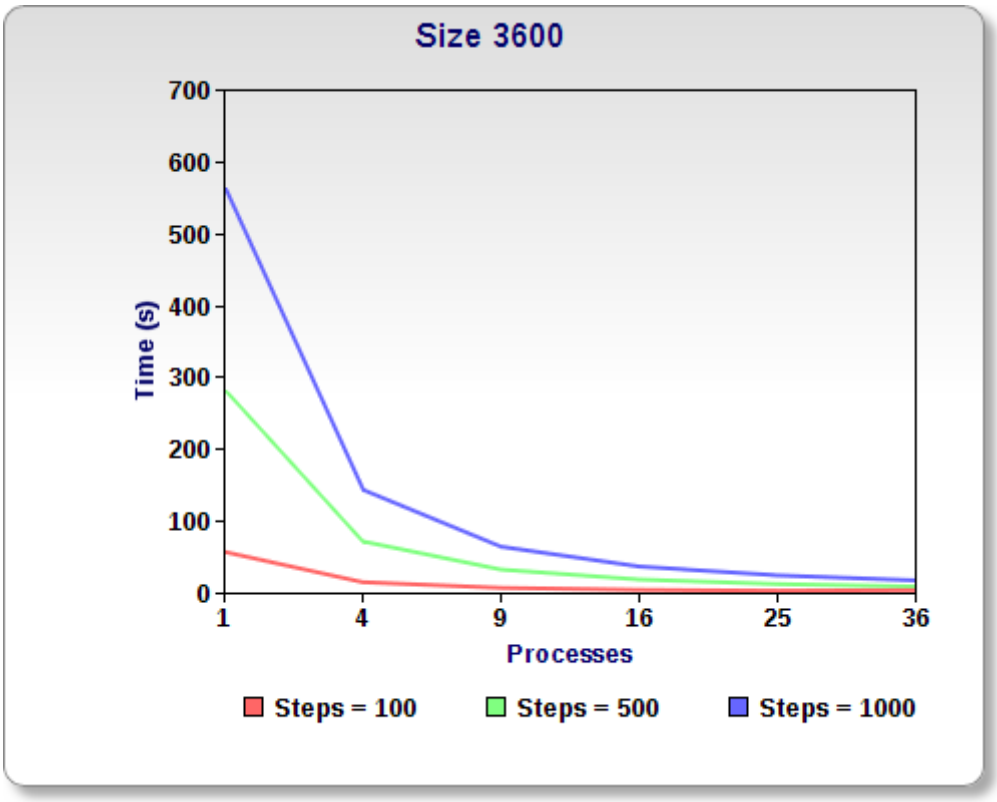
	SIZE	240	600	960	1440	1920	2880	3600
PROCESS	**secs**							
1		2.499496	15.747512	40.239147	90.402523	160.563936	361.038719	564.117146
4		0.680862	4.056747	10.395498	22.987192	40.616008	92.706169	142.975456
9		0.388163	1.885905	4.763554	10.590868	18.370583	41.623135	63.682888
16		0.282772	1.106742	2.661854	6.036482	10.544102	23.616397	36.080738
25		0.244472	0.771057	1.765621	3.822885	6.875222	15.396041	23.784373
36		0.232955	0.588817	1.273684	2.691409	4.717230	10.988511	16.794825

Εφόσον ο αριθμός των επαναλήψεων διπλασιάζεται όπως φαίνεται και από τον παραπάνω πίνακα οι χρόνοι εκτέλεσης σχεδόν διπλασιάζονται και αυτοί σε σχέση με τους προηγούμενους. Όπως και πριν τα αποτελέσματα είναι αναμενόμενα, διότι οι διεργασίες εκτελούν ακριβώς τις ίδιες εντολές για περισσότερα όμως βήματα. Η συμπεριφορά του χρόνου παραμένει η ίδια, δηλαδή αυξάνοντας τις διεργασίες για σταθερό μέγεθος προβλήματος η παραλληλία αυξάνεται και συνεπώς ο χρόνος μειώνεται. Αντίστοιχα, για σταθερό αριθμό διεργασιών με αυξανόμενο μέγεθος πλέγματος, το πλήθος δεδομένων προς επεξεργασία αυξάνεται, άρα ο χρόνος που απαιτείται είναι πολύ περισσότερος.

Size = 960



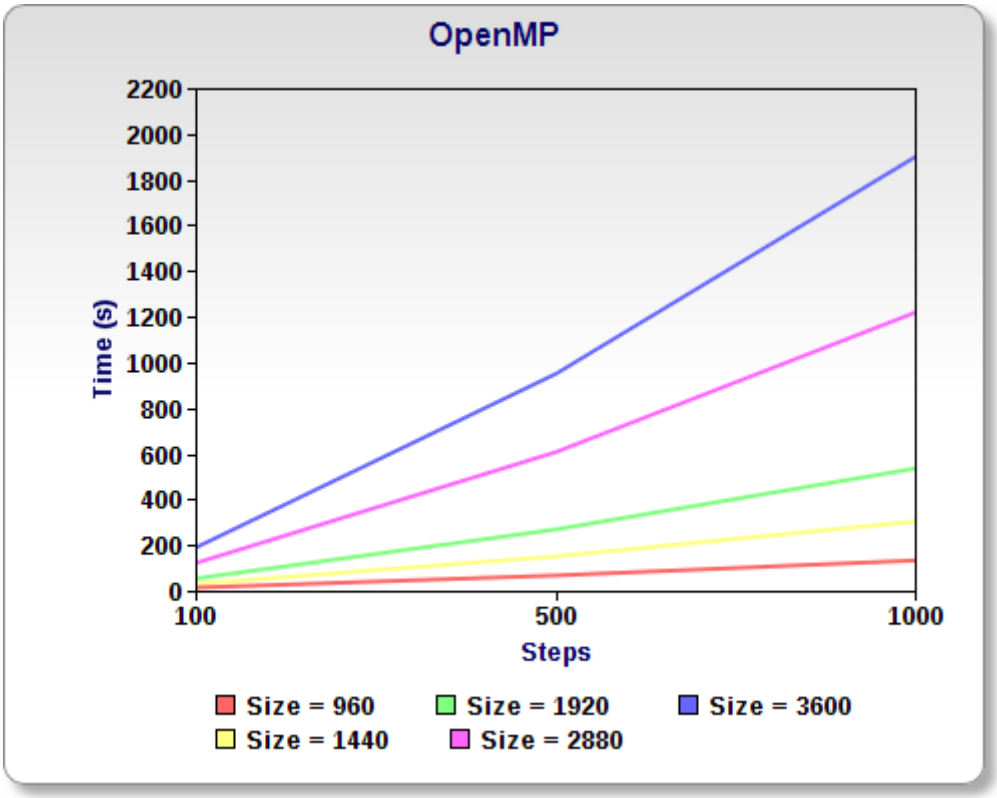
Στη συνέχεια παρουσιάζουμε ένα διάγραμμα για τη σύγκριση του αρχικού δοθέντος με το MPI πρόγραμμα για μέγεθος προβλήματος 3600x3600:



OpenMP:

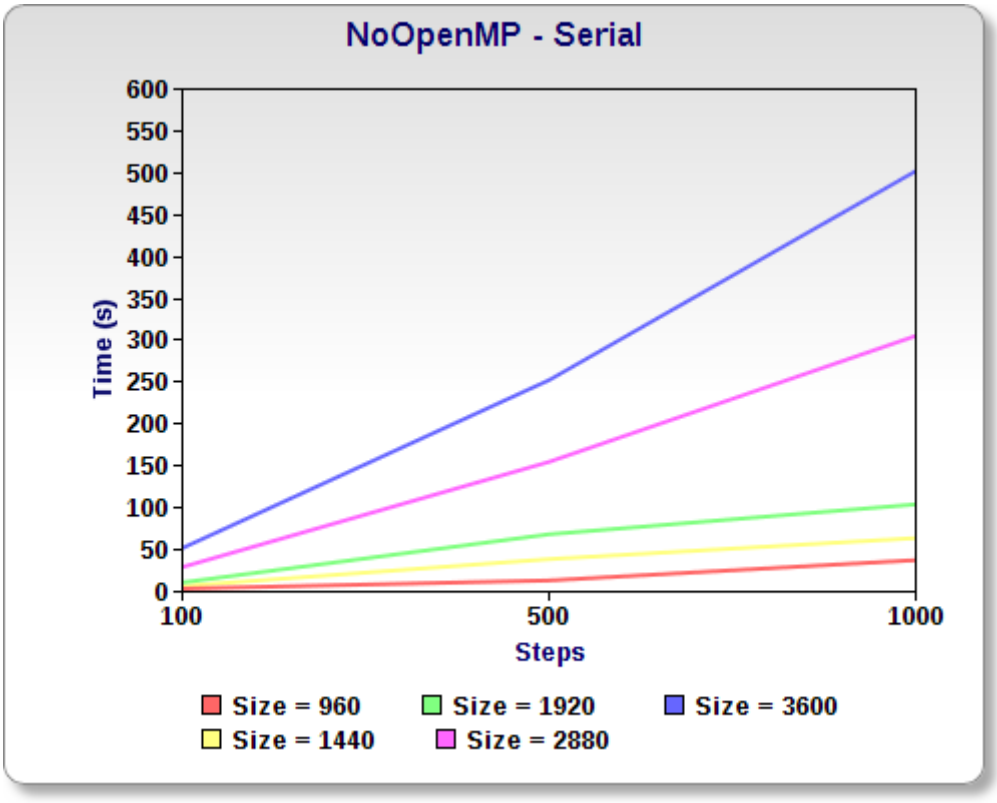
OpenMp	Size	240	600	960	1440	1920	2880	3600
Steps	secs							
100		0.813745	5.262573	13.5556	30.0790	53.5995	121.604	189.947
500		4.067187	26.29101	66.8084	150.487	268.442	607.023	949.735
1000		8.135172	51.7372	132.549	303.594	535.331	1218.671	1900.02

Στην περίπτωση του OpenMp δεν έχουμε πολλές διεργασίες, αλλά πολλαπλά νήματα. Συγκεκριμένα, οι εκτελέσεις έγιναν στον server με διεύθυνση 195.134.67.205 που περιγράφεται στο εργαστήριο. Παρατηρούμε ότι η αύξηση των επαναλήψεων στη δομή επανάληψης του προγράμματος, προκαλεί ανάλογη αύξηση στο χρόνο εκτέλεσης. Όσο αυξάνεται το μέγεθος του προβλήματος, αυξάνεται και ο χρόνος, αφού αυξάνεται ο φόρτος λόγω των περισσότερων δεδομένων προς επεξεργασία. Ο χρόνος είναι εμφανώς μικρότερος από το αρχικό δοθέν πρόγραμμα, ωστόσο υστερεί σε σχέση με την MPI έκδοση που αναπτύχθηκε για την άσκηση. Το παρακάτω διάγραμμα παρουσιάζει τη συμπεριφορά αυτή ξεκάθαρα:



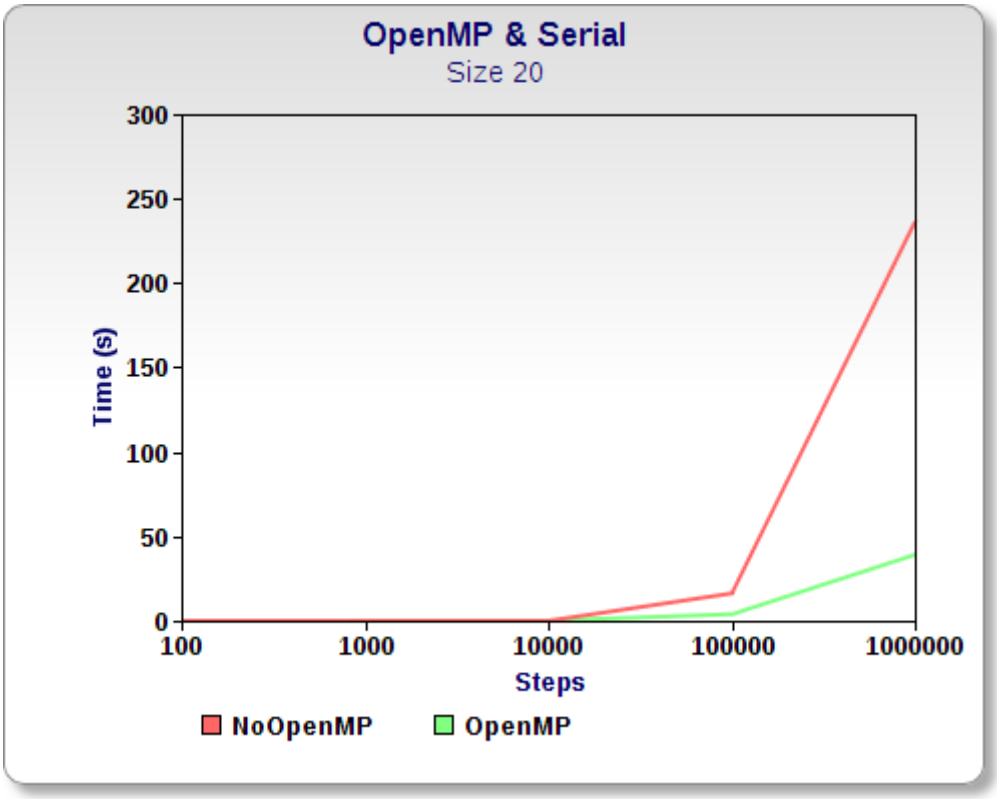
Serial	**Size**	240	600	960	1440	1920	2880	3600
Steps	**Secs**							
100		0.152771	0.949789	2.44471	5.51125	9.80874	28.1819	50.8031
500		0.754594	4.750386	12.2244	37.6866	67.3382	153.787	251.412
1000		1.507346	9.505539	36.4671	62.7846	103.033	304.221	500.957

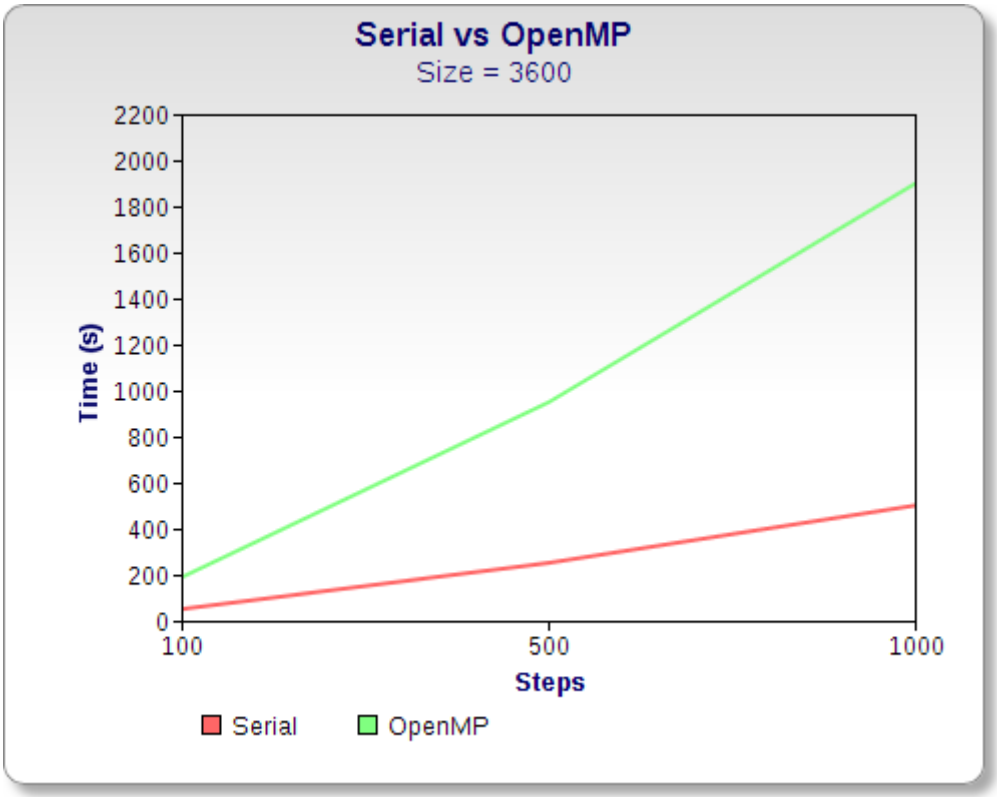
Οι χρόνοι αυτοί αφορούν στην εκτέλεση του ίδιου με το παραπάνω προγράμματος με τη διαφορά ότι μεταγλωττίζεται χωρίς το απαραίτητο flag για το openMp. Παραδόξως οι χρόνοι είναι πολύ καλύτεροι σε αυτήν την περίπτωση. Το παραπάνω μπορεί να επεξηγηθεί αν λάβουμε υπόψη το κόστος συγχρονισμού ανάμεσα στα νήματα. Και σε αυτήν την εκτέλεση η αύξηση των επαναλήψεων προκαλεί αύξηση του χρόνου, όπως επίσης και η αύξηση της διάστασης του πλέγματος. Παρακάτω παραθέτουμε το σχετικό διάγραμμα για τη σειριακή εκτέλεση:

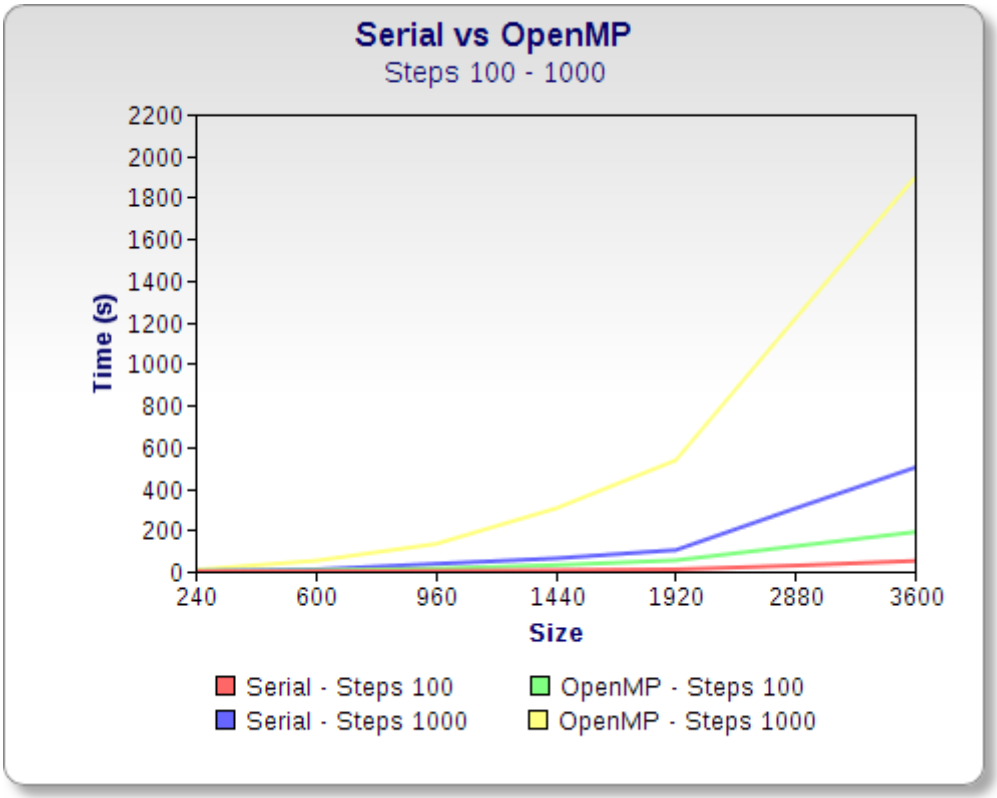


Steps	Size	No-OMP	OMP
	20		
100		0.0013	0.0047
1000		0.012	0.040
10000		0.010	0.039
100000		16.2587	3.8903
1000000		236.563	39.229

Ο παραπάνω πίνακας παρουσιάζει τα αποτελέσματα εκτελέσεων σειριακού και μη OpenMp για μέγεθος πλέγματος ίσο με 20x20. Η σύγκριση αυτή παρατίθεται προκειμένου να φανεί η διαφορά με την αύξηση των βημάτων. Όπως μπορούμε να παρατηρήσουμε, για μικρά μεγέθη επαναλήψεων η σειριακή μεταγλώττιση είναι πολύ περισσότερο αποδοτική, όμως για 100000 και περισσότερα βήματα η εκτέλεση του OpenMp είναι πολύ περισσότερο αποδοτική. Συνεπώς φαίνεται πως το OpenMp για λίγα βήματα προκαλεί μεγάλο overhead λόγω του συγχρονισμού και γι' αυτό είναι πολύ πιο αργό από το σειριακό. Η σύγκριση του OpenMP με το σειριακό πρόγραμμα παρουσιάζεται στα παρακάτω διαγράμματα:



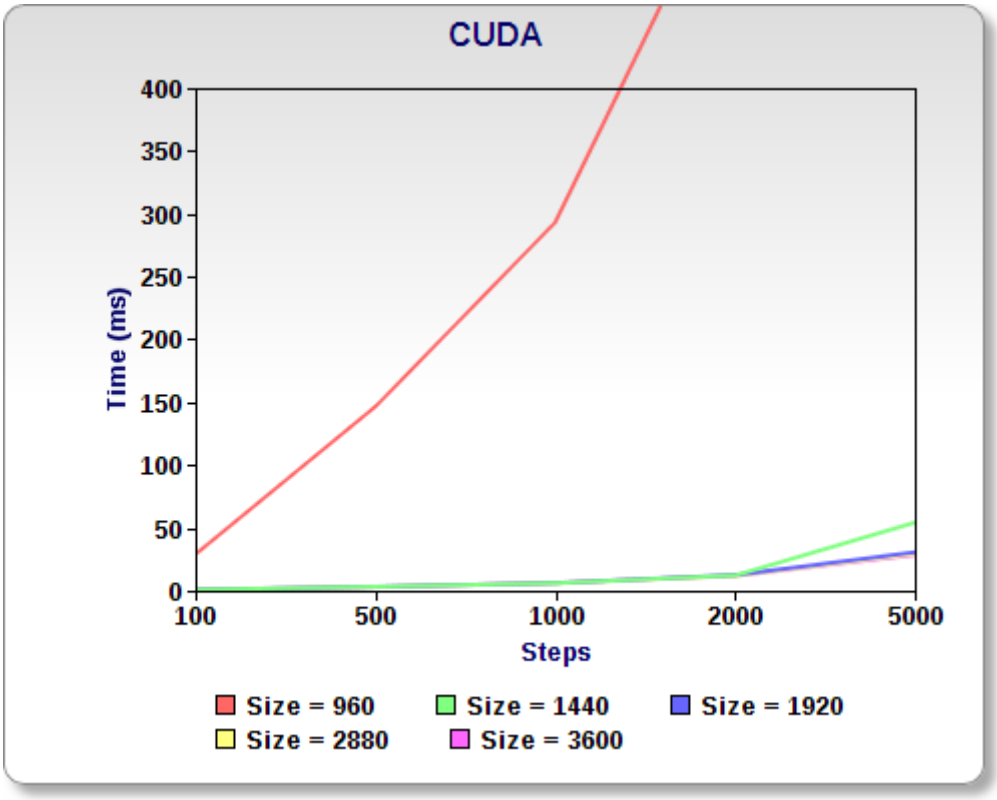

Serial vs OpenMP
Size = 3600
Time (s)
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
100
500
1000
Steps
Serial
OpenMP


Serial vs OpenMP
Steps 100 - 1000
Time (s)
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
240
600
960
1440
1920
2880
3600
Size
Serial - Steps 100
OpenMP - Steps 100
Serial - Steps 1000
OpenMP - Steps 1000

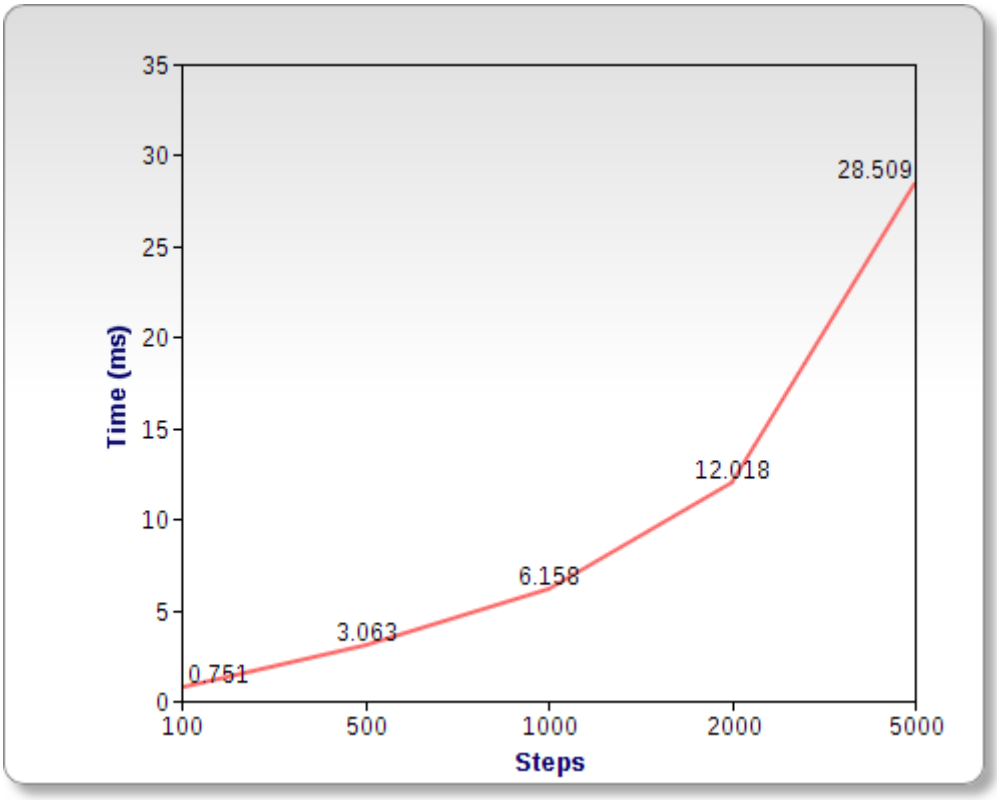
CUDA:

CUDA	Size	240	600	960	1440	1920	2880	3600
Steps	**millisecs**							
100		2.2120	9.8240	29.5460	0.7650	0.7410	0.7720	0.7510
500		10.5420	48.1890	147.0800	3.2010	3.1680	3.0650	3.0630
1000		21.0040	96.3590	293.8390	6.0080	6.1050	6.0570	6.1580
2000		42.0480	192.2090	587.4670	12.2760	12.5110	11.9220	12.0180
5000		105.2770	480.6630	3185.393	54.4530	30.6780	29.3680	28.5090

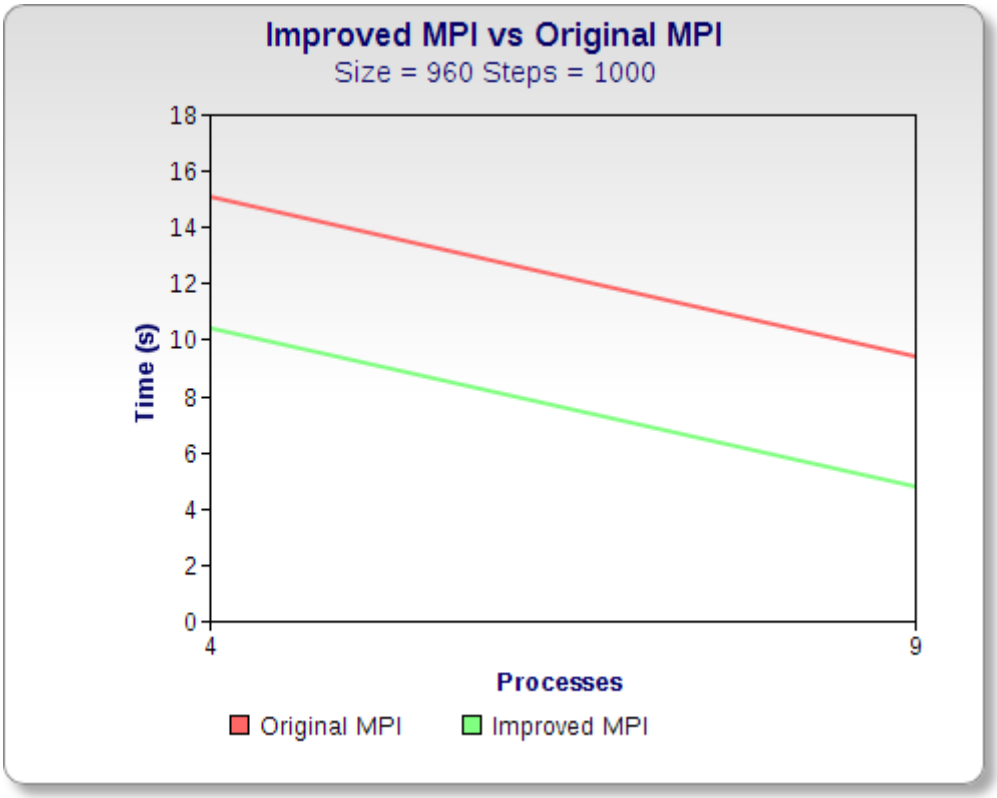
Οι χρόνοι στο πρόγραμμα με χρήση Cuda παρουσιάζει τους μικρότερους χρόνους από όλα. Αυτό συμβαίνει γιατί πλέον τον έλεγχο αναλαμβάνει η GPU. Οι δοκιμές έχουν γίνει όπως και για το OpenMp στον server που δίνεται και έχει κάρτα γραφικών υποστηριζόμενη από το περιβάλλον CUDA (NVidia GTX – 480). Και πάλι οποιαδήποτε αύξηση στα βήματα προκαλεί ανάλογη αύξηση στο χρόνο, όπως επίσης και η αύξηση του μεγέθους του προβλήματος μέχρι τη διάσταση ίση με 960 έχει ως αποτέλεσμα την αύξηση του χρόνου. Ωστόσο, από τη στιγμή που η διάσταση αυξάνεται, από το 1440x1440 κι έπειτα παρατηρείται τρομερή βελτίωση στο χρόνο εκτέλεσης σε σχέση με πριν και ανεξάρτητα από τη διάσταση οι χρόνοι είναι σχεδόν ίδιοι για σταθερό αριθμό επαναλήψεων. Τα διαγράμματα που περιγράφουν τη συμπεριφορά του CUDA είναι τα εξής:

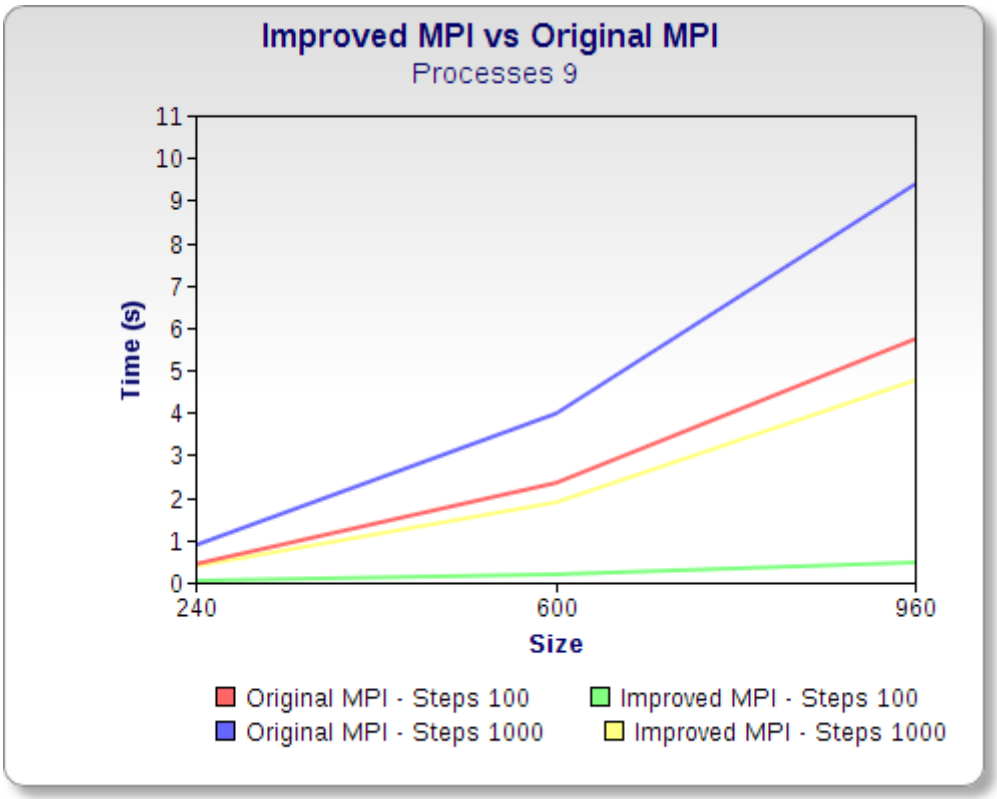

CUDA
400
350
300
250
200
150
100
50
0
Time (ms)
100
500
1000
2000
5000
Steps
Size = 960
Size = 1440
Size = 1920
Size = 2880
Size = 3600

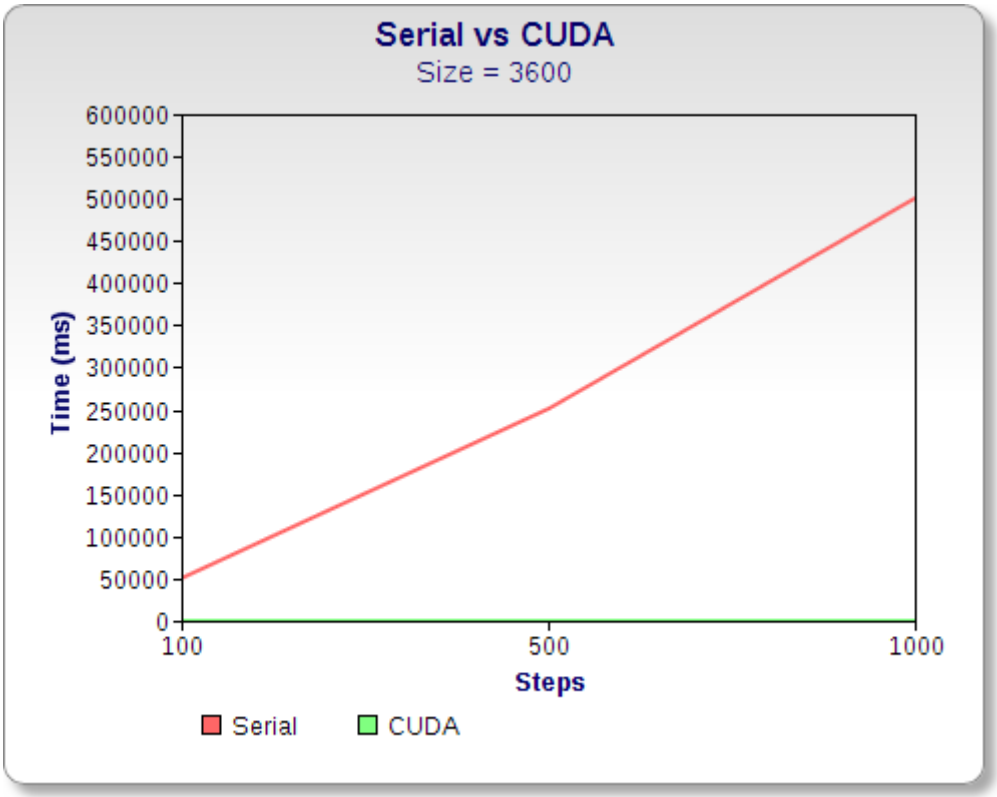
Size = 3600

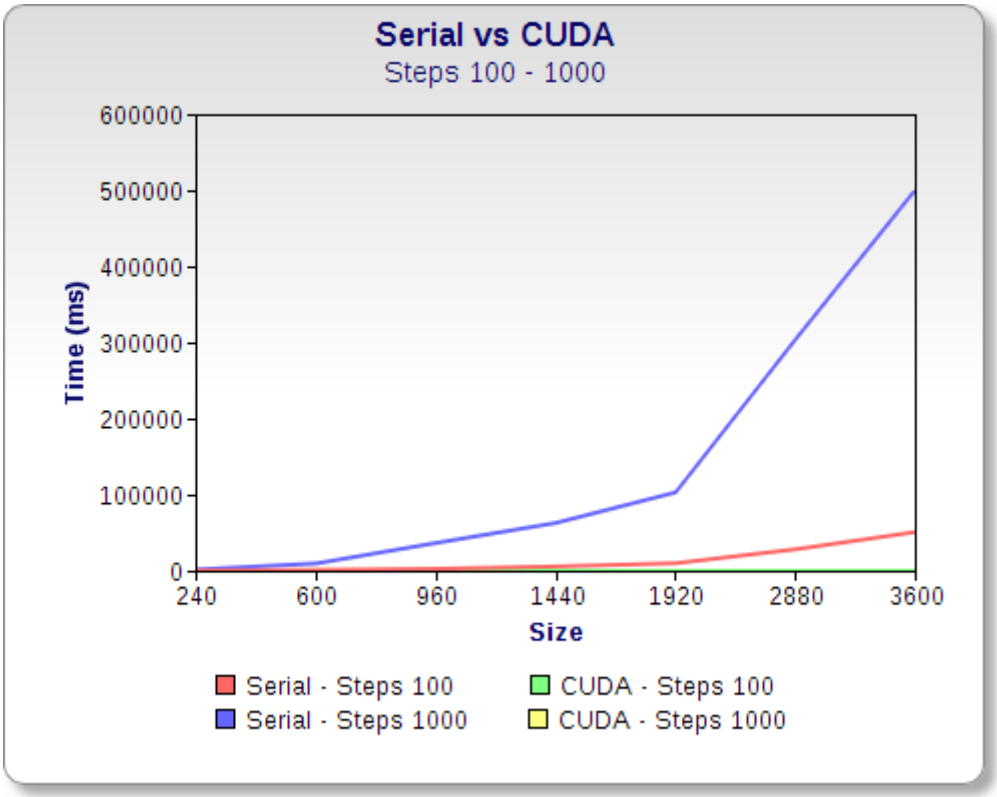

35
30
25
20
15
10
5
0
Time (ms)
28.509
12.018
6.158
3.063
0.751
100
500
1000
2000
5000
Steps

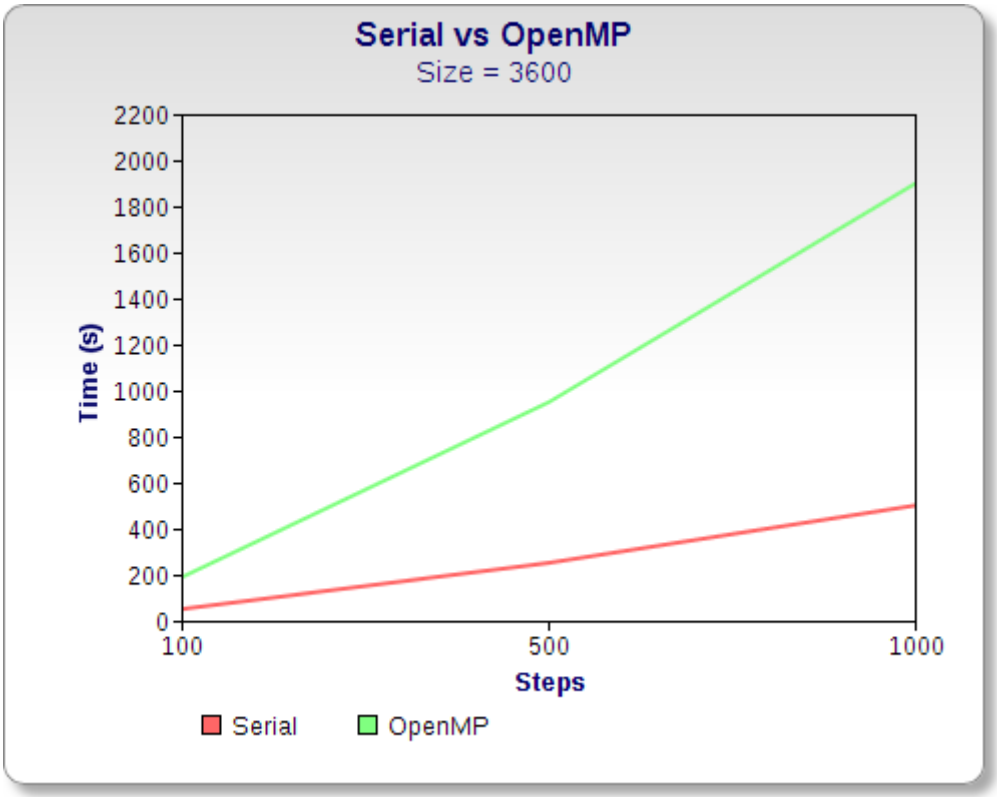
Περισσότερες Συγκρίσεις:

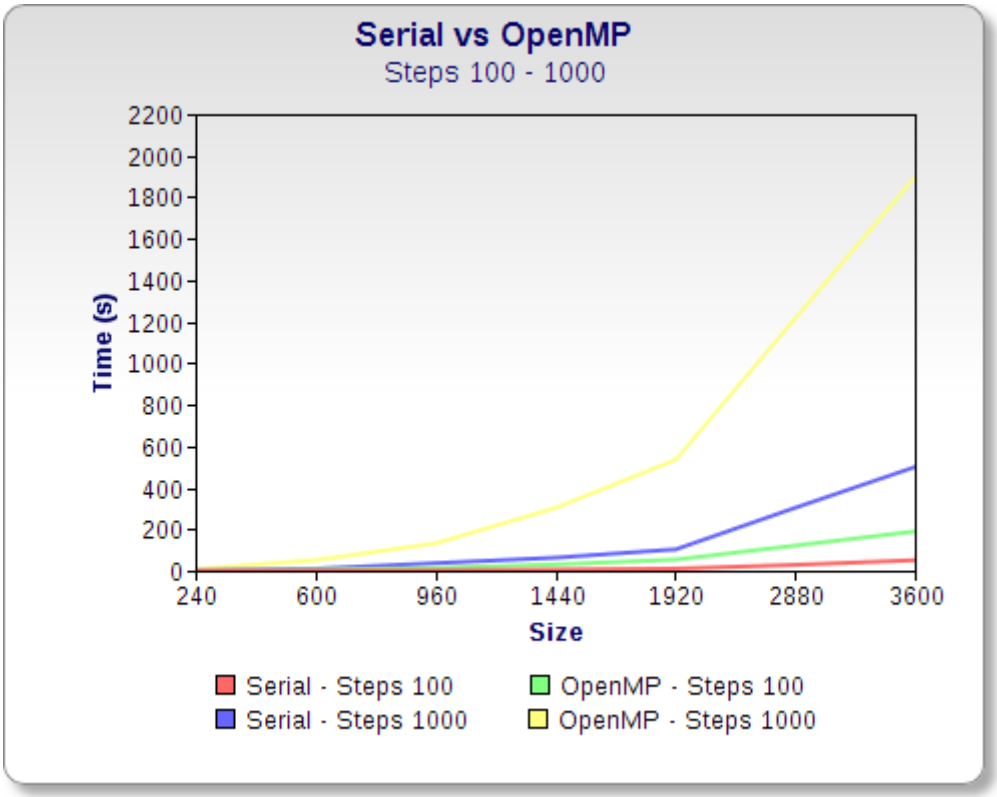





Serial vs CUDA
Size = 3600
600000
550000
500000
450000
400000
350000
300000
250000
200000
150000
100000
50000
0
Time (ms)
100
500
1000
Steps
Serial
CUDA


Serial vs CUDA
Steps 100 - 1000
600000
500000
400000
300000
200000
100000
0
Time (ms)
240
600
960
1440
1920
2880
3600
Size
Serial - Steps 100
Serial - Steps 1000
CUDA - Steps 100
CUDA - Steps 1000


Serial vs OpenMP
Size = 3600
Time (s)
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
100
500
1000
Steps
Serial
OpenMP

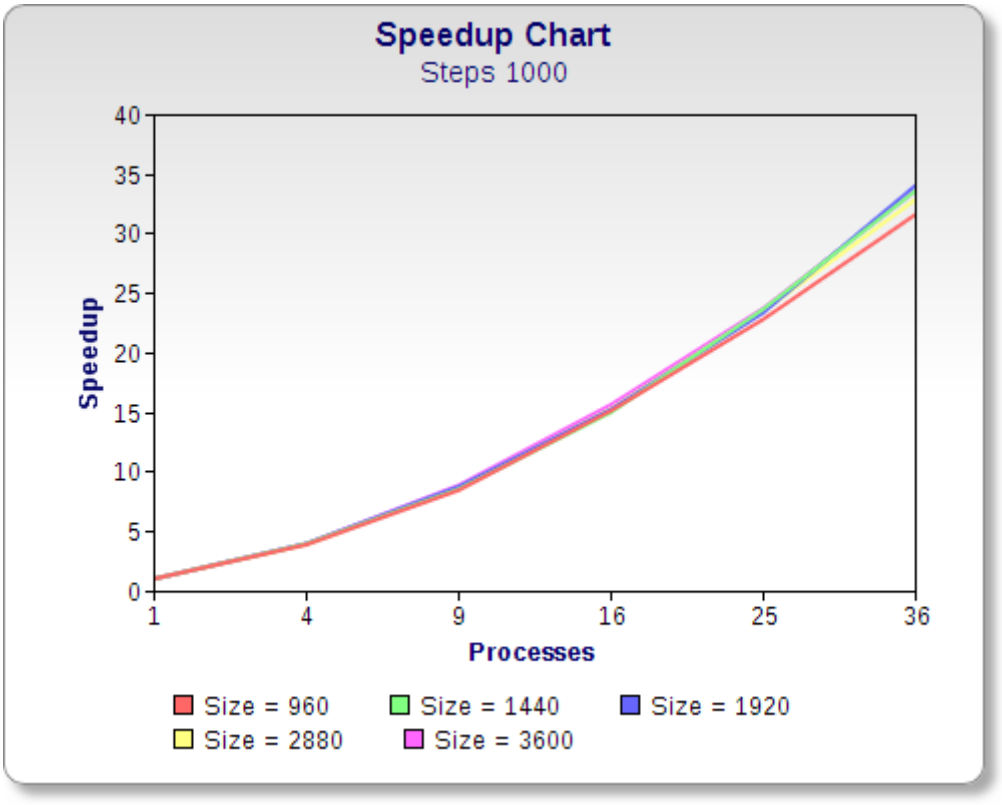

Serial vs OpenMP
Steps 100 - 1000
Time (s)
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
0
240
600
960
1440
1920
2880
3600
Size
Serial - Steps 100
OpenMP - Steps 100
Serial - Steps 1000
OpenMP - Steps 1000

Speedup – Efficiency

Παρακάτω παρουσιάζεται ο πίνακας που δείχνει την μεταβολή της επιτάχυνσης ανάλογα με το μέγεθος του προβλήματος και τον αριθμό των διεργασιών, καθώς επίσης και η σχετική γραφική παράσταση. Η επιτάχυνση (S) ορίζεται με βάση την σχέση S=Tserial / Tparal.

Speedup	**Size**	960	1440	1920	2880	3600
Processes	**Steps 1000**					
1		1	1	1	1	1
4		3.87	3.93	3.95	3.89	3.94
9		8.44	8.53	8.74	8.67	8.85
16		15.11	14.97	15.22	15.28	15.63
25		22.79	23.64	23.35	23.45	23.71
36		31.59	33.58	34.03	32.85	33.58

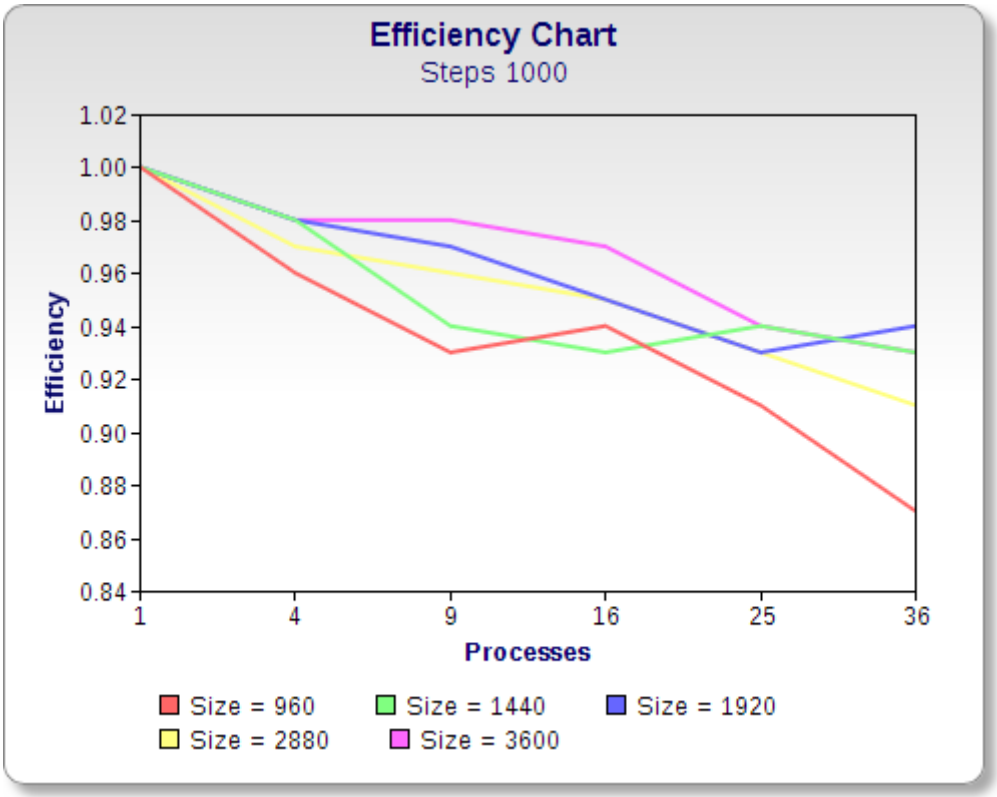
Όπως περιμέναμε, με αύξηση των processes, η επιτάχυνση αυξάνεται κι αυτή. Αυτό δικαιολογείται αφού το πρόβλημα μοιράζεται αναλόγως στις διεργασίες, και έτσι έχουμε καλή παραλληλία στην εκτέλεση, συνεπώς μικρότερο χρόνο άρα και μεγάλη επιτάχυνση. Επίσης παρατηρούμε ότι η επιτάχυνση δεν επηρεάζεται από την μεταβολή του μεγέθους του προβλήματος, αφού τα αποτελέσματα είναι σχεδόν ίδια. Αυτό συμβαίνει για τον ίδιο λόγο, επειδή έχουμε καταφέρει να μοιράσουμε αποδοτικά το πρόβλημα ανάμεσα στις διεργασίες. Το σχετικό διάγραμμα είναι το εξής:



Παρακάτω παρουσιάζεται ο πίνακας που δείχνει την μεταβολή της απόδοσης ανάλογα με το μέγεθος του προβλήματος και τον αριθμό των διεργασιών, καθώς επίσης και η σχετική γραφική παράσταση. Η απόδοση (E) ορίζεται με βάση την σχέση E=Tspeedup / Nprocesses.

Efficiency	**Size**	960	1440	1920	2880	3600
Processes	**Steps 1000**					
1		1	1	1	1	1
4		0.96	0.98	0.98	0.97	0.98
9		0.93	0.94	0.97	0.96	0.98
16		0.94	0.93	0.95	0.95	0.97
25		0.91	0.94	0.93	0.93	0.94
36		0.87	0.93	0.94	0.91	0.93

Σύμφωνα με τον παραπάνω πίνακα, παρατηρούμε ότι όσο μεγαλώνει το μέγεθος του προβλήματος και ο αριθμός των διεργασιών, η απόδοση παραμένει σταθερή. Αυτό μας δείχνει ότι το πρόβλημά μας είναι ισχυρά κλιμακωτό, αφού τα αποτελέσματα δε διαφέρουν ιδιαίτερα σε κάθε μέτρηση. Παραθέτουμε το διάγραμμα για την αποδοτικότητα στη συνέχεια ως εξής:


Efficiency Chart
Steps 1000
Efficiency
1.02
1.00
0.98
0.96
0.94
0.92
0.90
0.88
0.86
0.84
1
4
9
16
25
36
Processes
Size = 960
Size = 1440
Size = 1920
Size = 2880
Size = 3600