ΕΘΝΙΚΟ ΚΑΙ ΚΑΠΟΔΙΣΤΡΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Παράλληλα Συστήματα

Εργασία MPI 2015-16: Προσομοίωση μεταφοράς θερμότητας

Σπυρίδων Δ. Δελβινιώτης
ΑΜ: 1115201000037

ΑΘΗΝΑ

ΙΟΥΝΙΟΣ 2016

Περιεχόμενα

1. ΕΙΣΑΓΩΓΗ

Στην παρούσα εργασία εξετάζεται η λύση ενός προβλήματος που επιτρέπει την παράλληλη επεξεργασία δεδομένων, συγκεκριμένα την προσομοίωση μεταφοράς θερμότητας, με την βοήθεια τριών διαφορετικών αρχιτεκτονικών. Σκοπός της εργασίας αποτελεί, εκτός από την υλοποίηση των τριών αρχιτεκτονικών (MPI, Hybrid MPI/OpenMP, CUDA), η μέτρηση και αξιολόγηση της απόδοσης των προγραμμάτων διαφορετικής αρχιτεκτονικής καθώς και η μελέτη κλιμάκωσης με αυξανόμενο μέγεθος δεδομένων, αριθμό διεργασιών και νημάτων.

Ο κώδικας που αναπτύχθηκε, αποτελείται από τρεις διαφορετικές υλοποιήσεις. Η πρώτη αναπτύχθηκε σε περιβάλλον MPI, στο οποίο δημιουργούνται <n> διεργασίες στις μηχανές του αρχείου machines, καθεμιά εκ των οποίων αναλαμβάνει τον υπολογισμό τμήματος του πίνακα. Η δεύτερη αναπτύχθηκε με συνδυασμό των αρχιτεκτονικών MPI και OpenMP, με τους υπολογισμούς του εκάστοτε τμήματος του πίνακα σε νήματα, για βελτίωση της απόδοσης. Η τρίτη και τελευταία υλοποίηση αφορά τον προγραμματισμό GPUs μέσω του περιβάλλοντος CUDA και πραγματοποιείται μέσω της δημιουργίας νημάτων. Περαιτέρω λεπτομέρειες για την υλοποίηση και τις σχεδιαστικές επιλογές θα δοθούν στη συνέχεια.

2. ΥΛΟΠΟΙΗΣΗ

Σε αυτό το κεφάλαιο παρουσιάζονται στοιχεία της υλοποίησης των αρχιτεκτονικών που αναλύθηκαν.

2.1 Υλοποίηση MPI

Στην υλοποίηση της αρχιτεκτονικής με MPI η παραλληλία επιτυγχάνεται μέσω της επικοινωνίας διεργασιών οι οποίες συνδέονται με καρτεσιανή τοπολογία. Πιο συγκεκριμένα, κάθε διεργασία αρχικοποιεί ξεχωριστά το μέρος των δεδομένων που πρόκειται να επεξεργαστεί, δημιουργώντας υπό πίνακες. Κατά συνέπεια καταργείται η λογική master – slaves, που υπήρχε στην αρχική υλοποίηση. Έπειτα, επικοινωνεί 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, η οποία είναι δαπανηρή στην αποστολή στηλών του υποπίνακα, χρησιμοποιούνται datatypes. Τέλος, για περαιτέρω βελτίωση η συνάρτηση update χωρίζεται σε δύο τμήματα, την update_inside_table και την update_outside_table. 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.

2.2 Υβριδική Υλοποίηση MPI – Open MP

Για βελτίωση της απόδοσης του προγράμματος μας, έχουμε την δυνατότητα μέσω του περιβάλλοντος OpenMP, να δημιουργούμε σε κάθε μηχανή που τρέχει μια διεργασία MPI, έναν αριθμό νημάτων/threads που αναλαμβάνουν την παράλληλη πραγματοποίηση των υπολογισμών του πίνακα.

2.3 Υλοποίηση Open MP

Στα πλαίσια διεύρυνσης των γνώσεων έγινε μία υλοποίηση OpenMp. Στην υλοποίηση αυτή χρησιμοποιούνται μόνο νήματα χωρίς χρήση MPI.

2.4 Υλοποίηση CUDA

Στην υλοποίηση με CUDA η παραλληλία επιτυγχάνεται με τη χρήση της κάρτας γραφικών Nvidia. Κάθε thread που δημιουργείται αναλαμβάνει το υπολογισμό ενός πλέγματος του πίνακα. Αφού πραγματοποιηθούν οι απαραίτητες δεσμεύσεις και μεταφορές από την μνήμη host στην device, το πρόβλημα χωρίζεται σε υπό προβλήματα και δημιουργούνται N blocks με N threads σε κάθε block. Έπειτα από συγκεκριμένο αριθμό βημάτων, αφού πρώτα ανακτηθούν οι πίνακες στο host, πραγματοποιούνται οι απαραίτητες αποδεσμεύσεις μνήμης και το πρόγραμμα τερματίζει.

Πιο συγκεκριμένα, αρχικά αναπτύχθηκε ένα σειριακό πρόγραμμα χωρίς χρήση MPI/OpenMP/CUDA. Στη συνέχεια, προστέθηκε όλη λειτουργικότητα που απαιτεί η CUDA, όπως δέσμευση/αποδέσμευση μνήμης στη GPU, αντιγραφή πινάκων, συγχρονισμός νημάτων κ.α.

3. ΜΕΤΡΗΣΕΙΣ, ΕΠΙΔΟΣΕΙΣ ΚΑΙ ΚΛΙΜΑΚΩΣΗ

Στο κεφάλαιο αυτό θα γίνει παρουσίαση των μετρήσεων, των επιδόσεων και της κλιμάκωσης. Πρέπει να τονιστεί ότι οι μετρήσεις έγιναν στο εργαστήριο Linux της σχολής και ο αριθμός των διαθέσιμων κόμβων του Cluster ήταν 14 και κάθε ένα από αυτά υποστηρίζει 2 διεργασίες.

3.1 Αρχικό MPI

Το πλήθος των διεργασιών, για τις οποίες έγιναν οι μετρήσεις του αρχικού MPI, ήταν 4 και 9. Υπήρχε περιορισμός στον ελάχιστο και μέγιστο αριθμό διεργασιών.

Παρακάτω παρουσιάζονται αναλυτικοί πίνακες των αποτελεσμάτων για 100, 500 και 1000 βήματα.

STEPS: 100			
	Size (NxN)		
Processes (n)	240	600	960
4	1,28	3,70	8,53
9	1,15	3,36	7,37

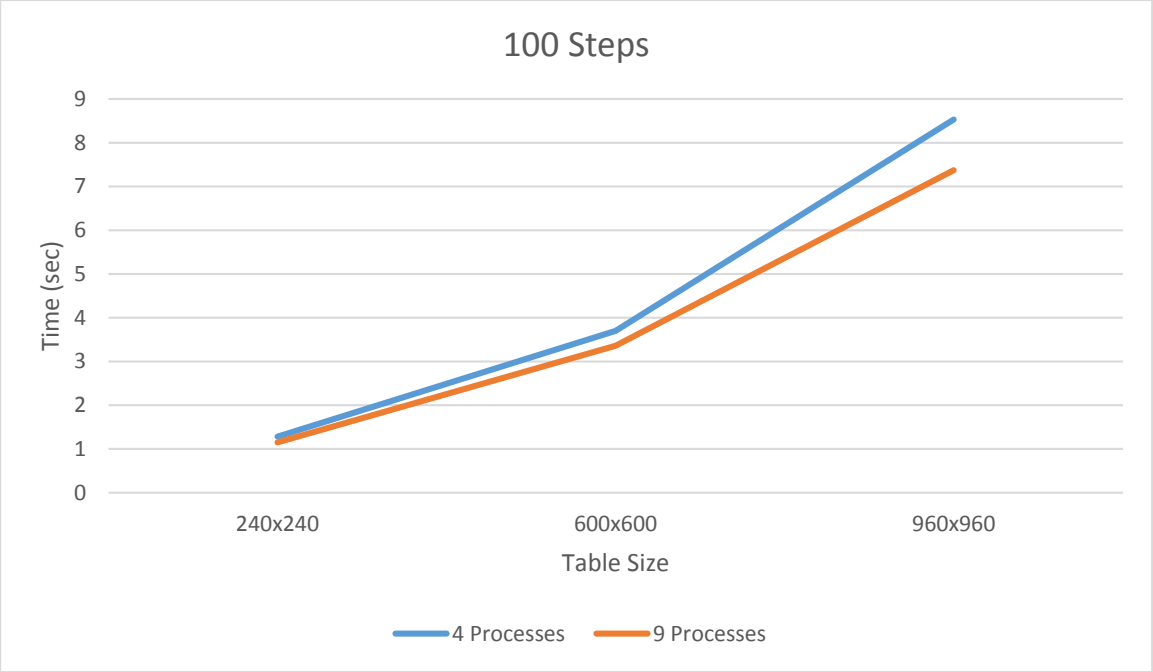
STEPS: 500			
	Size (NxN)		
Processes (n)	240	600	960
4	1,98	7,21	15,89
9	1,62	5,14	10,79

STEPS: 1000			
	Size (NxN)		
Processes (n)	240	600	960
4	3,48	11,58	25,74
9	2,39	7,80	15,53

Το αρχικό πρόγραμμα MPI έχει εμφανώς χειρότερους χρόνους σε σχέση με το πρόγραμμα MPI που αναπτύχθηκε στα πλαίσια της άσκησης. Τα παραπάνω αποτελέσματα οφείλονται στην έλλειψη πλήρους επικοινωνίας διεργασιών και τοπολογίας. Επιπλέον η ύπαρξη της έννοιας του master process, όπου μία διεργασία αναλαμβάνει την αρχικοποίηση και το διαμοιρασμό των δεδομένων, όπως και τη συγκέντρωση των αποτελεσμάτων, αποτελεί κύριο λόγο για την

μειωμένη επίδοση του αρχικού προγράμματος. Όσο αυξάνονται οι διεργασίες ο χρόνος βελτιώνεται, ενώ όσο αυξάνεται το μέγεθος του προβλήματος, ο χρόνος εκτέλεσης αυξάνεται.

Παρακάτω παρατίθεται διάγραμμα απόδοσης για 100 βήματα, τεσσάρων και εννέα διεργασιών, σε συνάρτηση με τον χρόνο (σε seconds) και το μέγεθος του πίνακα.



3.2 Βελτιωμένο MPI

Το πλήθος των διεργασιών, για τις οποίες έγιναν οι μετρήσεις του βελτιωμένου MPI, ήταν 1, 4, 16 και 25.

Παρακάτω παρουσιάζονται αναλυτικοί πίνακες των αποτελεσμάτων για 100, 500 και 1000 βήματα.

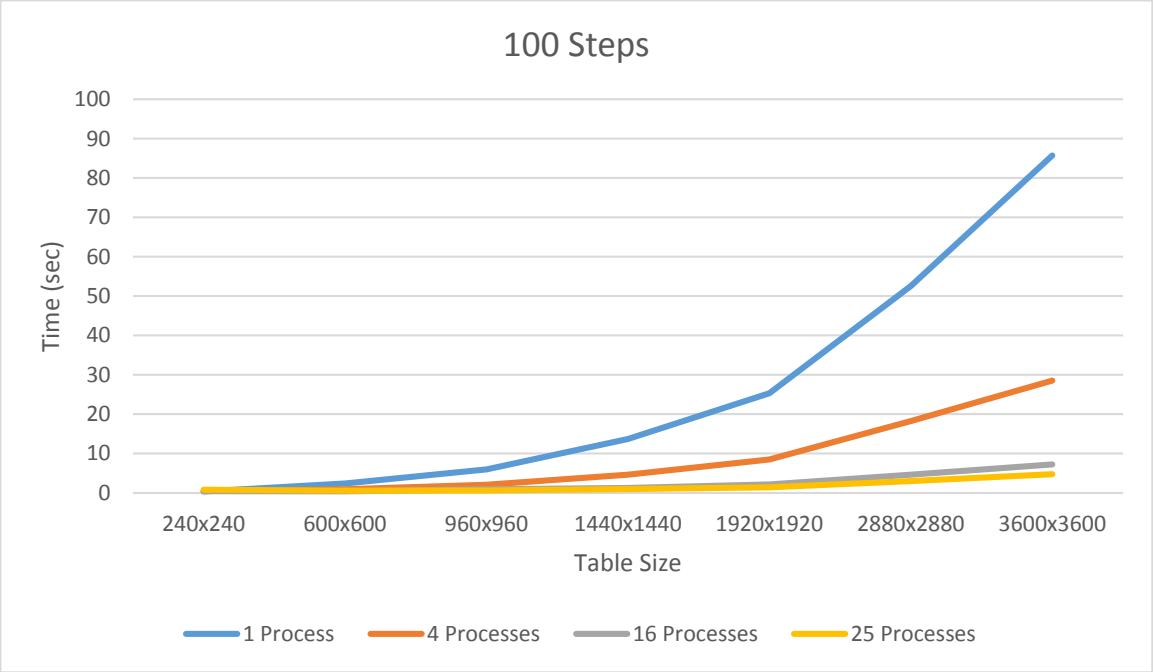
STEPS: 100							
	Size (NxN)						
Processes (n)	**240**	**600**	**960**	**1440**	**1920**	**2880**	**3600**
1	0.38	2.40	5.97	13.68	25.33	52.54	85.72
4	0.49	0.95	2.06	4.63	8.50	18.25	28.54
16	0.43	0.40	0.95	1.25	2.14	4.65	7.25
25	0.83	0.55	0.61	0.97	1.44	3.02	4.76

STEPS: 500							
	Size (NxN)						
Processes (n)	240	600	960	1440	1920	2880	3600
1	1.88	11.98	30.10	67.63	119.16	269.31	425.32
4	1.78	4.23	10.42	23.22	41.20	91.29	143.07
16	1.68	1.98	2.90	6.12	10.74	23.04	36.08
25	1.49	1.85	2.30	3.92	6.86	15.95	42.35

STEPS: 1000							
	Size (NxN)						
Processes (n)	240	600	960	1440	1920	2880	3600
1	3.82	22.98	59.87	135.42	241.49	540.06	847.12
4	2.88	8.66	21.76	46.44	83.04	182.97	286.11
16	2.54	3.52	5.81	11.93	21.10	46.12	71.82
25	3.53	3.80	4.60	7.91	13.66	30.01	46.51

Παρατηρούμε ότι για κάθε σταθερή τιμή διάστασης αυξάνοντας τον αριθμό των διεργασιών έχουμε μείωση στο χρόνο εκτέλεσης. Αυτό είναι δικαιολογημένο, διότι αυξάνεται η παραλληλία, την οποία εκμεταλλευόμαστε μέσω της επικοινωνίας και της τοπολογίας διεργασιών. Αν κρατηθεί ο αριθμός των διεργασιών σταθερός και αυξηθεί η διάσταση του πλέγματος, ο χρόνος γίνεται πολύ μεγαλύτερος, αφού τα δεδομένα προς επεξεργασία αυξάνονται.

Παρακάτω παρατίθεται διάγραμμα απόδοσης για 100 βήματα, μίας, τεσσάρων, δεκαέξι και εικοσιπέντε διεργασιών, σε συνάρτηση με τον χρόνο (σε seconds) και το μέγεθος του πίνακα.



3.3 Hybrid MPI/Open MP

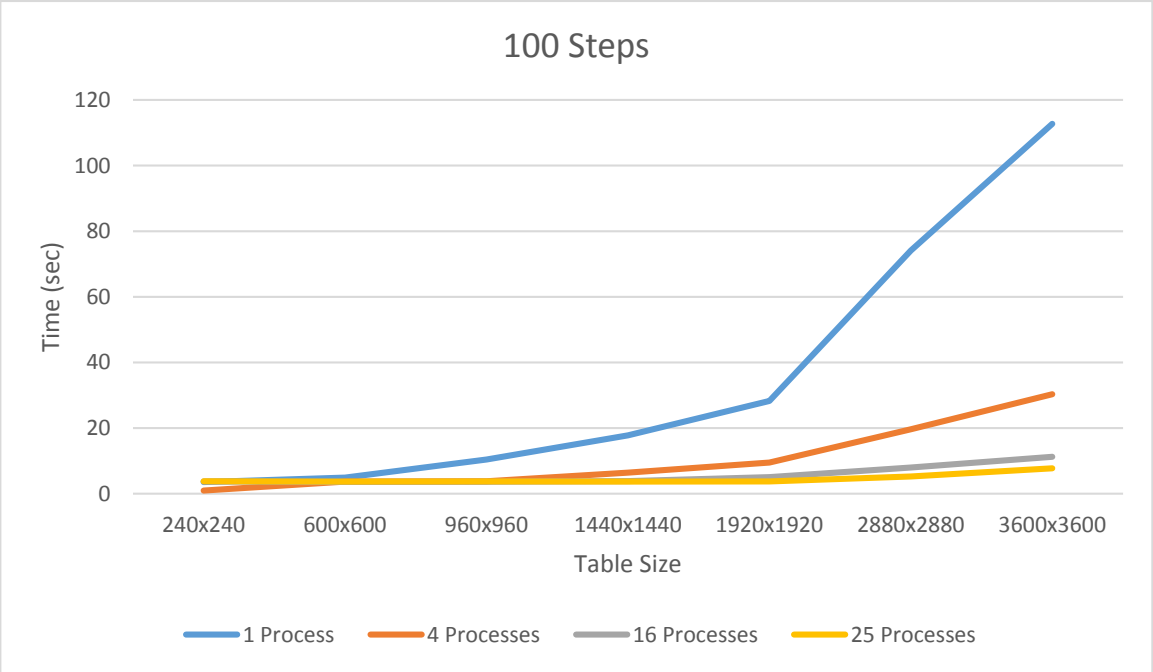
Το πλήθος των διεργασιών, για τις οποίες έγιναν οι μετρήσεις της Υβριδικής υλοποίησης του MPI/OpenMP, ήταν 1, 4, 16 και 25. Το OpenMP ήταν υπεύθυνο για τον αριθμό των νημάτων που θα δημιουργούσε.

Παρακάτω παρουσιάζεται αναλυτικός πίνακας των αποτελεσμάτων για 100 βήματα.

STEPS: 100							
	Size (NxN)						
Processes (n)	240	600	960	1440	1920	2880	3600
1	3.52	4.89	10.42	17.78	28.27	74.11	112.74
4	0.93	3.67	3.75	6.41	9.47	19.61	30.32
16	3.71	3.59	3.61	3.83	5.03	7.96	11.22
25	3.78	3.69	3.73	3.66	3.73	5.26	7.74

Παρατηρούμε ότι για κάθε σταθερή τιμή διάστασης αυξάνοντας τον αριθμό των διεργασιών έχουμε μείωση στο χρόνο εκτέλεσης. Αυτό είναι δικαιολογημένο, διότι αυξάνεται η παραλληλία, την οποία εκμεταλλευόμαστε μέσω της επικοινωνίας και της τοπολογίας διεργασιών και νημάτων.

Παρακάτω παρατίθεται διάγραμμα απόδοσης για 100 βήματα, μίας, τεσσάρων, δεκαέξι και εικοσιπέντε διεργασιών, σε συνάρτηση με τον χρόνο (σε seconds) και το μέγεθος του πίνακα.

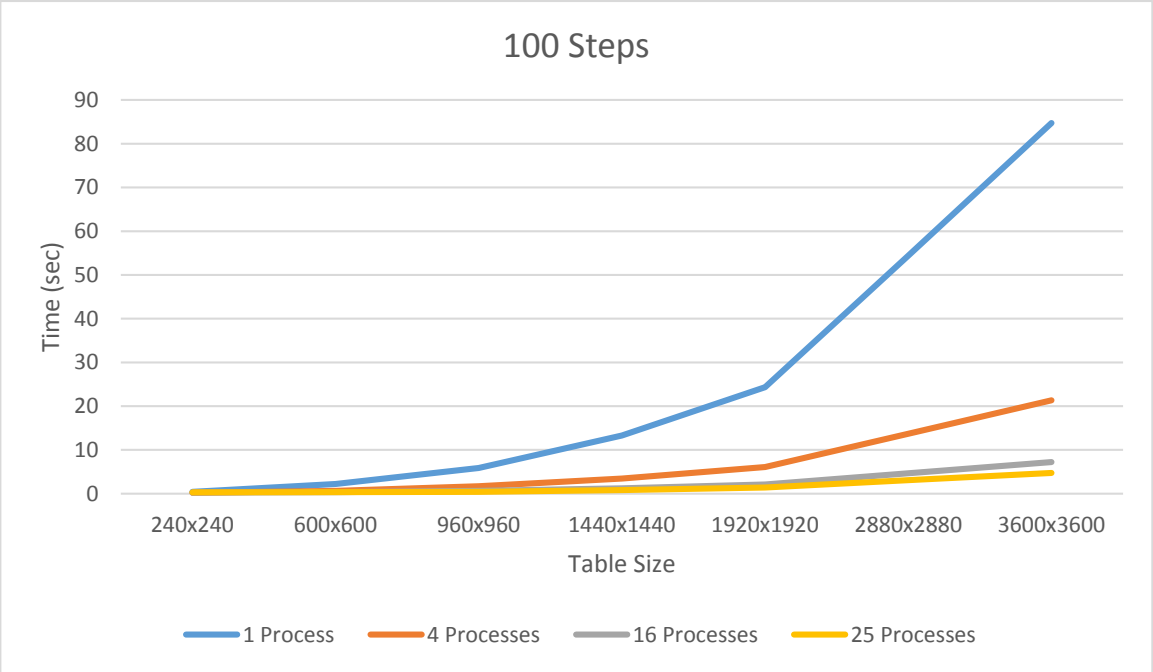


Στη συνέχεια θα μελετηθεί η σειριακή εκτέλεση της Υβριδικής υλοποίησης MPI/OpenMP. Ουσιαστικά έχει προκύψει από την μεταγλώττιση της Υβριδικής υλοποίησης MPI/OpenMP χωρίς το Flag "–fopenmp".

Παρακάτω παρουσιάζεται αναλυτικός πίνακας των αποτελεσμάτων για 100 βήματα.

STEPS: 100							
	Size (NxN)						
Processes (n)	240	600	960	1440	1920	2880	3600
1	0.42	2.21	5.86	13.30	24.33	54.39	84.71
4	0.18	0.65	1.71	3.46	6.08	13.70	21.33
16	0.24	0.36	0.64	1.23	2.10	4.65	7.22
25	0.29	0.34	0.42	0.83	1.41	3.07	4.72

Παρακάτω παρατίθεται διάγραμμα απόδοσης για 100 βήματα, μίας, τεσσάρων, δεκαέξι και εικοσιπέντε διεργασιών, σε συνάρτηση με τον χρόνο (σε seconds) και το μέγεθος του πίνακα.



3.4 Open MP

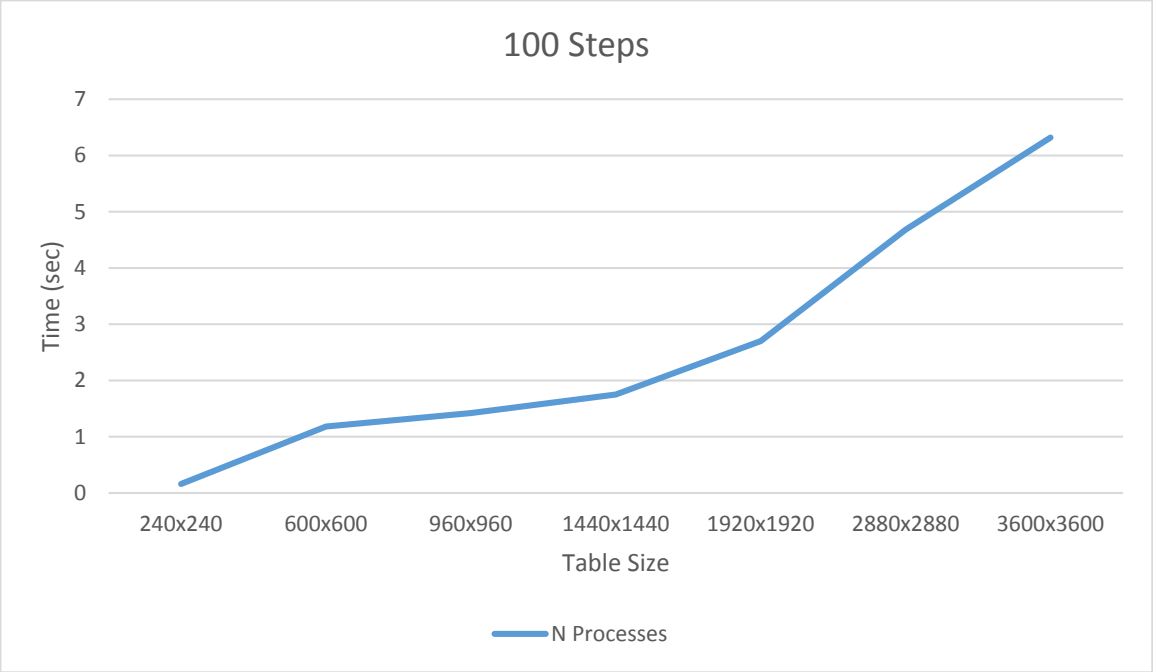
Στα πλαίσια εμβάθυνσης των γνώσεων έγινε μία υλοποίηση του OpenMP χωρίς MPI. Στην περίπτωση του OpenMp δεν υπάρχουν πολλές διεργασίες, αλλά πολλαπλά νήματα.

Το OpenMP ήταν υπεύθυνο για τον αριθμό των νημάτων που θα δημιουργούσε.

Παρακάτω παρουσιάζεται αναλυτικός πίνακας των αποτελεσμάτων για 100 βήματα.

STEPS: 100							
	Size (NxN)						
Processes (n)	**240**	**600**	**960**	**1440**	**1920**	**2880**	**3600**
n	0.16	1.18	1.42	1.75	2.70	4.68	6.32

Παρατηρείται ότι ο χρόνος είναι εμφανώς μικρότερος από το αρχικό δοθέν πρόγραμμα, ωστόσο υστερεί σε σχέση με την βελτιωμένη MPI έκδοση που αναπτύχθηκε.



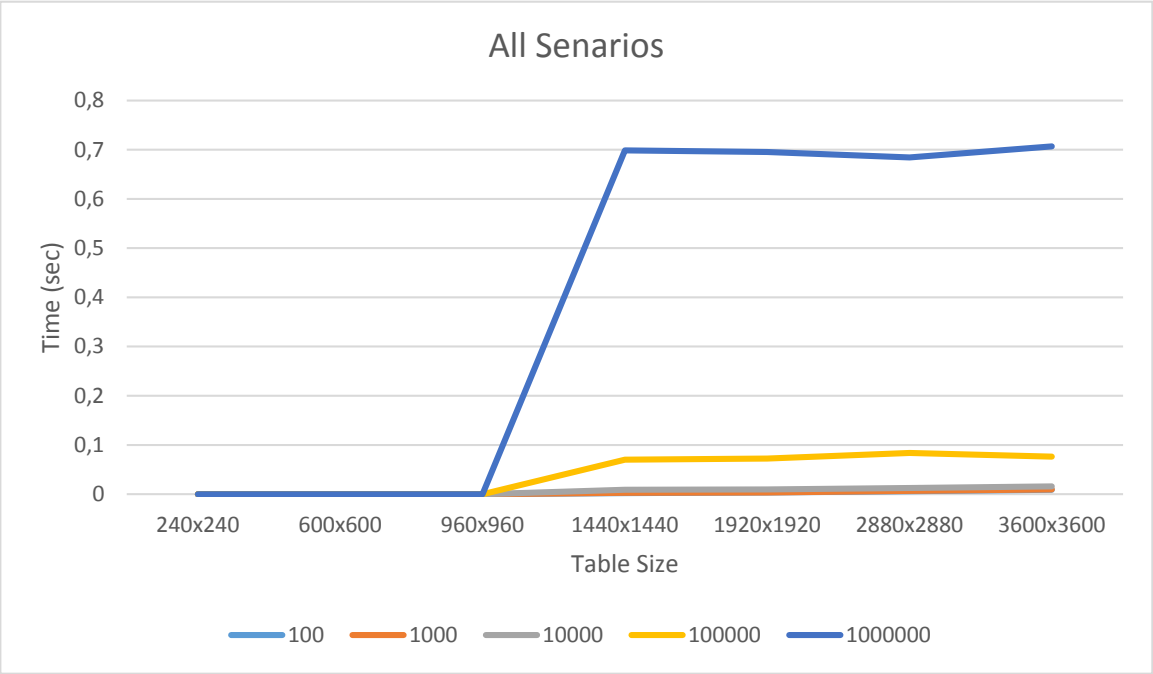
3.5 CUDA

Στην περίπτωση της υλοποίησης CUDA, τα αποτελέσματα είναι πραγματικά εκπληκτικά. Προσωπικά δεν περίμενα τέτοια βελτίωση.

Παρακάτω ακολουθεί πίνακας των αποτελεσμάτων.

STEPS	Size (NxN)						
	240	600	960	1440	1920	2880	3600
100				0.0021	0.0034	0.0060	0.0091
1000				0.0027	0.0036	0.0067	0.0096
10000				0.0091	0.0097	0.0127	0.0157
100000				0.0702	0.0725	0.0838	0.0762
1000000				0.6985	0.6955	0.6844	0.7066

Παρατηρείται ότι οι χρόνοι είναι πολύ χαμηλοί συγκριτικά με τις υπόλοιπες υλοποιήσεις. Πρέπει να γίνει αντιληπτό ότι η CUDA έχει την δυνατότητα να τρέξει σε πάρα πολλά νήματα, συνεπώς το πρόβλημα μπορεί να παραλληλοποιηθεί αποτελεσματικά.

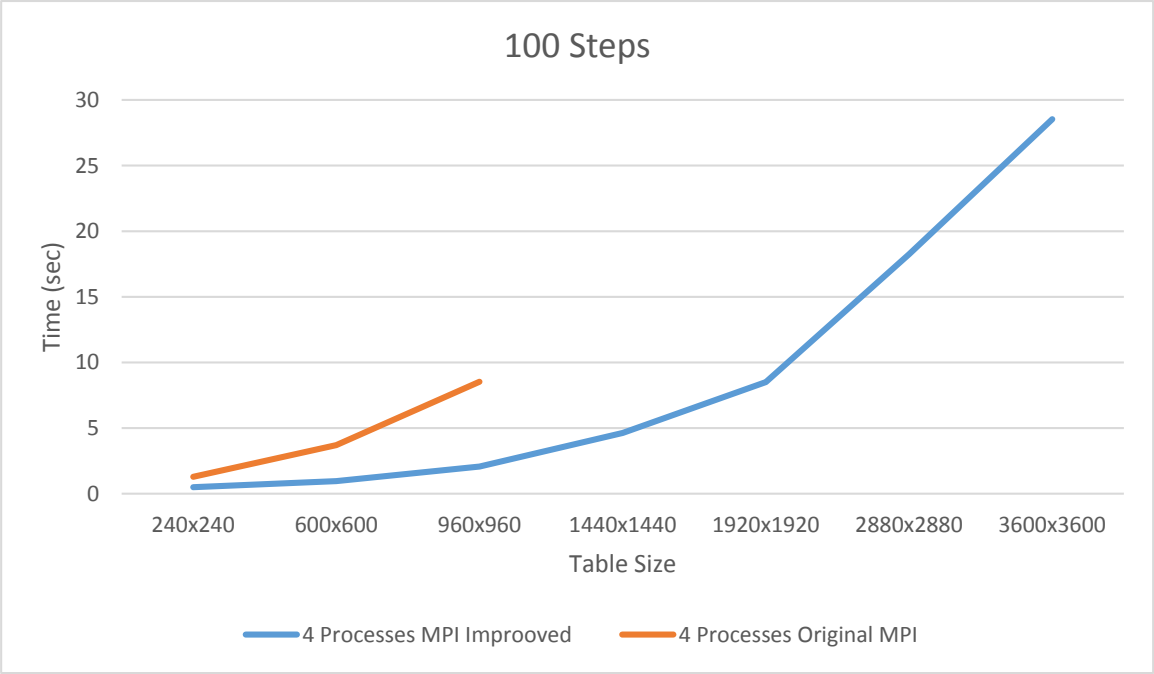


3.6 Συγκρίσεις μετρήσεων

Στο κεφάλαιο αυτό θα γίνει παρουσίαση των συγκρίσεων που έγιναν ανάμεσα στις παραπάνω υλοποιήσεις.

3.6.1 Αρχικό MPI – Βελτιωμένο MPI

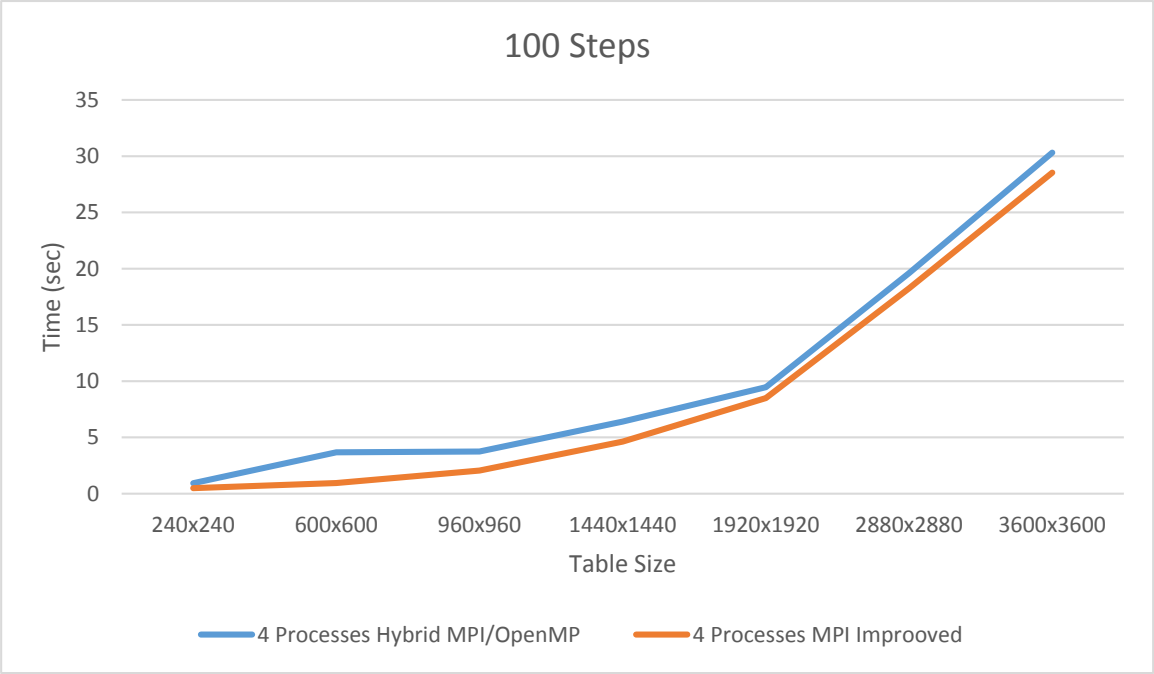
Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στο αρχικό MPI και στο βελτιωμένο MPI.



Παρατηρείται μεγάλη βελτίωση ανάμεσα στα δύο προγράμματα. Αυτό είναι απόλυτα λογικό καθώς η βελτίωση αυτή οφείλεται στο διαμοιρασμό των δεδομένων σε Block και όχι σειρές, στην ασύγχρονη επικοινωνία και στην καρτεσιανή τοπολογία των διεργασιών. Επιπλέον, σημαντικό ρόλο έχει η χρήση datatypes, διότι έτσι αποφεύχθηκαν πολλαπλές αντιγραφές.

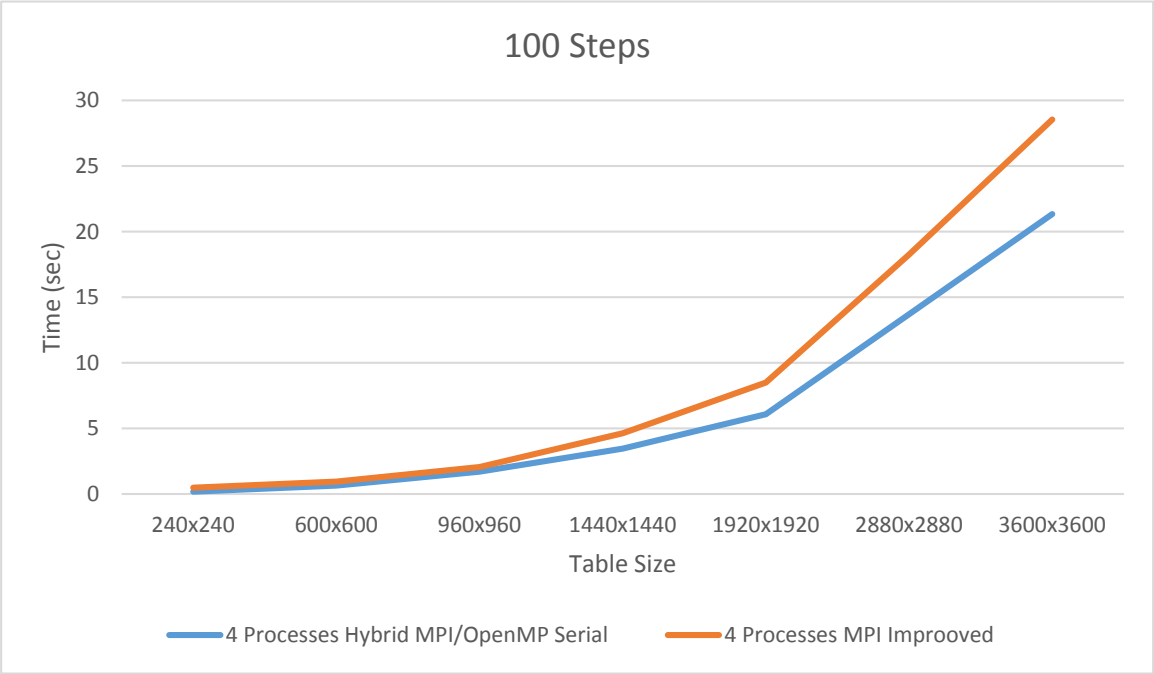
3.6.2 Βελτιωμένο MPI – Hybrid MPI/Open MP

Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στο βελτιωμένο MPI και στην Υβριδική υλοποίηση MPI/OpenMP.



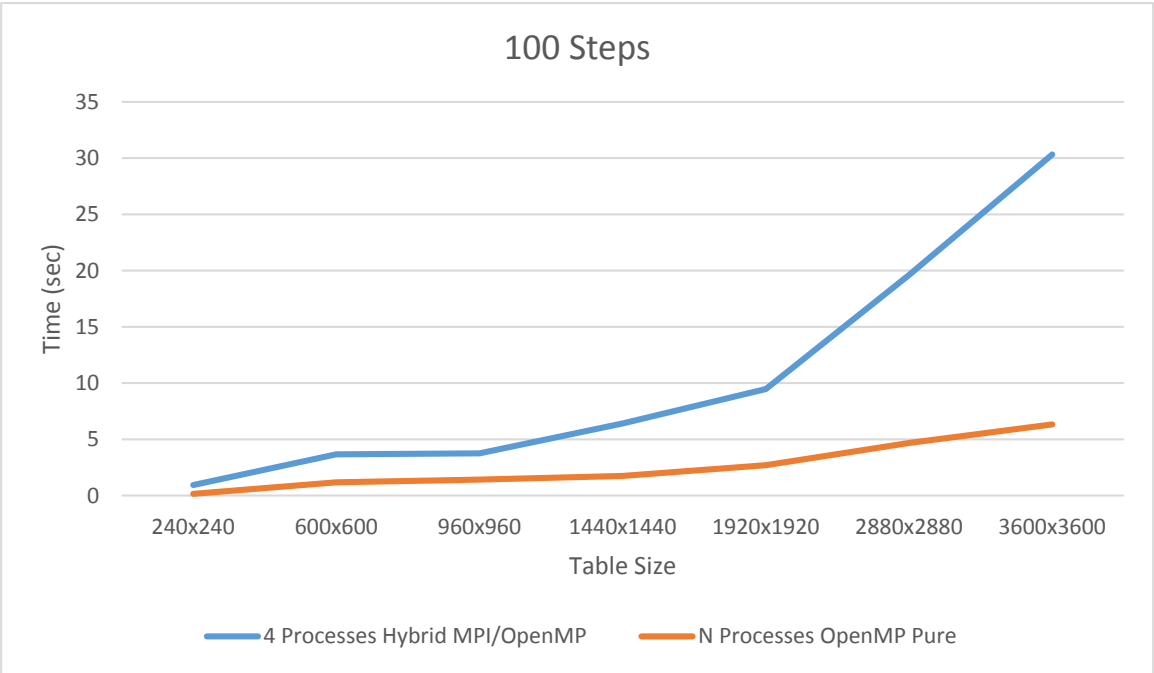
Παρατηρείται σχεδόν παρόμοια συμπεριφορά ανάμεσα στα δύο προγράμματα. Η υβριδική υλοποίηση είναι σαφώς χειρότερη. Αυτό είναι δικαιολογημένο, λαμβάνοντας υπόψιν το κόστος συγχρονισμού των νημάτων, που τελικά έχει αρνητικό αντίκτυπο στον τελικό χρόνο του προγράμματος. Υπάρχουν διάφορες τεχνικές που μπορούν να εφαρμοστούν προκειμένου η προσθήκη του OpenMP να έχει θετική επίδραση στην απόδοση.

Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στο βελτιωμένο MPI και στην σειριακά Υβριδική υλοποίηση MPI/OpenMP.



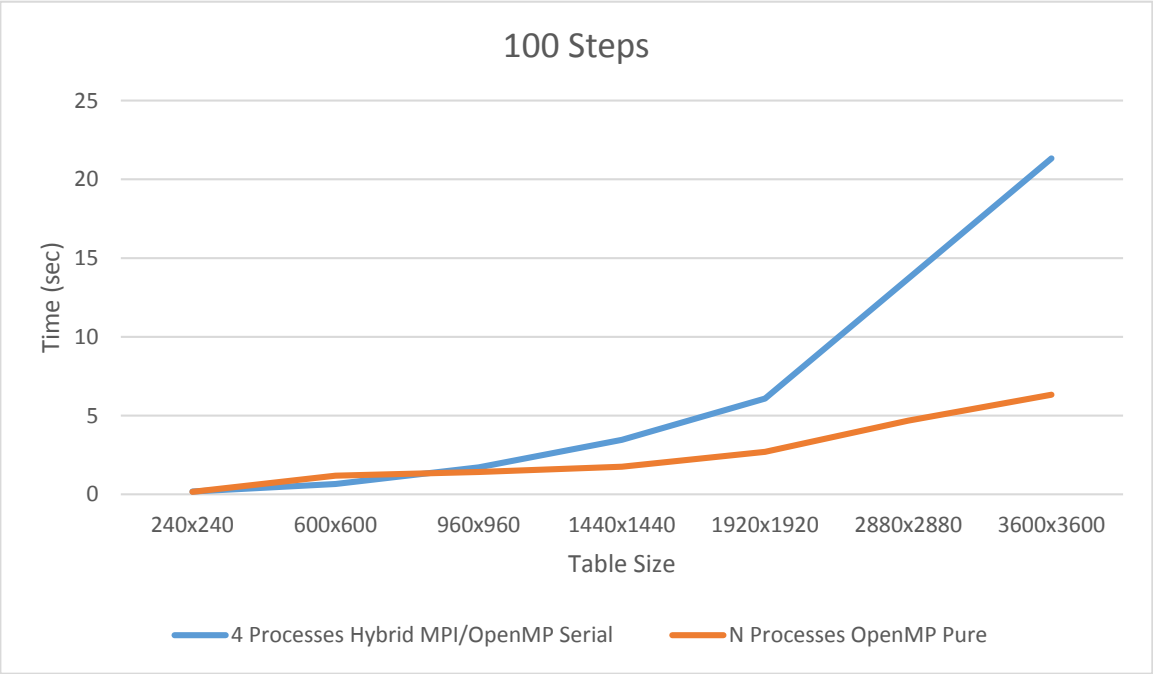
3.6.3 Open MP - Hybrid MPI/Open MP

Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στην Υβριδική υλοποίηση MPI/OpenMP και την υλοποίηση OpenMp.



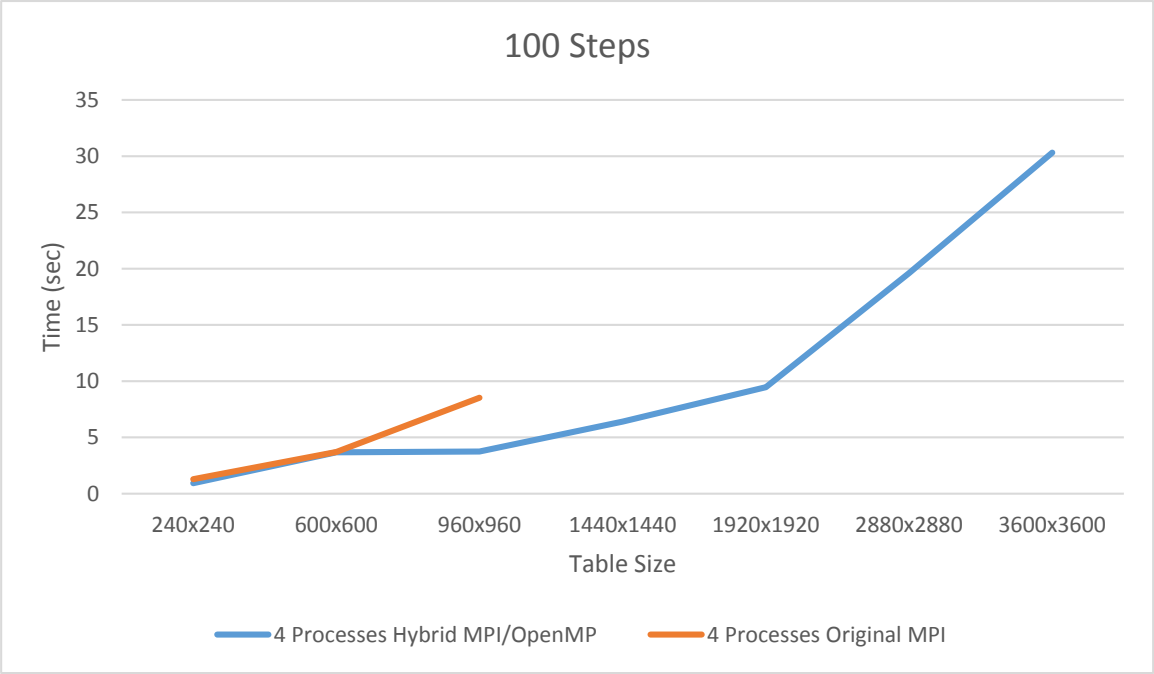
Παρατηρείται πολύ μεγάλη βελτίωση ανάμεσα στις δύο υλοποιήσεις.

Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στην σειριακά Υβριδική υλοποίηση MPI/OpenMP και την υλοποίηση OpenMp.

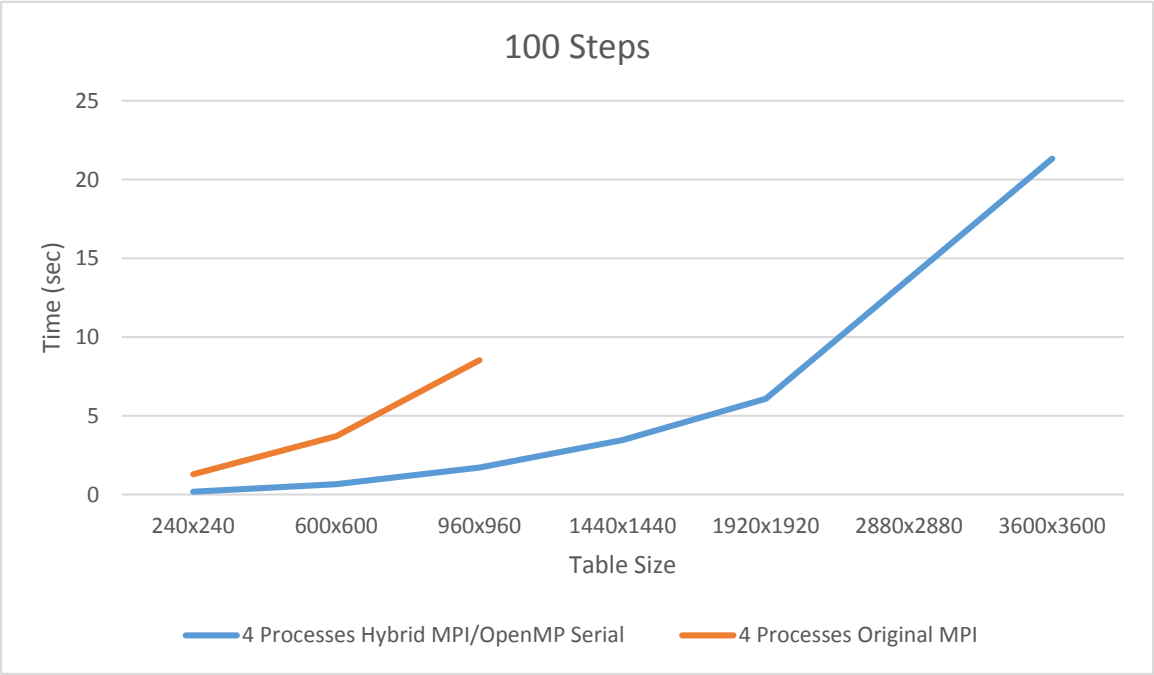


3.6.4 Αρχικό MPI - Hybrid MPI/Open MP

Παρακάτω παρουσιάζεται σύγκριση ανάμεσα στην αρχικό MPI και την Υβριδική υλοποίηση MPI/OpenMP.



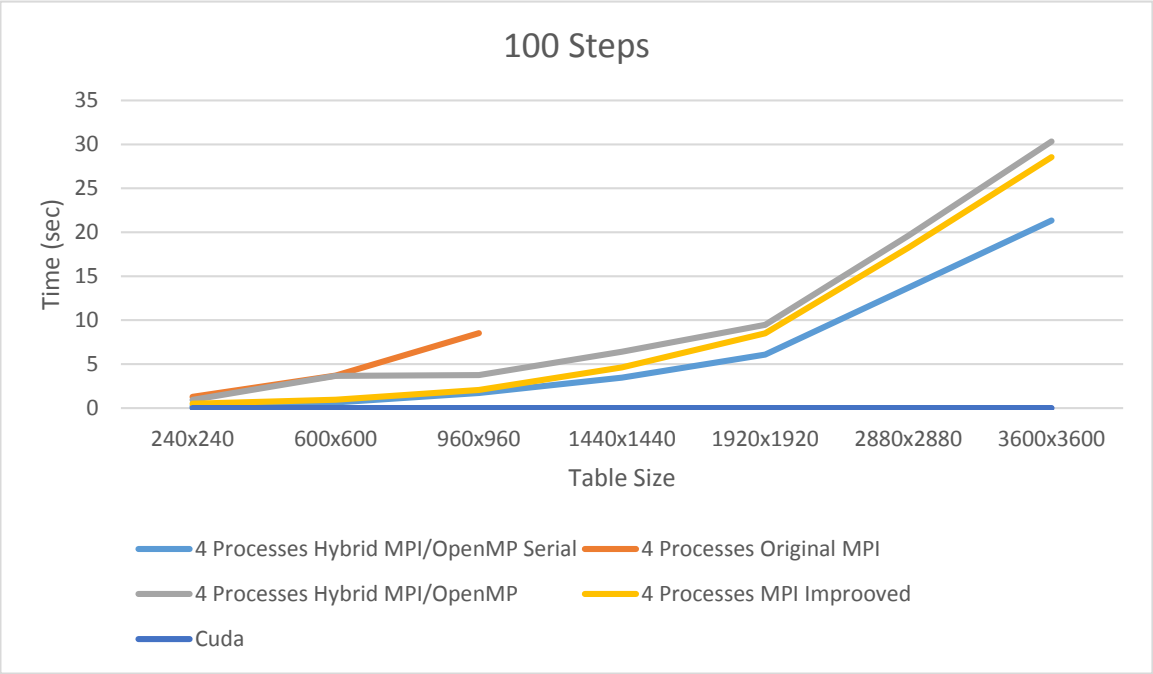
Παρακάτω αναφέρεται σύγκριση ανάμεσα στην αρχικό MPI και την σειριακή Υβριδική υλοποίηση MPI/OpenMP.



3.6.5 CUDA

3.6.6 Συγκεντρωτικές μετρήσεις

Παρακάτω παρουσιάζεται συγκεντρωτικό διάγραμμα για 100 βήματα και 4 διεργασίες.



3.7 Speedup – Efficiency

Σε αυτήν την ενότητα γίνεται η παρουσίαση της μεταβολής της επιτάχυνσης ανάλογα με το μέγεθος του προβλήματος και του πόρους (πλήθος διεργασιών) που χρησιμοποιήθηκαν. Πιο συγκεκριμένα η επιτάχυνση, ή αλλιώς *Speedup*, ορίζεται από τον παρακάτω τύπο.

$$Speedup = \frac{\text{Tserial}}{\text{Tparallel}}$$

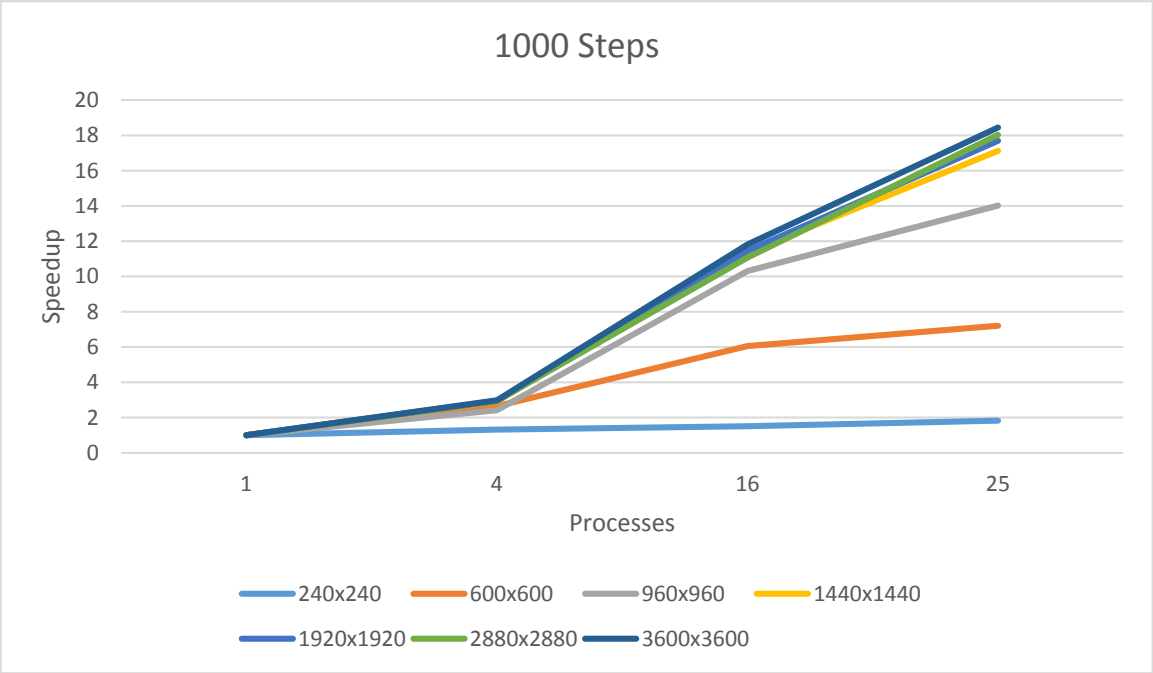
Όπου *Tserial* ο χρόνος εκτέλεσης σειριακά και *Tparallel* ο χρόνος εκτέλεσης παράλληλα.

Ακολουθεί αναλυτικός πίνακας αποτελεσμάτων για την Επιτάχυνση, ή αλλιώς *Speedup,* για 1000 βήματα.

STEPS: 1000							
	Size (NxN)						
Processes (n)	**240**	**600**	**960**	**1440**	**1920**	**2880**	**3600**
1	1	1	1	1	1	1	1
4	1.32	2.65	2.41	2.91	2.91	2.95	2.96
16	1.51	6.52	10.31	11.35	11.45	11.07	11.81
25	1.82	6.04	13.02	17.12	17.68	18.01	18.42

Παρατηρείται ότι με την αύξηση των διεργασιών (processes) η επιτάχυνση αυξάνεται περισσότερο. Αυτό δικαιολογείται αφού το πρόβλημα μοιράζεται αναλόγως στις διεργασίες, και έτσι έχουμε καλύτερη παραλληλία στην εκτέλεση και κατά συνέπεια μικρότερο χρόνο εκτέλεσης. Μια επιπλέον παρατήρηση είναι ότι η επιτάχυνση δεν επηρεάζεται από την μεταβολή του μεγέθους του προβλήματος. Αυτό συμβαίνει γιατί το πρόβλημα έχει κατανεμηθεί αποδοτικά ανάμεσα στις διεργασίες.

Παρακάτω ακολουθεί διαγραμματική αναπαράσταση του παραπάνω πίνακα.



Στη συνέχεια παρουσιάζεται η μεταβολή της απόδοσης ανάλογα με το μέγεθος του προβλήματος και το πλήθος των διεργασιών. Η μεταβολή της απόδοσης, ή αλλιώς Efficiency, ορίζεται από τον παρακάτω τύπο.

$$Efficiency = \frac{\text{Tspeedup}}{\text{N processes}}$$

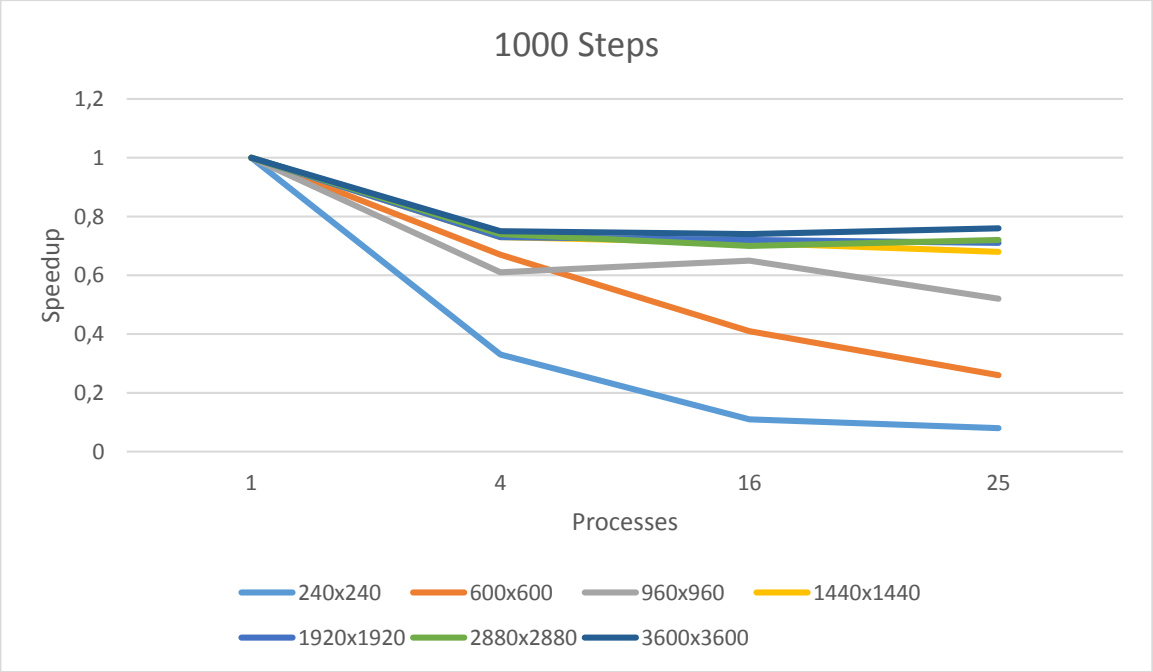
Όπου *Tspeedup* η Επιτάχυνση και *N processes* το πλήθος των διεργασιών που χρησιμοποιήθηκαν.

Παρακάτω ακολουθεί αναλυτικός πίνακας της μεταβολής της απόδοσης για 1000 βήματα.

STEPS: 1000							
	Size (NxN)						
Processes (n)	240	600	960	1440	1920	2880	3600
1	1	1	1	1	1	1	1
4	0.33	0.67	0.61	0.73	0.73	0.74	0.75
16	0.11	0.41	0.65	0.71	0.72	0.70	0.74
25	0.08	0.26	0.52	0.68	0.71	0.72	0.75

Παρατηρείται ότι όσο μεγαλώνει το πλήθος των διεργασιών και το μέγεθος του προβλήματος, η απόδοση μένει σε σταθερά επίπεδα. Αυτό οφείλεται στο ότι το πρόβλημά είναι ισχυρά κλιμακωτό, αφού τα αποτελέσματα δε διαφέρουν ιδιαίτερα σε κάθε μέτρηση.

Παρακάτω ακολουθεί διαγραμματική αναπαράσταση του παραπάνω πίνακα.

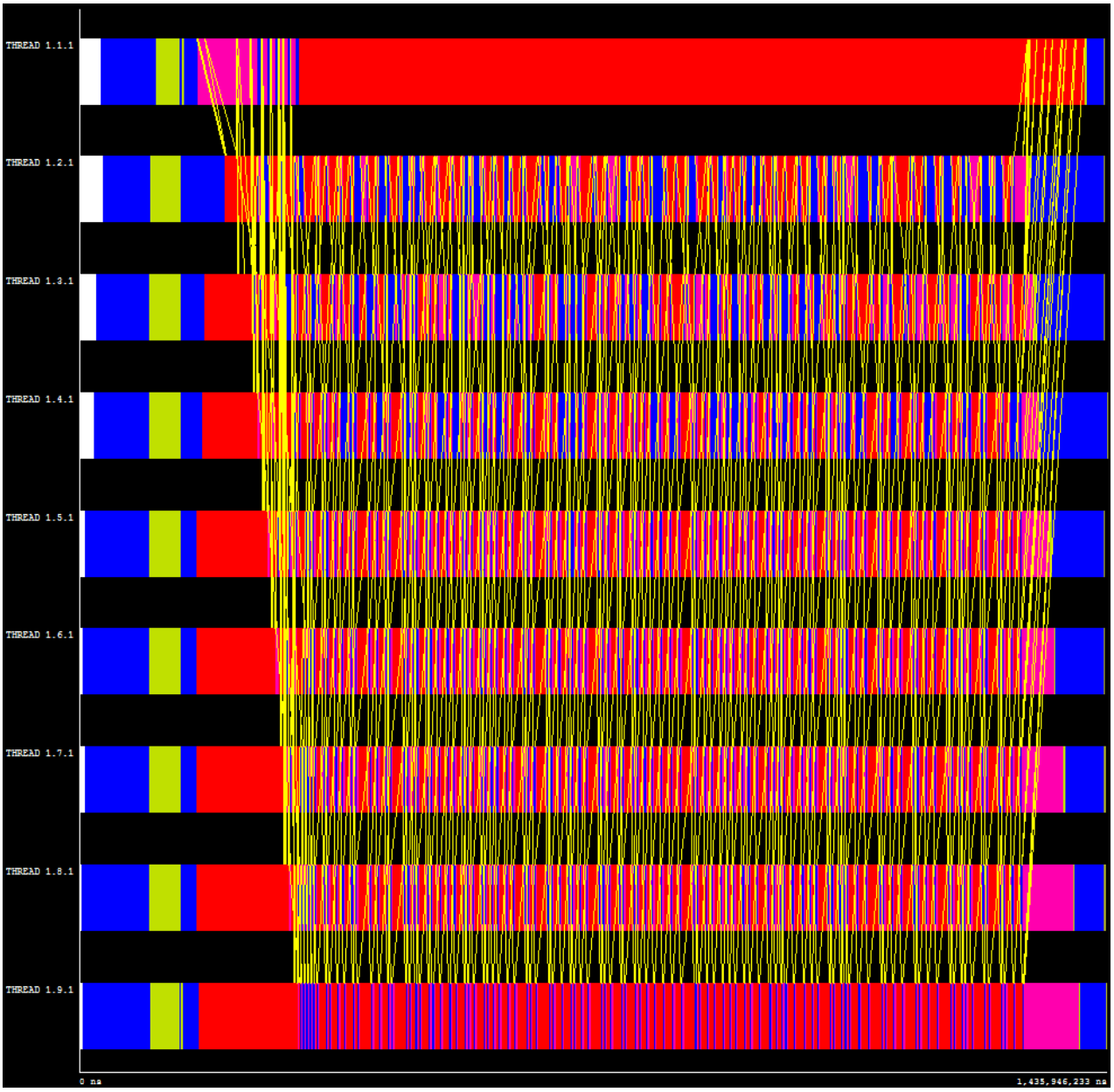


4 ΜΕΤΡΗΣΕΙΣ PARAVER

Σε αυτό το κεφάλαιο θα γίνει παρουσίαση των αποτελεσμάτων από το Paraver.

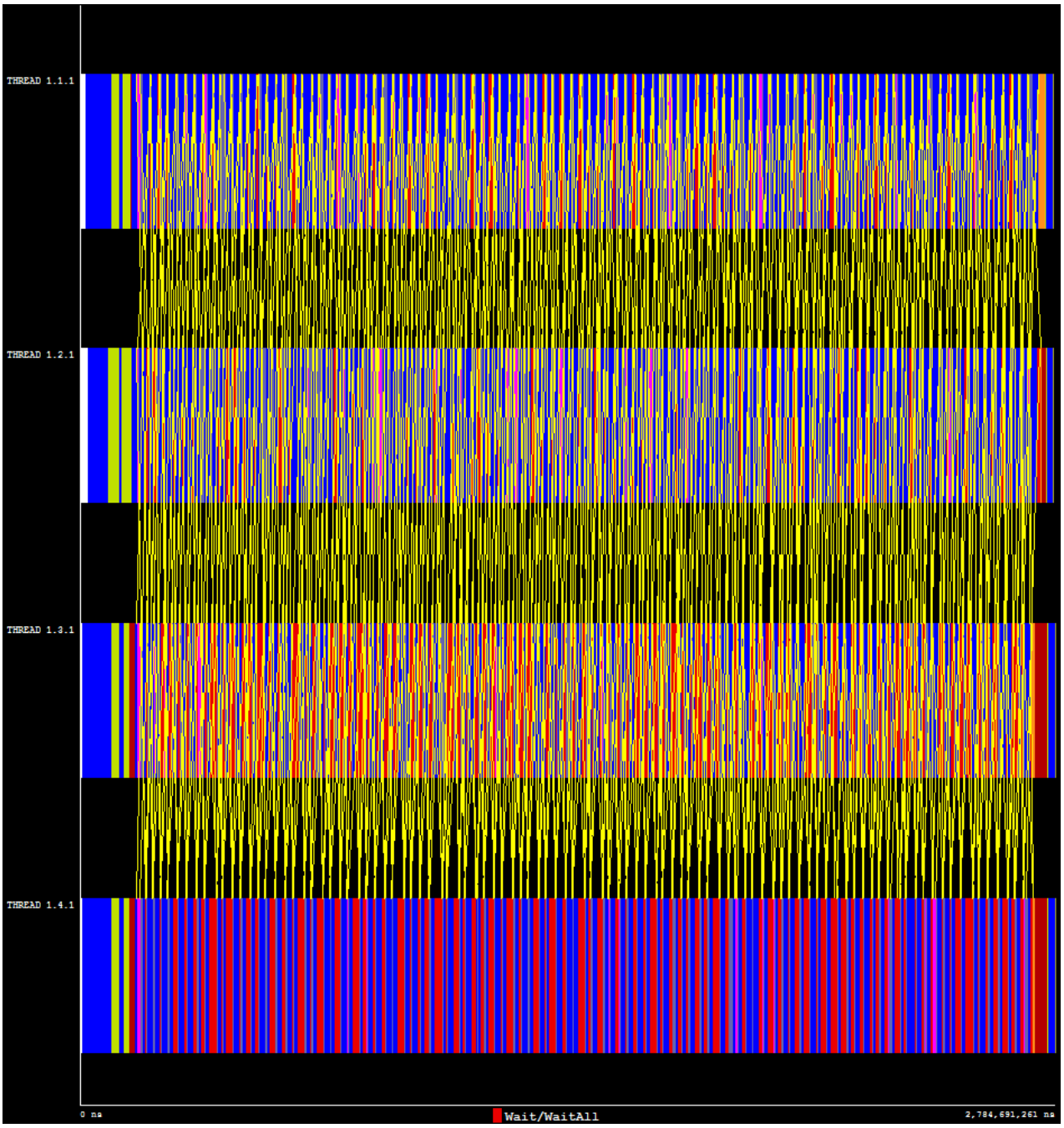
4.1 Αρχικό MPI

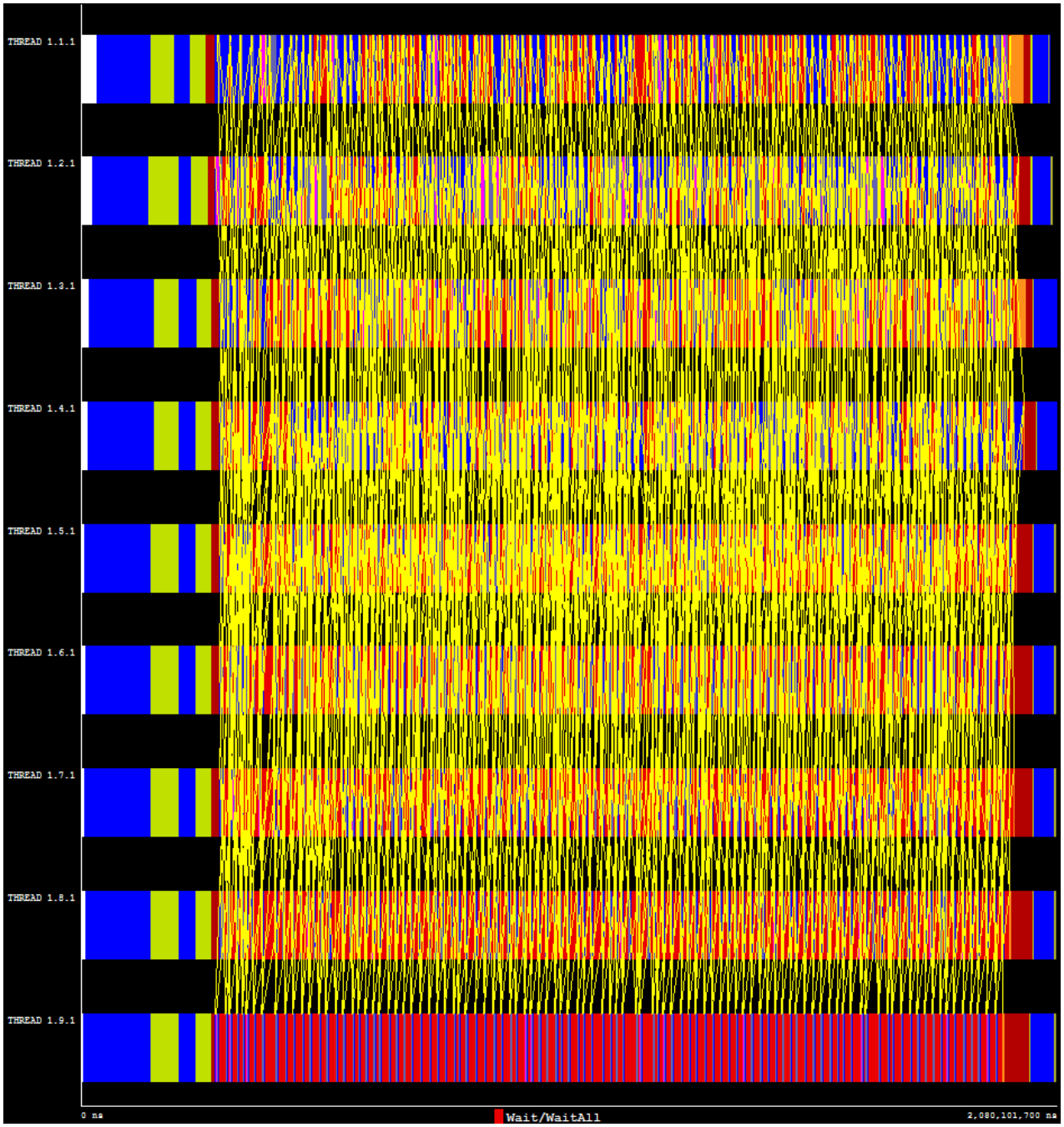
Στη συνέχεια γίνεται παρουσίαση των αποτελεσμάτων από το Paraver για 9 διεργασίες, για το αρχικό MPI.

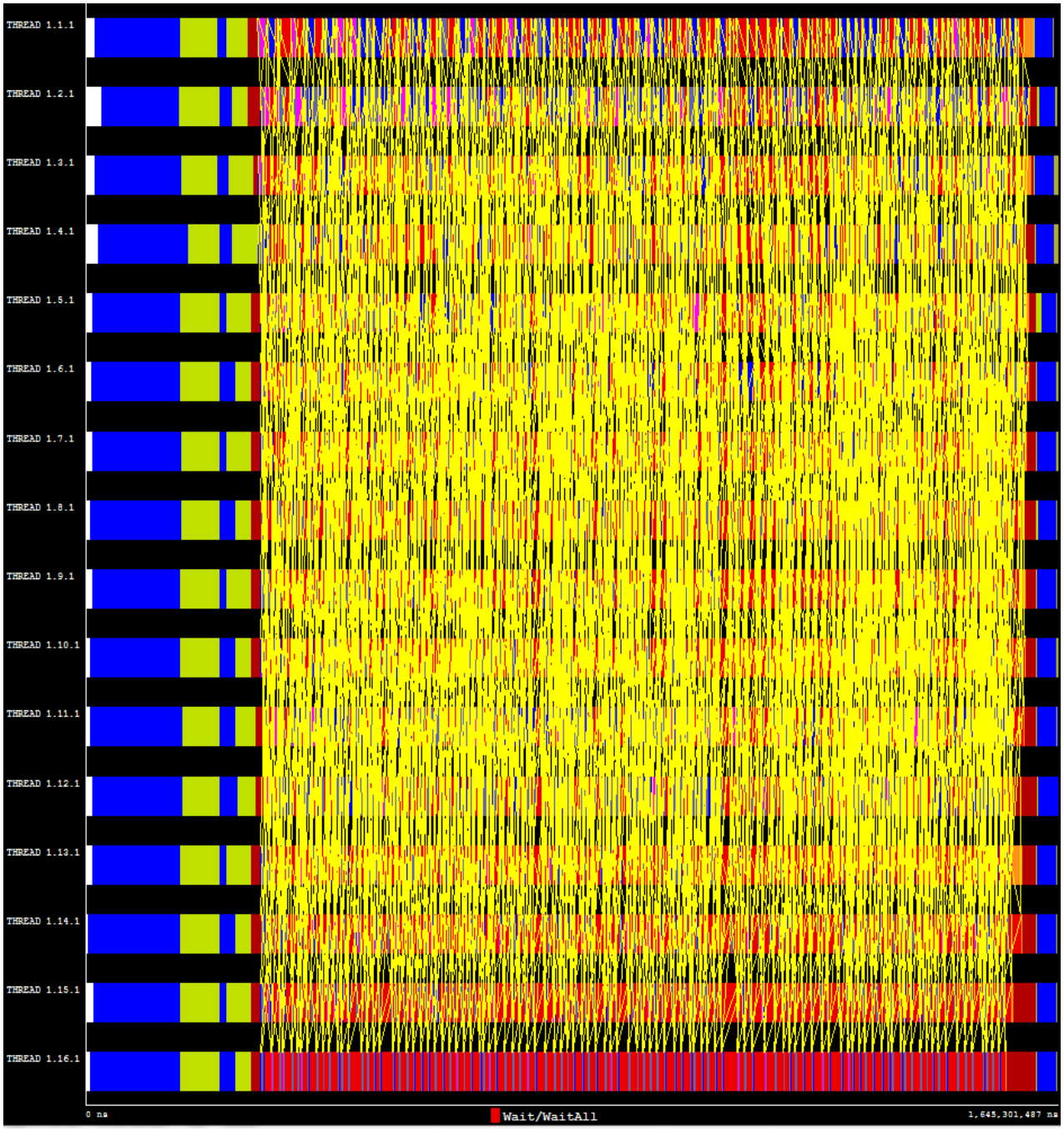


4.2 Βελτιωμένο MPI

Στη συνέχεια γίνεται παρουσίαση των αποτελεσμάτων από το Paraver για 4, 9 και 16 διεργασίες, για την βελτιωμένη υλοποίηση MPI.

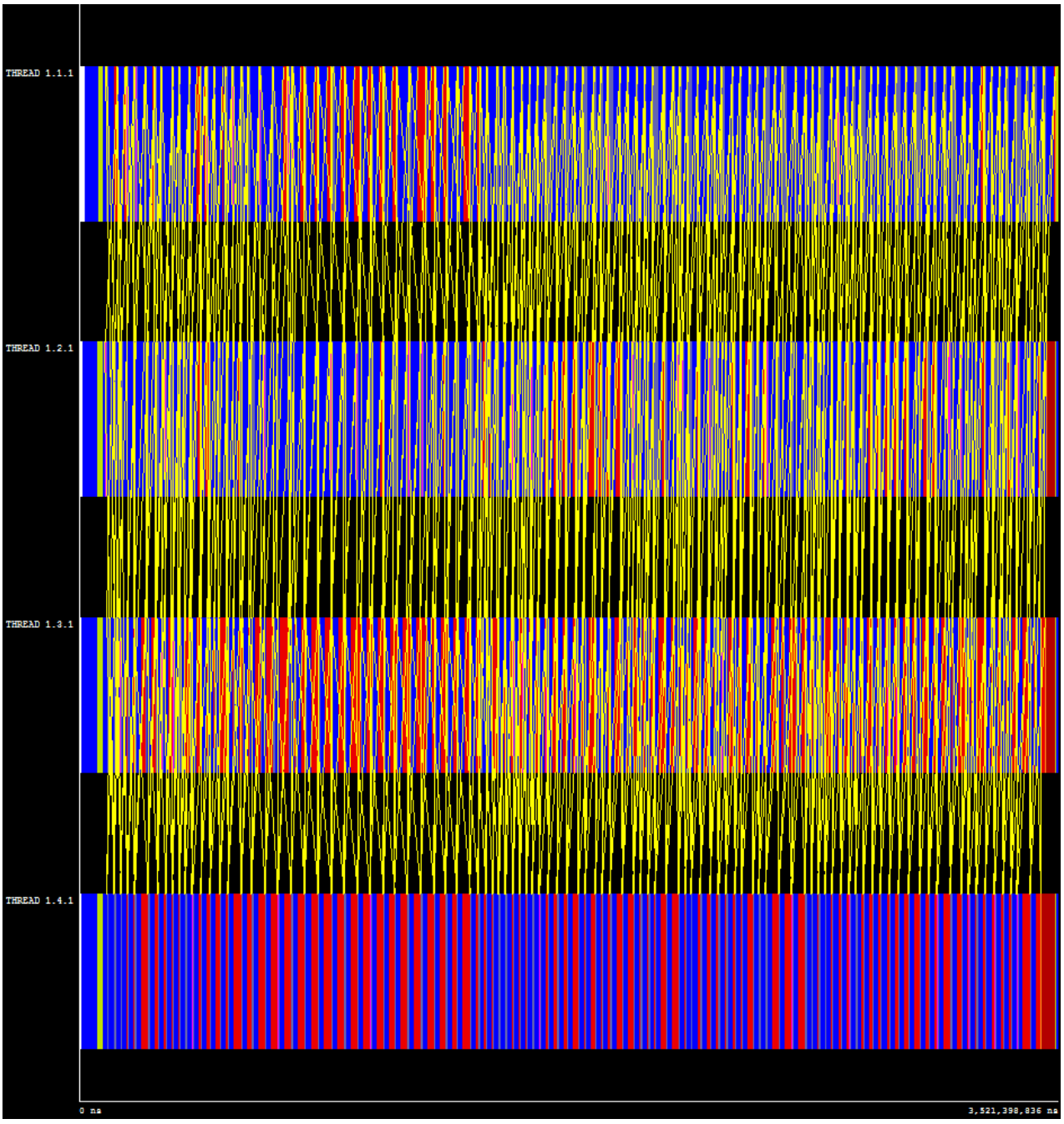


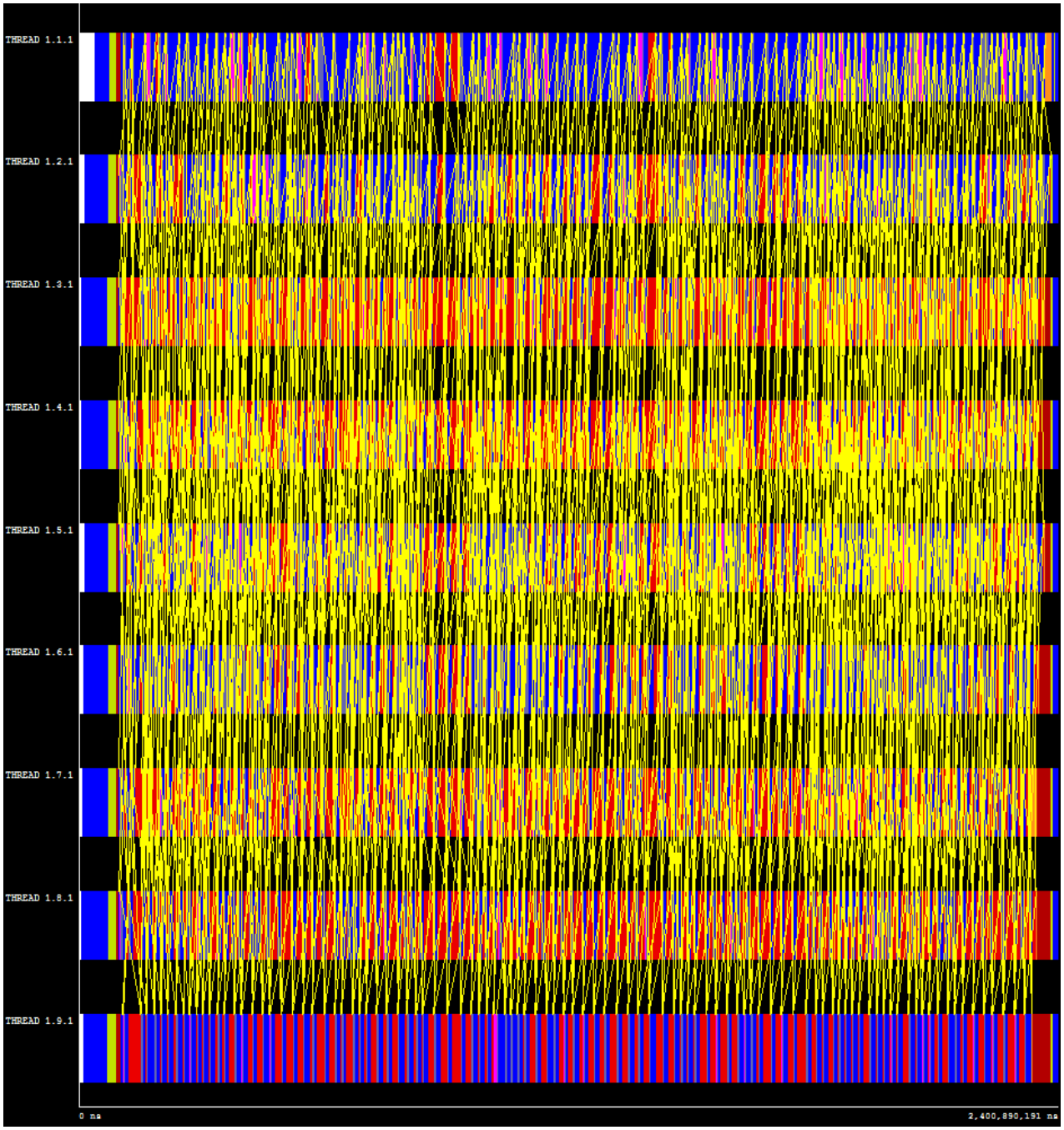

THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
THREAD 1.5.1
THREAD 1.6.1
THREAD 1.7.1
THREAD 1.8.1
THREAD 1.9.1
0 ns
Wait/WaitAll
2,080,101,700 ns


THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
THREAD 1.5.1
THREAD 1.6.1
THREAD 1.7.1
THREAD 1.8.1
THREAD 1.9.1
THREAD 1.10.1
THREAD 1.11.1
THREAD 1.12.1
THREAD 1.13.1
THREAD 1.14.1
THREAD 1.15.1
THREAD 1.16.1
0 ns
Wait/WaitAll
1,645,301,487 ns

4.3 Hybrid MPI/Open MP

Στη συνέχεια γίνεται παρουσίαση των αποτελεσμάτων από το Paraver για 4 και 9 διεργασίες, για την Υβριδική υλοποίηση MPI/OpenMp.




THREAD 1.1.1
THREAD 1.2.1
THREAD 1.3.1
THREAD 1.4.1
THREAD 1.5.1
THREAD 1.6.1
THREAD 1.7.1
THREAD 1.8.1
THREAD 1.9.1
0 ns
2,400,890,191 ns

5 ΜΕΤΑΓΛΩΤΗΣΗ ΚΑΙ ΕΚΤΕΛΕΣΗ

Στο κεφάλαιο αυτό παρουσιάζονται οδηγίες για την μεταγλώττιση και την εκτέλεση των υλοποιήσεων.

5.1 Αρχικό MPI

Υπάρχει Makefile που αναλαμβάνει την μεταγλώττιση των απαραίτητων αρχείων, καθώς και την διαγραφή τους (*make clean*).

Διαφορετικά οι εντολές που εκτελούνται είναι:

mpicc -o MPI_original mpi_heat2D.c

Για την εκτέλεση χρησιμοποιείται η εντολή:

mpiexec –f machines –n <πλήθος διεργασιών> MPI_original

5.2 Βελτιωμένο MPI

Υπάρχει Makefile που αναλαμβάνει την μεταγλώττιση των απαραίτητων αρχείων, καθώς και την διαγραφή τους (*make clean*).

Διαφορετικά οι εντολές που εκτελούνται είναι:

mpicc -o MPI_main MPI_main.c –lm

Για την εκτέλεση χρησιμοποιείται η εντολή:

mpiexec –f machines –n <πλήθος διεργασιών> MPI_main

5.3 Hybrid MPI/Open MP

Υπάρχει Makefile που αναλαμβάνει την μεταγλώττιση των απαραίτητων αρχείων, καθώς και την διαγραφή τους (*make clean*).

Διαφορετικά οι εντολές που εκτελούνται είναι:

mpicc -fopenmp -o OpenMP_main OpenMP_main.c –lm

Και για σειριακή μεταγλώττιση είναι:

mpicc -o Serial_OpenMP_main OpenMP_main.c -lm

Για την εκτέλεση χρησιμοποιείται η εντολή:

Mpiexec –f machines –n <πλήθος διεργασιών> OpenMP_main

Και αντίστοιχα:

Mpiexec –f machines –n <πλήθος διεργασιών> Serial_OpenMP_main

5.4 OpenMp

Μεταγλώττιση:

gcc –o OpenMP_pure Pure_OpenMP_main.c

Εκτέλεση:

OpenMP_pure

5.5 CUDA

Υπάρχει Makefile που αναλαμβάνει την μεταγλώττιση των απαραίτητων αρχείων, καθώς και την διαγραφή τους (*make clean*).

Για την εκτέλεση χρησιμοποιούνται οι εντολές:

- *qlogin*
- *make*
- *exit*
- *qsub run.sh*
- *more run.sh.o**

6 ΕΠΙΛΟΓΟΣ

Κλείνοντας, η άσκηση πραγματοποιήθηκε σε NetBeans 8.1 με remote build host το linux02.di.uoa.gr . Για το μέρος CUDA χρησιμοποιήθηκαν οι εντολές του εργαστηρίου και το Host 195.134.67.205 .

Οι μετρήσεις έγιναν στα linux της σχολής πρώτες πρωινές ώρες με σκοπό να αποφευχθούν καθυστερήσεις. Δυστυχώς δεν το κατάφερα αυτό διότι συμφοιτητές μου είχαν προγράμματα που έτρεχαν μήνες συνεχόμενα.

Επιπλέον, θα ήθελα να γίνει αντιληπτό ότι η εργασία ήταν να γίνει στα πλαίσια ομάδας αλλά δυστυχώς ο συνάδελφος άλλαξε γνώμη λίγο πριν την ξεκινήσω. Ζητώ την κατανόησή σας για τυχόν παραλήψεις και λάθη, διότι είχα προγραμματίσει να μην την κάνω ατομικά.

Ευχαριστώ πολύ,

Σπύρος Δελβινιώτης