ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ Η/Υ ΚΑΙ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΕΡΓΑΣΙΑ ΣΤΗ ΠΑΡΑΛΛΗΛΗ ΕΠΕΞΕΡΓΑΣΙΑ

ΣΥΝΕΡΓΑΤΕΣ:

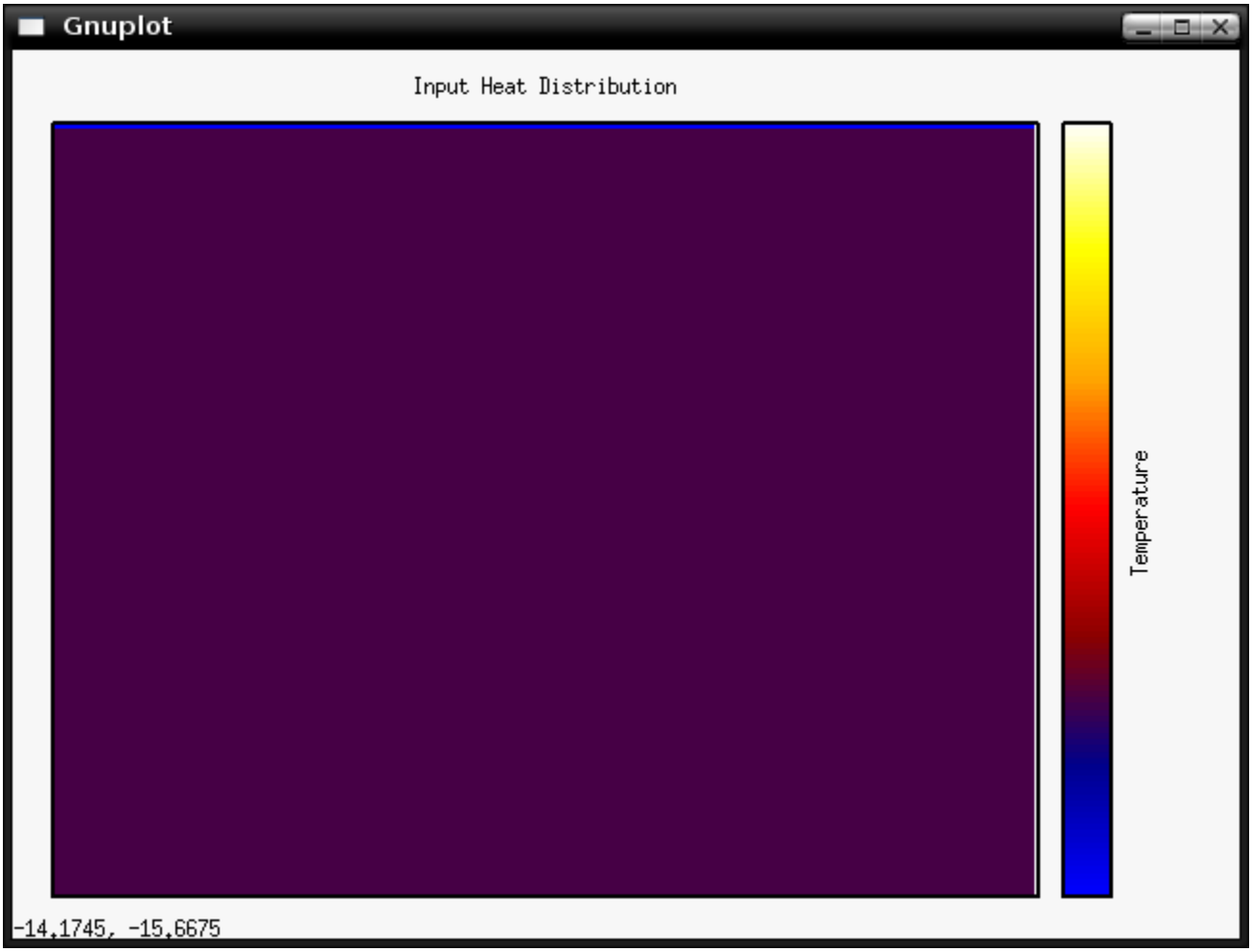
ΠΑΝΑΓΙΩΤΗΣ ΑΛΕΞΑΝΔΡΟΥ	Α.Μ.:3619
ΚΩΝ/ΝΟΣ ΑΡΑΒΑΝΗΣ	Α.Μ.:3628
ΠΑΝΑΓΙΩΤΗΣ ΓΕΩΡΓΙΟΥ	Α.Μ.:3638
ΙΩΑΝΝΗΣ ΟΙΚΟΝΟΜΙΔΗΣ	Α.Μ.:3711

ΠΑΤΡΑ ΜΑΙΟΣ 2008

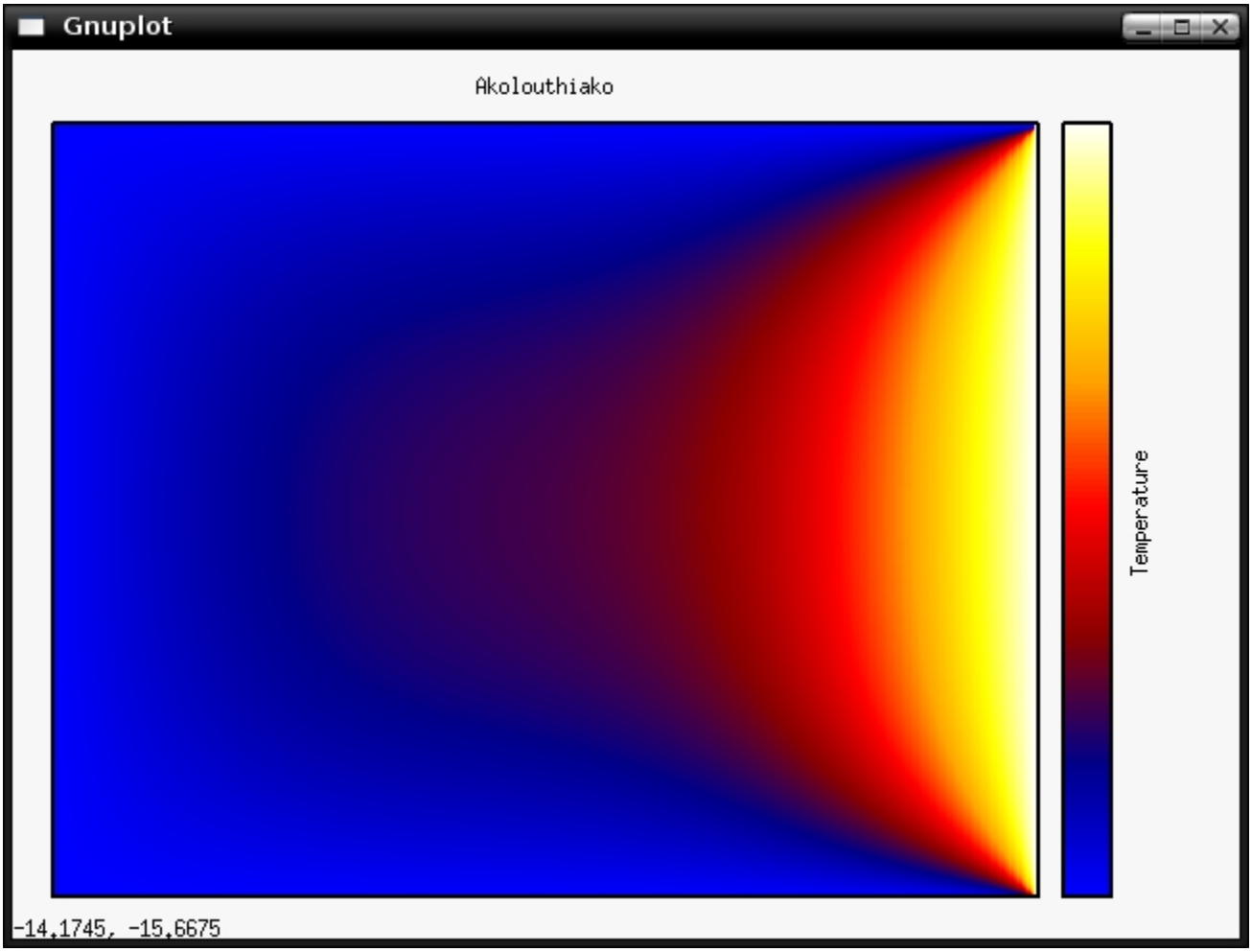
Η φετινή εργασία είχε σαν στόχο την παραλληλοποίηση του αλγορίθμου Jacobi με χρήση Threads και OpenMP και την μελέτη της απόδοσης τους με τη βοήθεια μιας ορθογώνιας πλάκας στην οποία υπολογίζουμε την κατανομή της θερμότητας.
Για τον λόγο αυτό υλοποιήσαμε έναν ακολουθιακό αλγόριθμο, ο οποίος χρησιμοποιήθηκε σαν σημείο αναφοράς για τα επόμενα.

Ακολουθιακό:
Για την ακολουθιακή υλοποίηση του αλγορίθμου Jacobi αρχικά διαβάσαμε από ένα αρχείο input.txt, που μας δόθηκε, τα δεδομένα μας, δηλαδή τις διαστάσεις της πλάκας, το όριο (EPSILON) το οποίο δεν πρέπει να ξεπερνά η διαφορά παλιάς και καινούριας θερμοκρασίας και τις αρχικές θερμοκρασίες των σημείων της πλάκας. Ακολούθως υλοποιήσαμε τον αλγόριθμο Jacobi όπως μας υποδείχτηκε από την εκφώνηση της άσκησης και τον χρονομετρήσαμε χρησιμοποιώντας τη συνάρτηση gettimeofday(). Στο σημείο αυτο πρέπει να σημειώσουμε οτι κάναμε την εξής αλλαγή στον Jacobi, αντικαταστήσαμε τους πίνακες με δείκτες ώστε να κάνουμε πιο γρήγορα την εκχώρηση των νέων τιμών στο before, καθώς το before πλέον απλά θα δείχνει στον after και ο after (για του οποίου τις τιμές δεν ενδιαφερόμαστε) στον before. Τέλος, αποθηκεύσαμε τα αποτελέσματα σε ένα output.txt αρχείο για την εξαγωγή του διαγράμματος κατανομής θερμότητας. Στο σημείο αυτό πρέπει να επισημάνουμε πως η χρονομέτρηση έγινε μόνο στο κώδικα του αλγορίθμου Jacobi καλώντας την gettimeoday πριν την αρχή του και στο τέλος του και υπολογίζοντας την διαφορά.

Ακολουθεί η κατανομή θερμότητας του input.txt (είναι κοινή για όλες τις υλοποιήσεις):



Ακολουθεί η κατανομή θερμότητας για το ακολουθιακό:



Ο χρόνος που διήρκεσε η επεξεργασία για την συγκεκριμένη είσοδο ήταν: 55.336603sec

ThreadsA:
Όσον αφορά τον τρόπο ανάγνωσης και γραψίματος δεδομένων καθώς και χρονομέτρησης του παράλληλου τμήματος ισχύει ότι στο ακολουθιακό κομμάτι. Σε αυτή την περίπτωση μας ζητήθηκε να χωρίσουμε το πλέγμα των σημείων προς ανανέωση σε ισομερή τμήματα, ο αριθμός των οποίων είναι ίσος με τον αριθμό των νημάτων.
Ξεκινώντας δημιουργούμε τον αριθμό των νημάτων που χρειαζόμαστε και αναθέτουμε στο καθένα την εκτέλεση του Jacobi(int t_i).
Ξεκινώντας την συνάρτηση ορίζουμε τις γραμμές του πλέγματος που θα αναλάβει το κάθε νήμα με βάση το t_i (id του νήματος) και ανανεώνουμε τις παλιές θερμοκρασίες με τις νέες τους τιμές. Τα προβλήματα που συναντήσαμε σχετικά με την παραλληλοποίηση ήταν τέσσερα εξού και η χρήση των τριών barriers και του ενός mutex, τα οποία θα αναλύσουμε τώρα.
Θα ξεκινήσουμε με την χρήση του δεύτερου barrier ο οποίος ενεργοποιήθηκε σε αυτό το σημείο ώστε όλα τα νήματα να έχουν τρέξει τα παραπάνω loop που βρίσκουν τις καινούριες τιμές που πρόκειται να πάρει η πλάκα καθώς και να έχουν ανανεώσει την τιμή του diff_thread (private μεταβλητή για κάθε νήμα).
Ακολούθως μπλοκάρουμε το κομμάτι κώδικα που ψάχνει να βρει ποιο είναι το

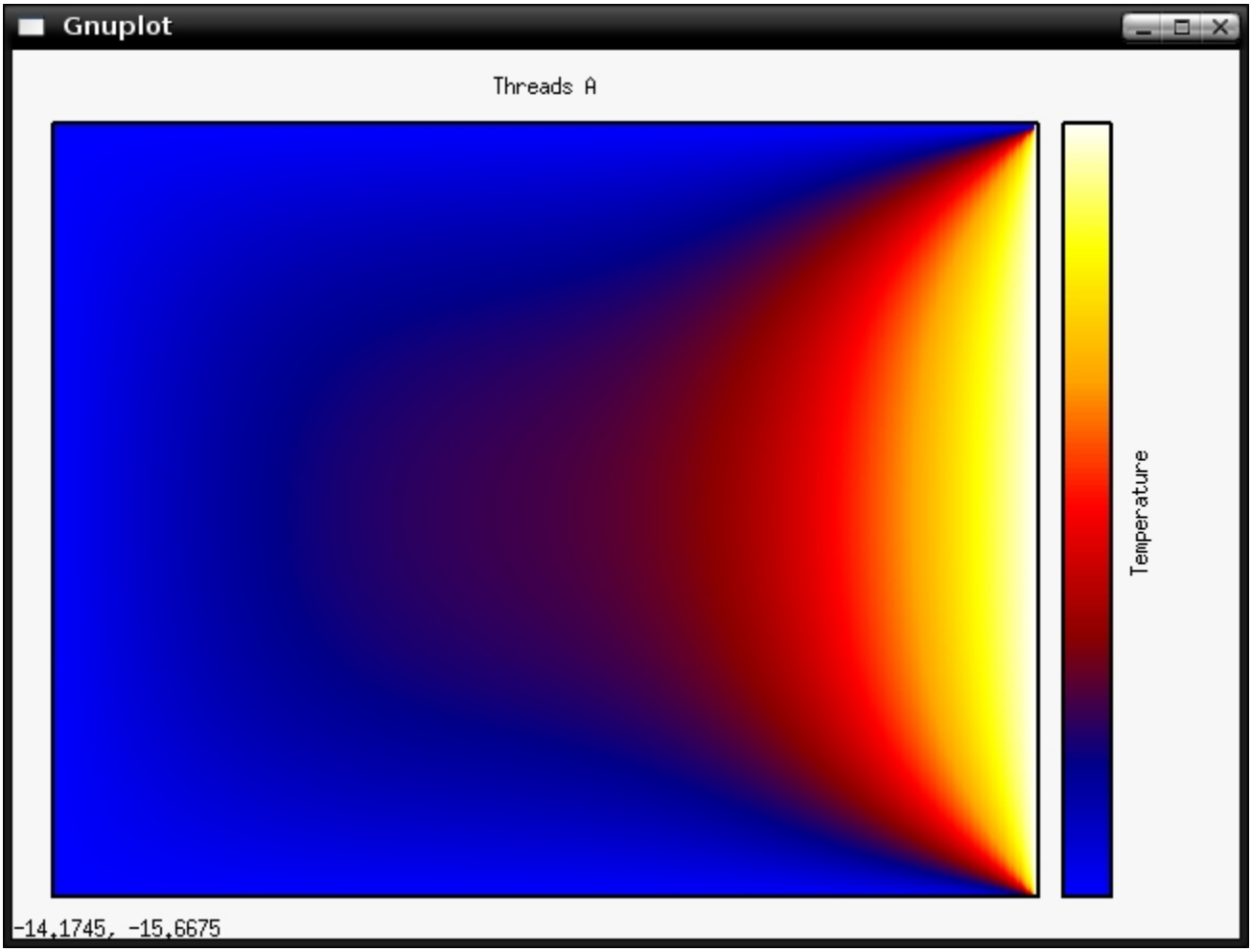
μεγαλύτερο diff για να αποφευχθεί να καταγραφεί λανθασμένη τιμή. Π.χ. αν δεν υπήρχε mutex και 2 νήματα έκαναν ταυτόχρονο έλεγχο του if και τα δύο είχαν diff_thread μεγαλύτερο του diff, τότε ενδέχεται να κρατηθεί η μικρότερη τιμή από τα δύο αυτά diff_threads. Για την καλύτερη κατανόηση παραθέτουμε και το κομμάτι αυτό του κώδικα:

```
//we make a mutex lock in order to chech the diff_threads of all processes

//and save the biggest one to the variable with the name diff

pthread_mutex_lock(&my_mutex);

if(diff < diff_thread)

{

     diff=diff_thread;

}

pthread_mutex_unlock(&my_mutex);        //unlock the thread
```

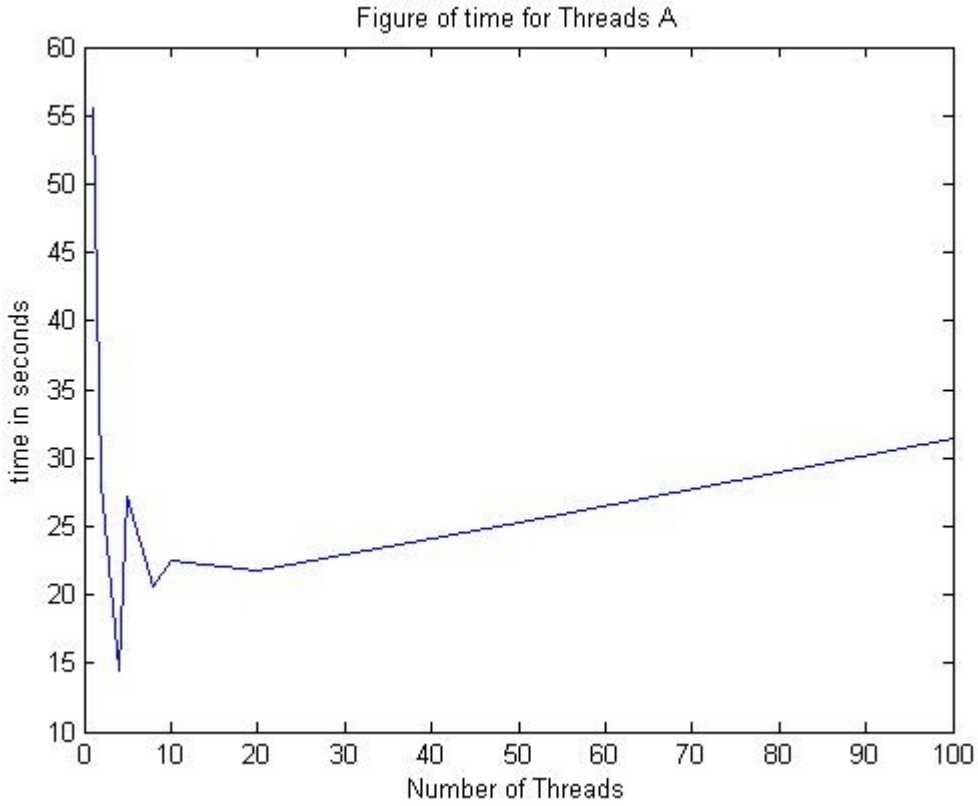
Το τρίτο barrier μπήκε ώστε να αναγκάσει όλα τα νήματα να περιμένουν να τελειώσουν τον έλεγχο για το ποιο έχει το μεγαλύτερο diff_thread ώστε το global diff να έχει την πραγματικά μεγαλύτερη τιμή για να μην γίνει λανθασμένη έξοδος από κάποιο thread.
Το πρώτο barrier, όπως εύκολα μπορεί να καταλάβει κανείς έχει μπει για να περιμένουμε όλα τα νήματα να έχουν ολοκληρώσει τον έλεγχο με το epsilon πριν δώσουμε καινούρια τιμή στο diff. Δηλαδή σε περίπτωση που δεν υπήρχε θα μπορούσε το πρώτο νήμα που κάνει την ανανέωση του diff με 0.0 να επηρεάσει τον έλεγχο των άλλων με το epsilon με αποτέλεσμα τα υπόλοιπα νήματα να κάνουν λανθασμένη έξοδο από τον Jacobi.

Κλείνοντας, να αναφέρουμε ότι στον κώδικα της main μετά την δημιουργία των νημάτων (create) περιμένουμε (join) να καταστραφούν όλα τα νήματα πριν συνεχίσουμε το απομείνων ακολουθιακό κομμάτι.

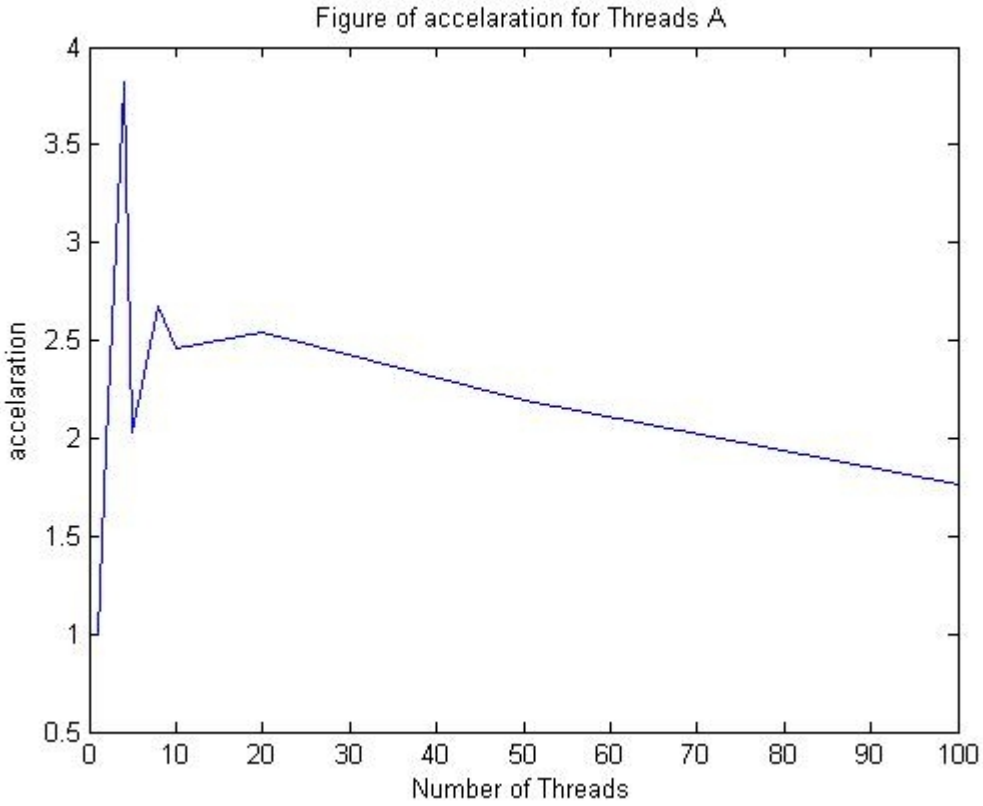
Ακολουθεί η κατανομή θερμότητας για τα ThreadsA:



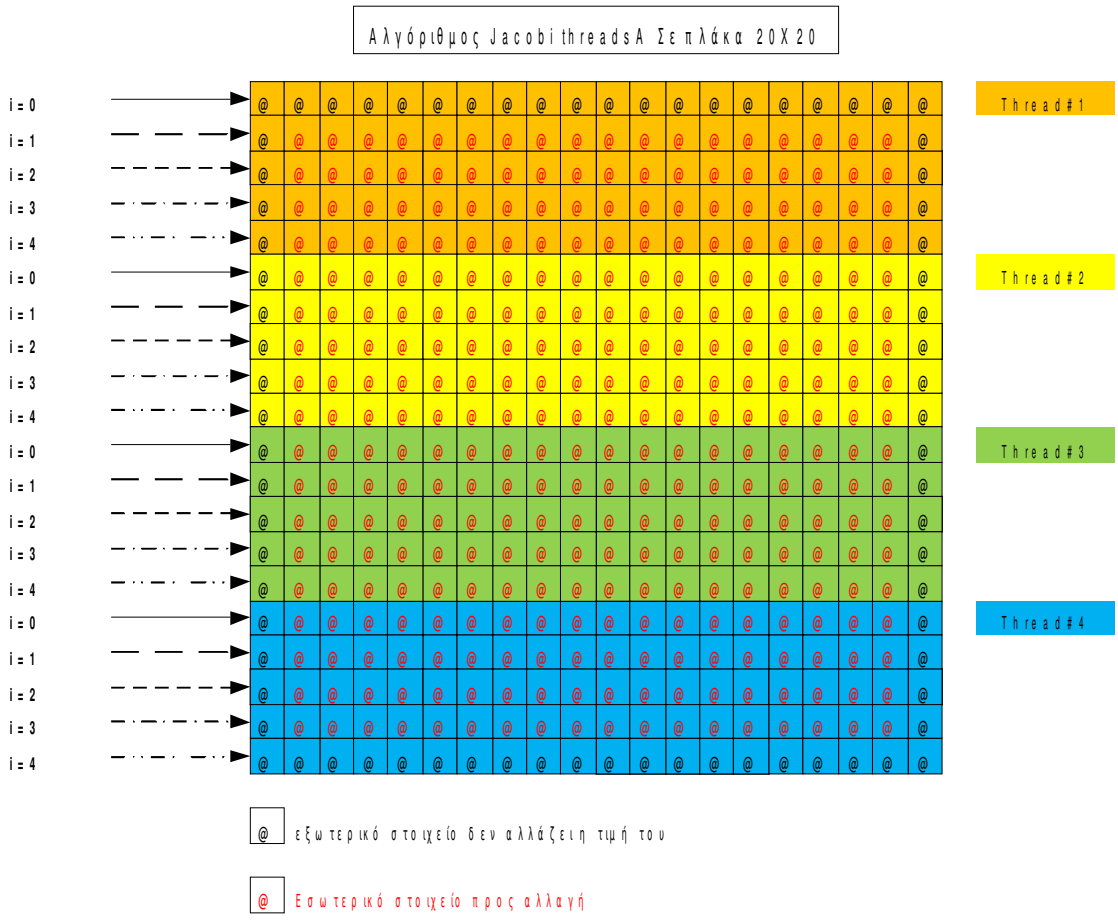
Ο χρόνος που διήρκεσε η επεξεργασία για την συγκεκριμένη είσοδο ήταν και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



Η επιτάχυνση time_of(Akolouthiako)/time_of(ThreadsA) για την συγκεκριμένη είσοδο και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



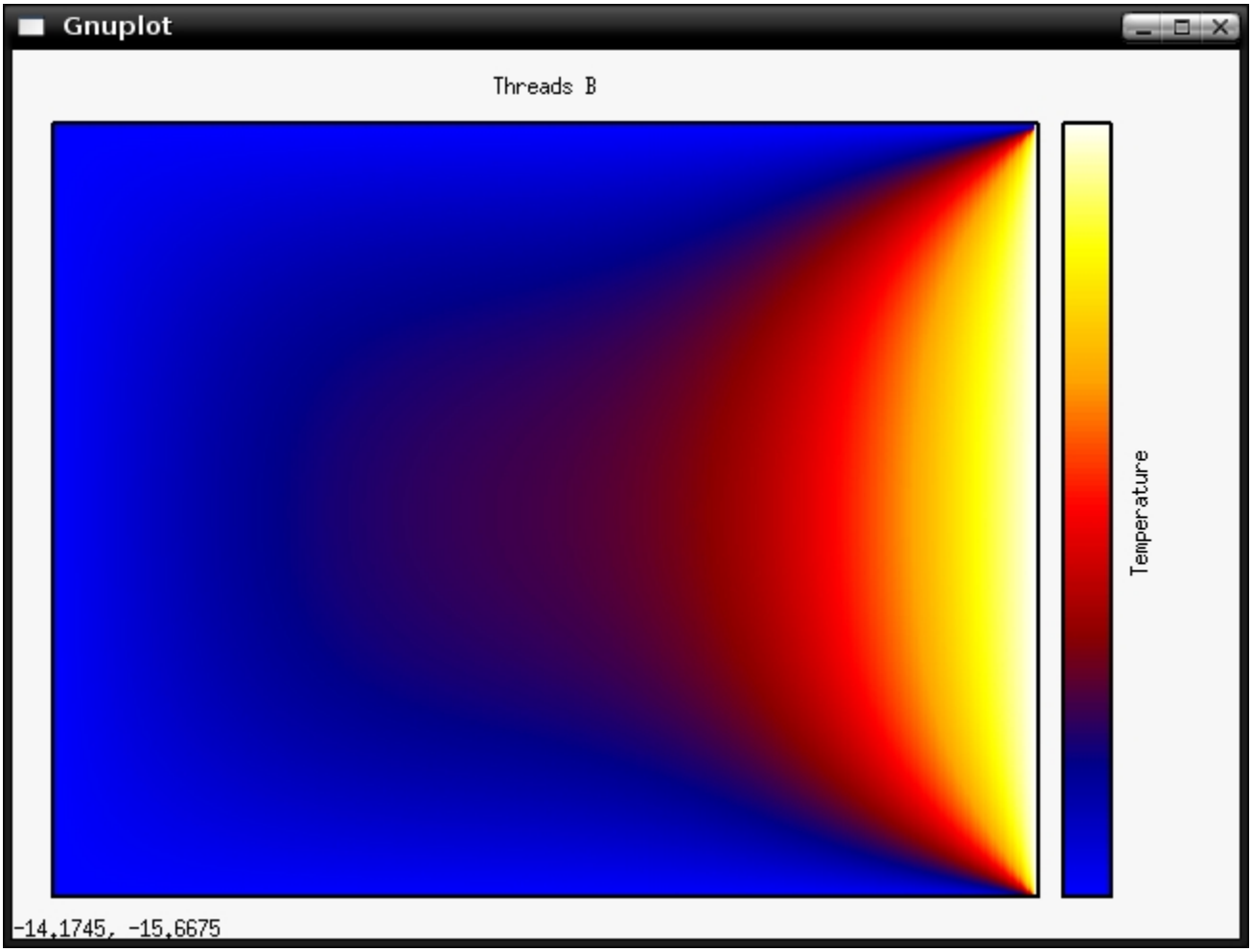
Σχηματική αναπαράσταση στο τρόπο επεξεργασίας της πλάκας:



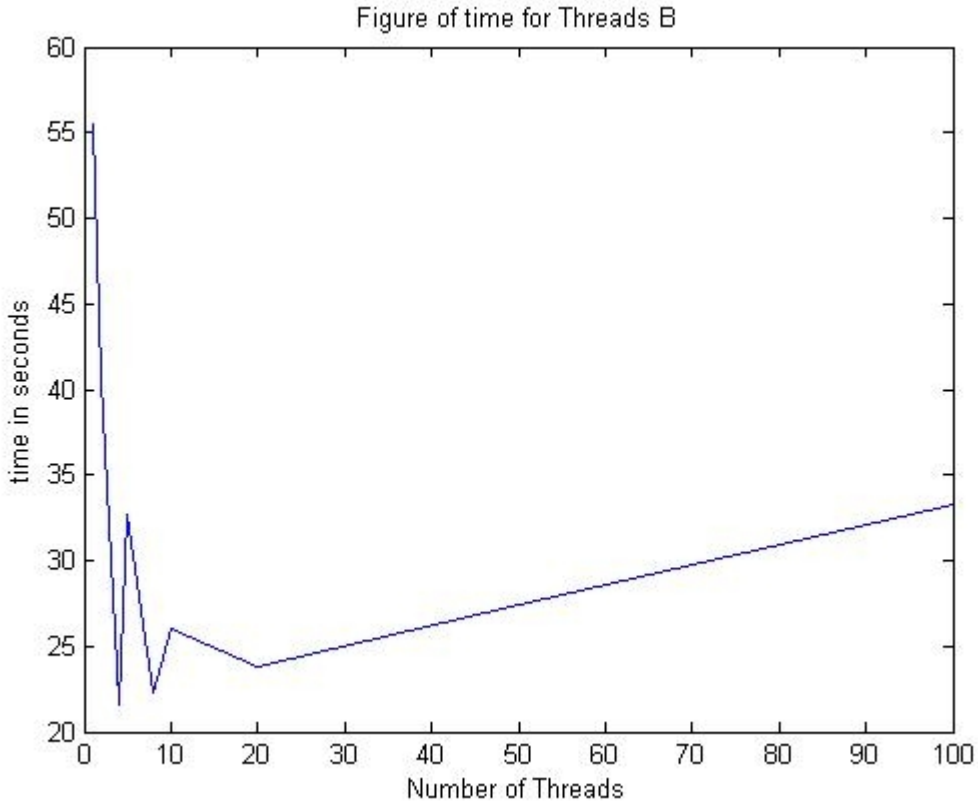
ThreadsB:

Όσον αφορά τον τρόπο ανάγνωσης και γραψίματος δεδομένων καθώς και χρονομέτρησης του παράλληλου τμήματος ισχύει ότι στο ακολουθιακό κομμάτι. Σε αυτή την περίπτωση υλοποίησης με νήματα μας ζητήθηκε το κάθε thread να επεξεργάζεται μια γραμμή κάθε φορά μέχρι να ανανεωθούν όλες οι γραμμές για κάθε γύρο εκτέλεσης.
Ξεκινώντας δημιουργούμε τον αριθμό των νημάτων που χρειαζόμαστε και αναθέτουμε στο καθένα την εκτέλεση του Jacobi(int t_i) και ανανεώνουμε τις παλιές θερμοκρασίες με τις νέες τους τιμές.
Ξεκινώντας χωρίζουμε την πλάκα σε τμήματα ανάλογα με τον αριθμό των νημάτων. Ύστερα αναθέτουμε σε κάθε νήμα μια γραμμή από το κάθε τμήμα.
Τα προβλήματα που συναντήσαμε σχετικά με την παραλληλοποίηση ήταν και εδώ τέσσερα εξού και η χρήση των τριών barriers και του ενός mutex, για τον ίδιο ακριβώς λόγο όπως στο ThreadsA.

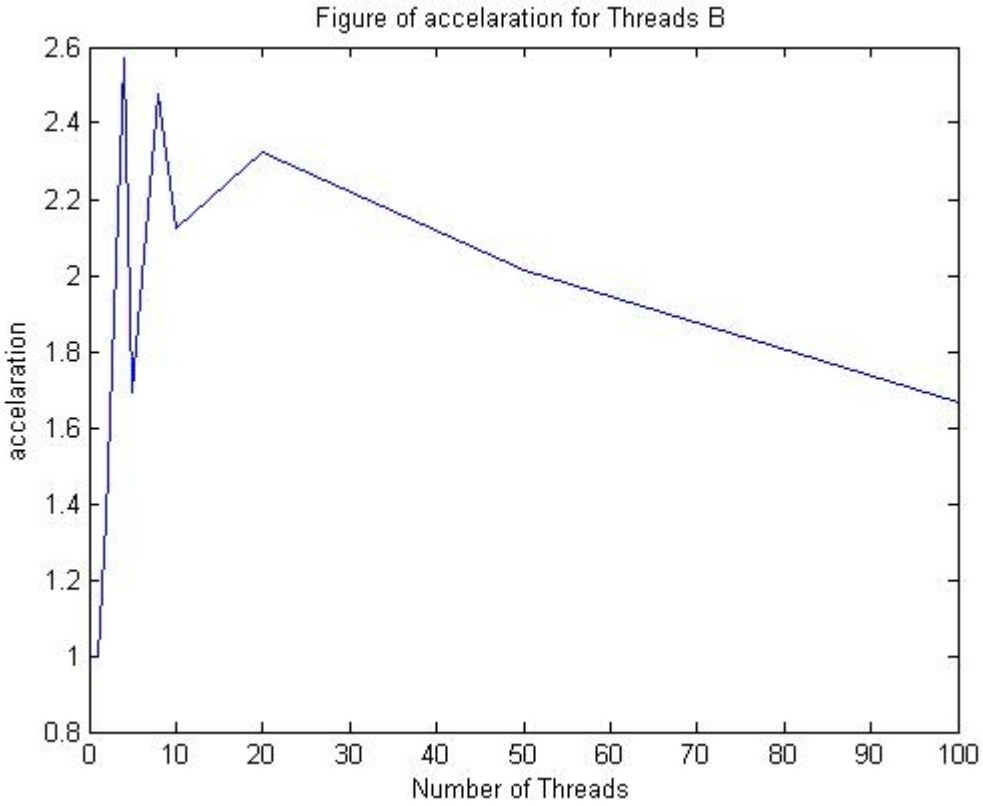
Ακολουθεί η κατανομή θερμότητας για τα ThreadsB:



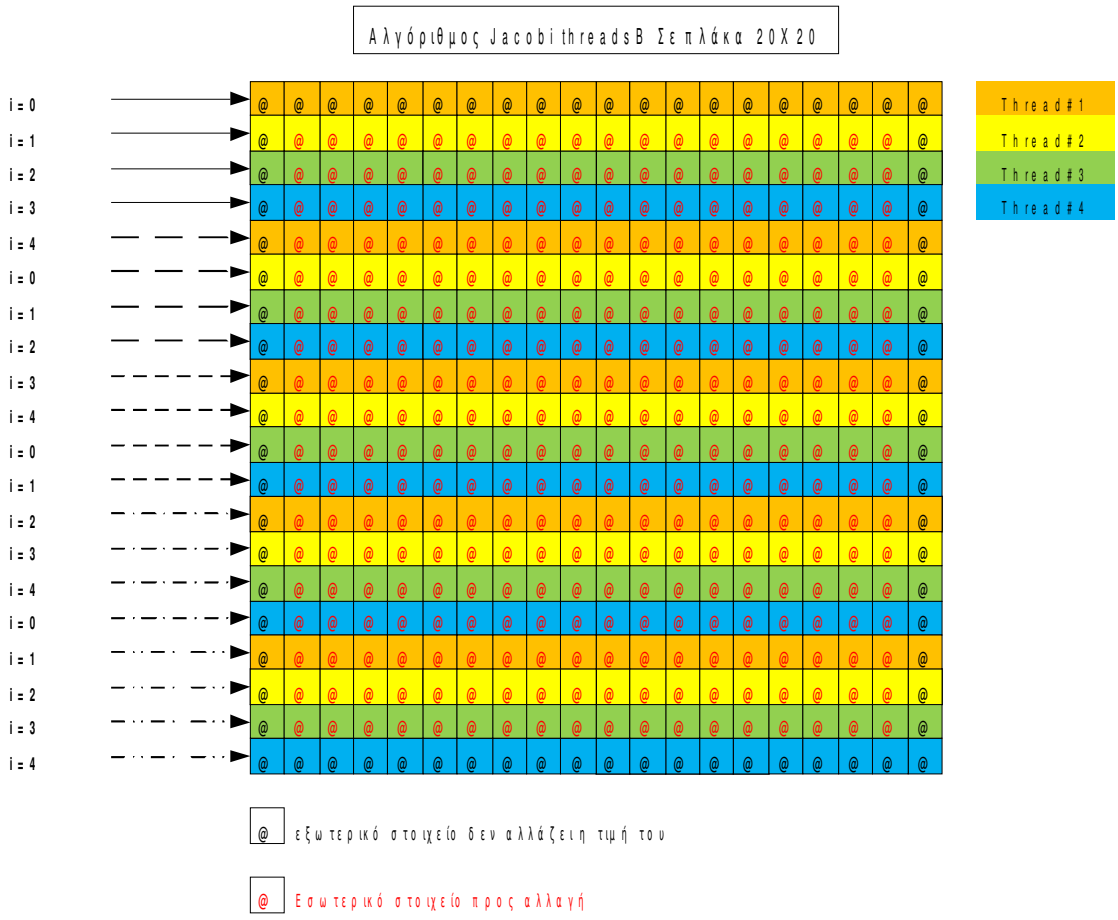
Ο χρόνος που διήρκεσε η επεξεργασία για την συγκεκριμένη είσοδο ήταν και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



Η επιτάχυνση time_of(Akolouthiako)/time_of(ThreadsB) για την συγκεκριμένη είσοδο και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



Σχηματική αναπαράσταση στο τρόπο επεξεργασίας της πλάκας:



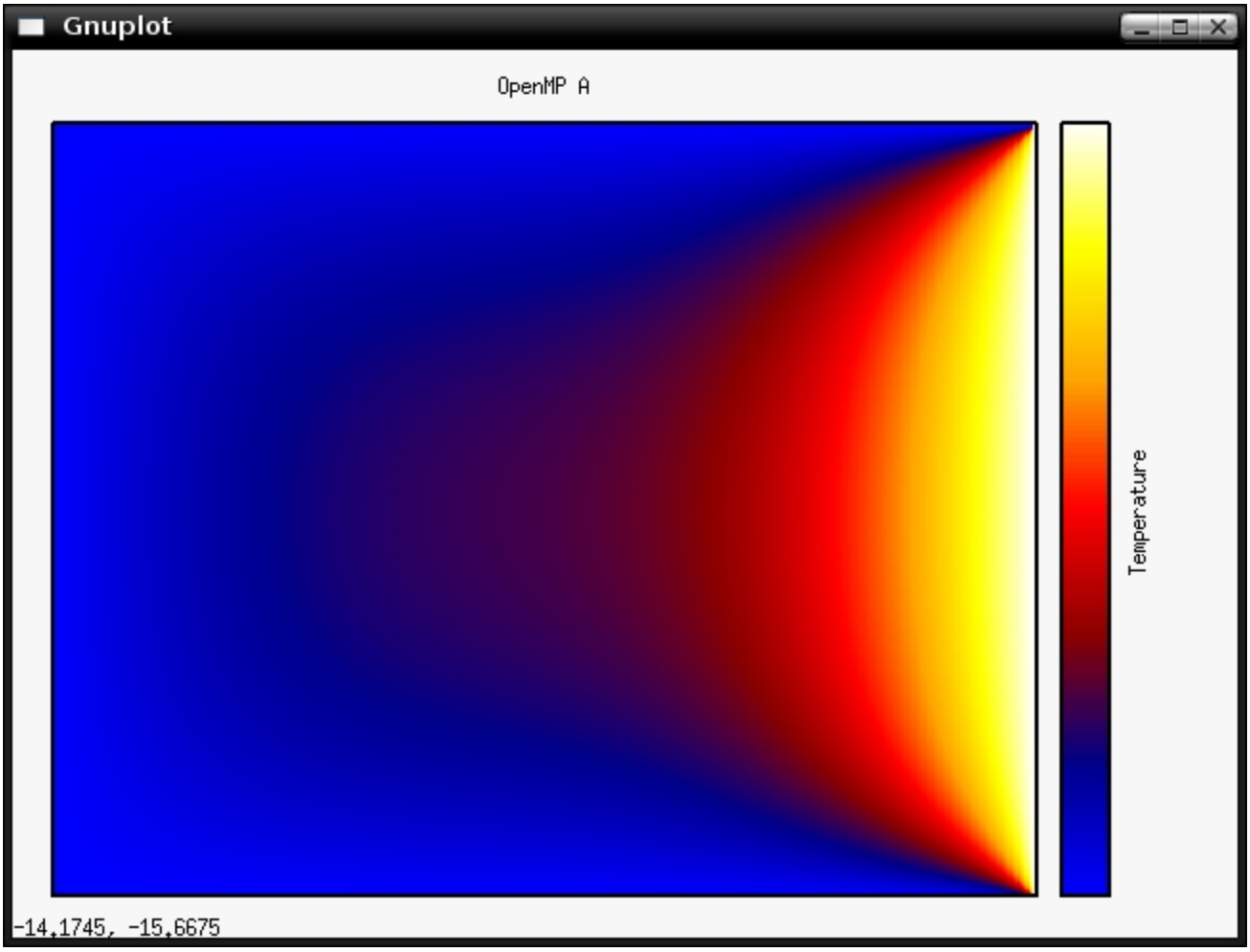
OpenMPA:

Όσον αφορά τον τρόπο ανάγνωσης και γραψίματος δεδομένων καθώς και χρονομέτρησης του παράλληλου τμήματος ισχύει ότι στο ακολουθιακό κομμάτι. Σε αυτή την περίπτωση μας ζητήθηκε με την χρήση OpenMP να υλοποιήσουμε μια υψηλότερου επιπέδου πολυνηματική εκδοχή θέτοντας την παράμετρο δρομολόγησης (schedule) στη στατική (static) επιλογή.

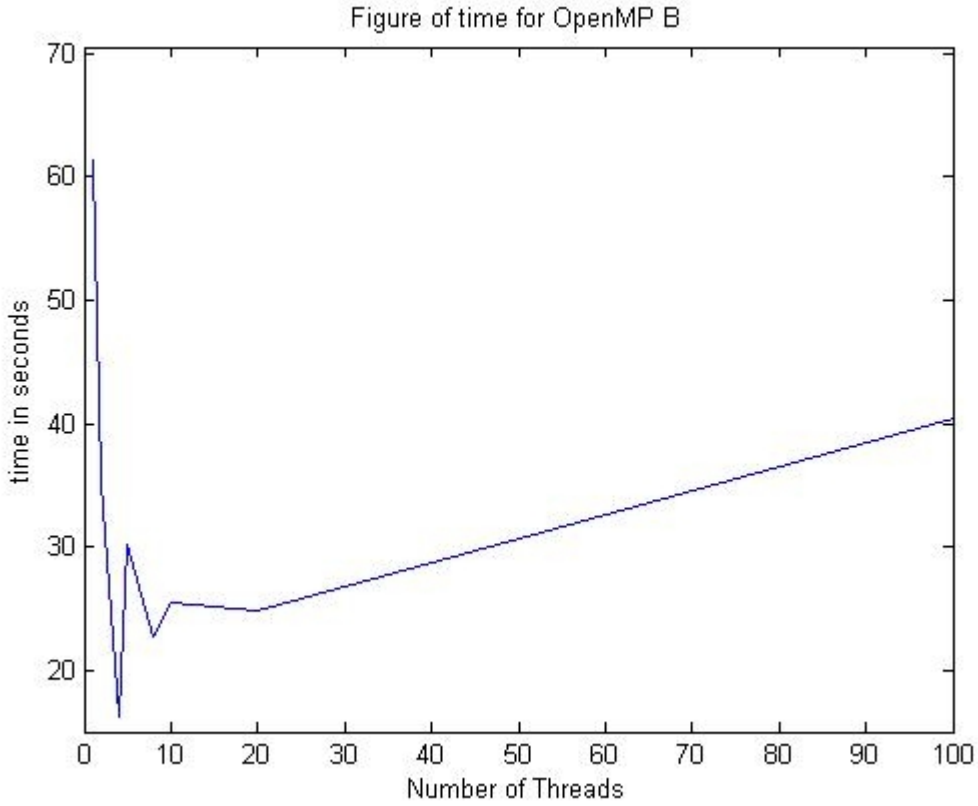
Η παραλληλοποίηση του αλγορίθμου έγινε εσωτερικά του ατέρμονος (όπως παρουσιάζεται στο ακολουθιακό) βρόχου καθώς δεν μπορούσαμε να σπάσουμε το for loop σε chunks λόγω της μη γνώσης των επαναλήψεων του for loop. Επομένως, παραλληλοποιήσαμε τον εσωτερικό κώδικα του loop. Τα προβλήματα που παρουσιάστηκαν ήταν τα εξής :Αρχικά τις επαναλήψεις του κάθε loop έπρεπε να τις σπάσουμε σε chunks και λόγω του ότι έπρεπε να χρησιμοποιήσουμε την παράμετρο δρομολόγησης ως static, κάναμε χρήση της #pragma omp for schedule(static) για κάθε επόμενο for loop. Συνεχίζοντας κάναμε τοποθέτηση του κώδικα #pragma omp flush(var) όπου χρειαζόταν για να προλαβαίνουμε να ανανεώνουμε την μνήμη με τις καινούριες τιμές του var. Επίσης, έπρεπε να θέσουμε τον κώδικα που βρίσκει το μεγαλύτερο diff όλων των threads σε mutex (#pragma omp critical) για τον ίδιο λόγο που τοποθετήσαμε σε mutex το ίδιο κομμάτι στα Threads. Τέλος, λόγω του ότι δεν μπορούμε να κάνουμε χρήση της εντολής break σε κομμάτι κώδικα που παραλληλοποιείται με OpenMP τη συνθήκη για να κάνουμε έξοδο από τον

αλγόριθμο Jacobi την βάλαμε στο ατέρμον for loop.

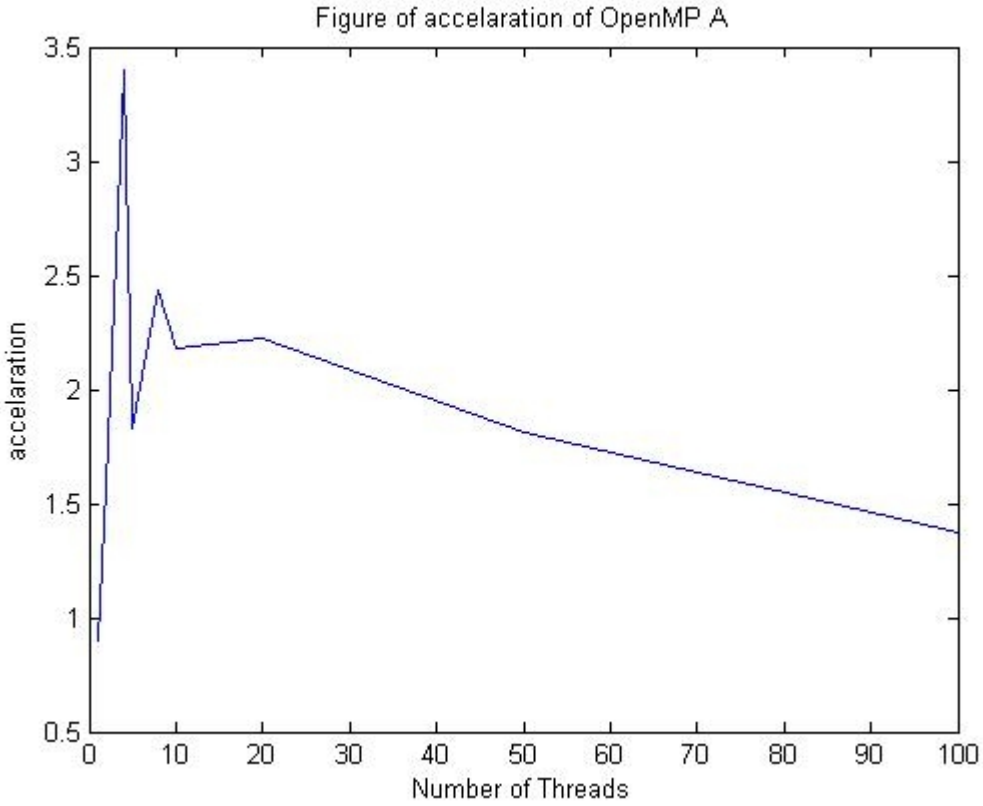
Ακολουθεί η κατανομή θερμότητας για τα OpenMPA:



Ο χρόνος που διήρκεσε η επεξεργασία για την συγκεκριμένη είσοδο ήταν και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:

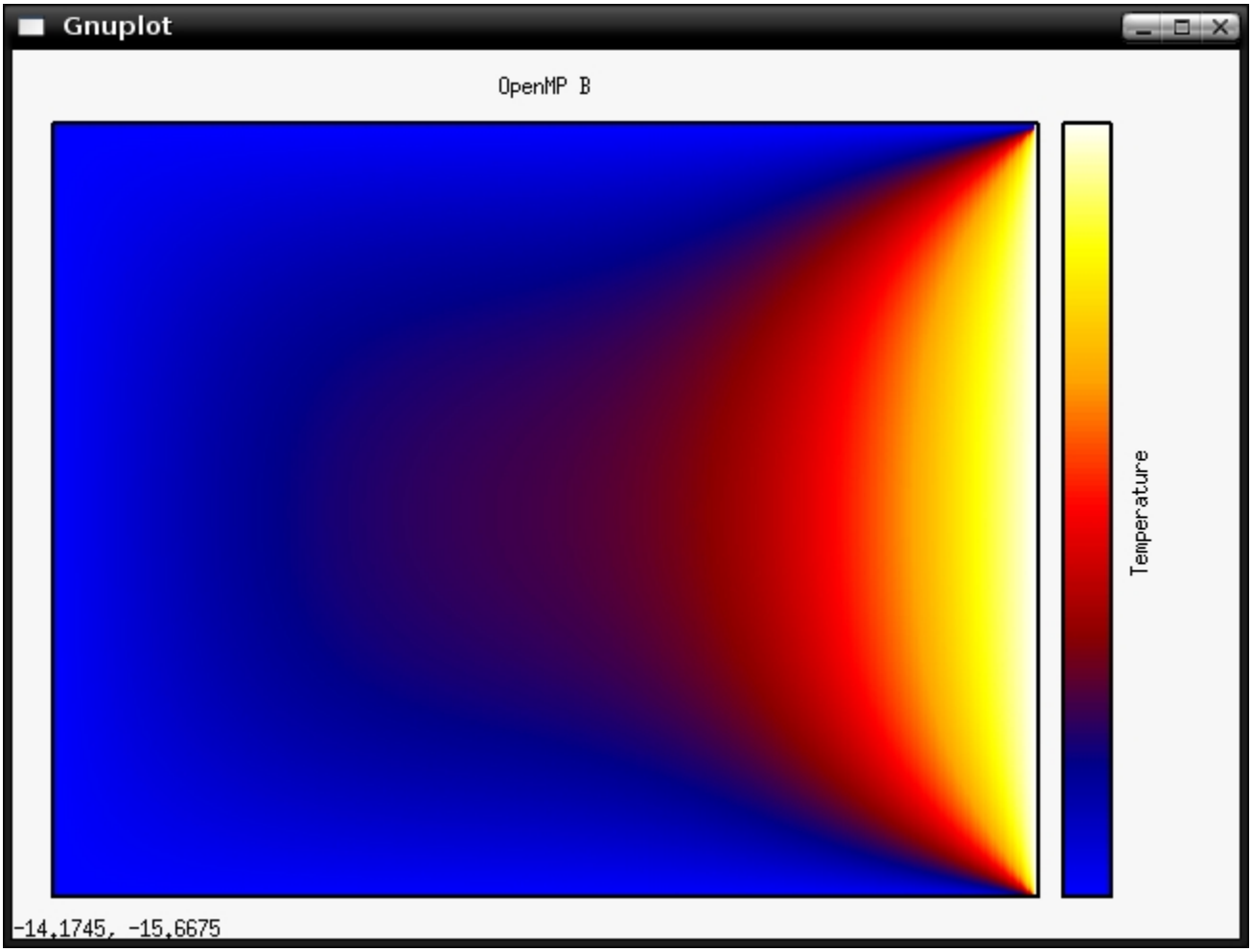


Η επιτάχυνση time_of(Akolouthiako)/time_of(OpenMPA) για την συγκεκριμένη είσοδο και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:

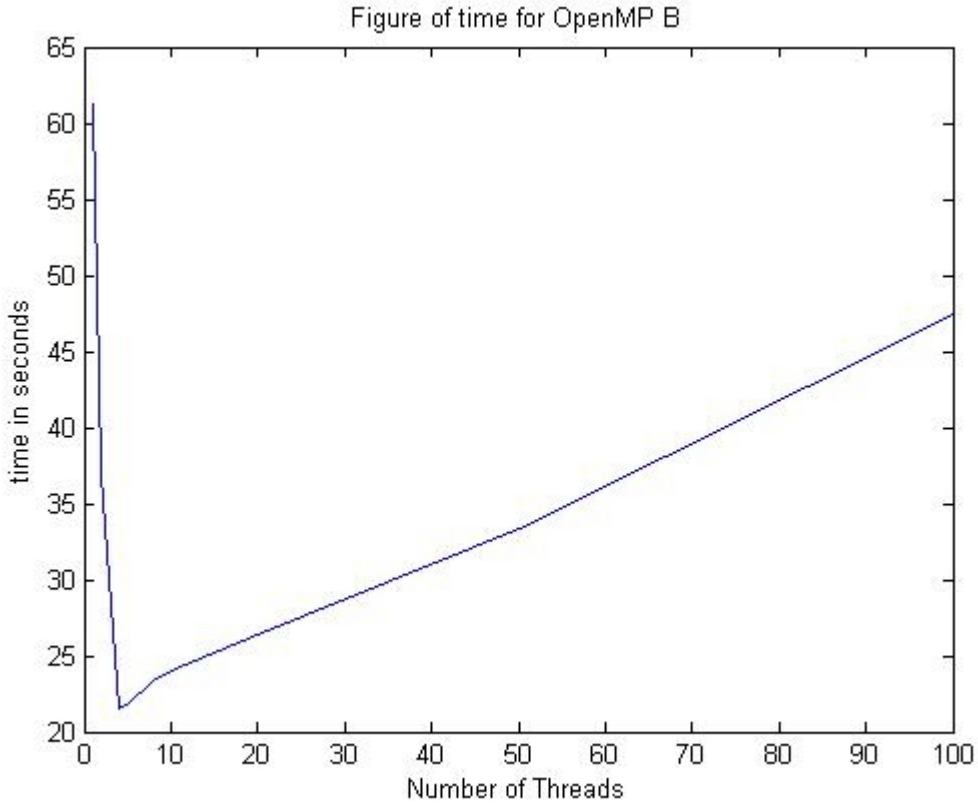


OpenMPB:
Όσον αφορά τον τρόπο ανάγνωσης και γραψίματος δεδομένων καθώς και χρονομέτρησης του παράλληλου τμήματος ισχύει ότι στο ακολουθιακό κομμάτι. Σε αυτή την περίπτωση μας ζητήθηκε με την χρήση OpenMP να υλοποιήσουμε μια υψηλότερου επιπέδου πολυνηματική εκδοχή θέτοντας την παράμετρο δρομολόγησης (schedule) στην επιλογή οδήγησης (guided).
Η παραλληλοποίηση του αλγορίθμου έγινε με τον ίδιο ακριβώς τρόπο όπως και στα OpenMPA με καμία άλλη διαφορά πέραν του ότι τις επαναλήψεις του κάθε for loop εσωτερικά του ατέρμονος βρόχου (όπως εμφανίζεται στο ακολουθιακό) έπρεπε να τις σπάσουμε σε chunks χρησιμοποιώντας την παράμετρο δρομολόγησης ως guided (#pragma omp for schedule(guided)).

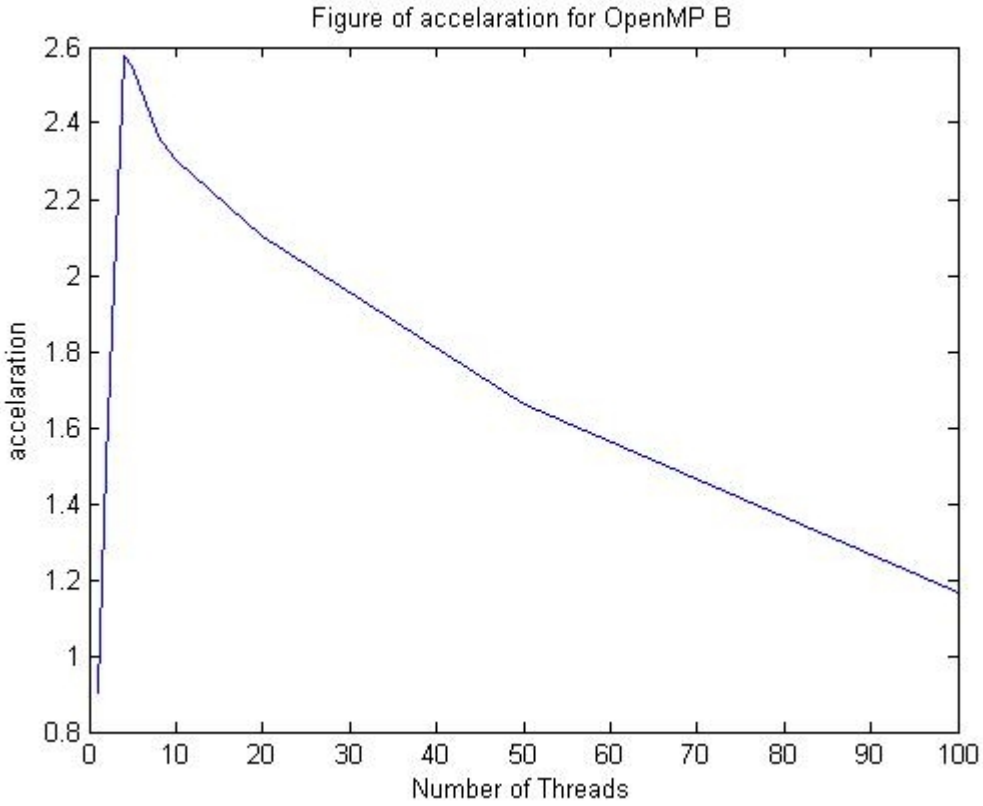
Ακολουθεί η κατανομή θερμότητας για τα OpenMPB:



Ο χρόνος που διήρκεσε η επεξεργασία για την συγκεκριμένη είσοδο ήταν και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



Η επιτάχυνση time_of(Akolouthiako)/time_of(OpenMPB) για την συγκεκριμένη είσοδο και επιλέγοντας κάθε φορά το πλήθος των νημάτων φαίνεται στην παρακάτω γραφική παράσταση:



Σχολιασμός αποτελεσμάτων:
Συγκεντρωτικοί πίνακες αποτελεσμάτων

Πίνακας παρουσίασης χρόνων εκτελέσεων

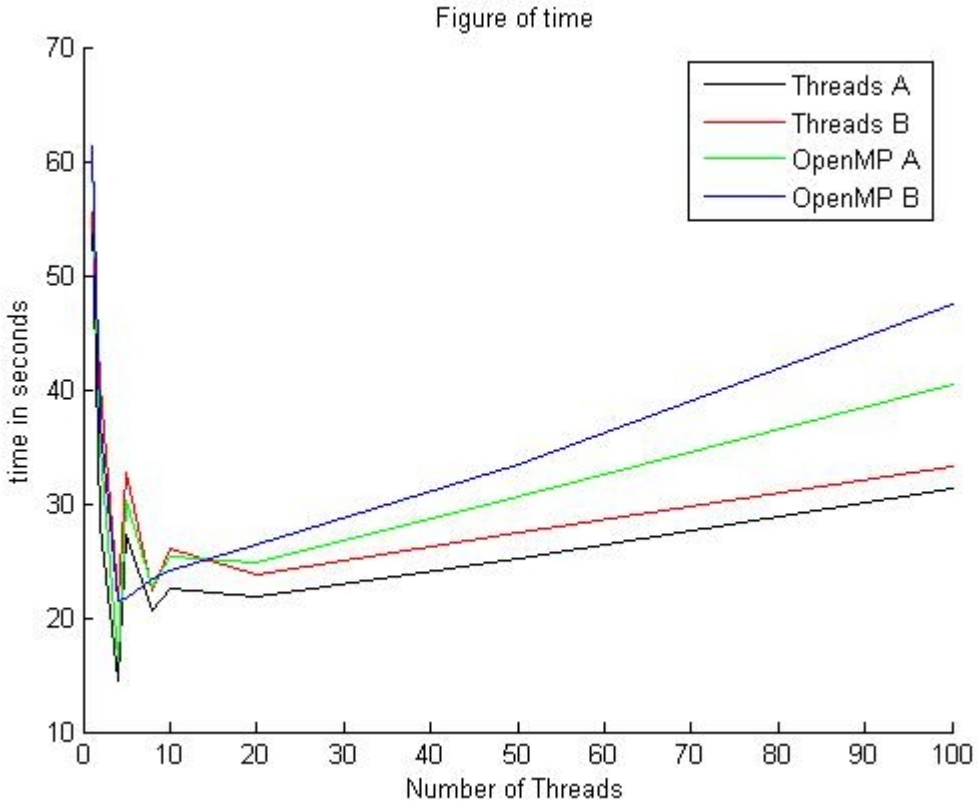
	Seconds			
No of Threads	Threads A	Threads B	OpenMP A	OpenMP B
1	55,545	55,54	61,327	61,264
2	28,224	40,637	35,189	36,845
4	14,5	21,525	16,273	21,492
5	27,214	32,716	30,185	21,723
8	20,667	22,336	22,766	23,371
10	22,516	26,031	25,433	24,041
20	21,758	23,797	24,856	26,341
50	25,245	27,462	30,612	33,336
100	31,388	33,221	40,392	47,428

Πίνακας παρουσίασης των επιταχύνσεων

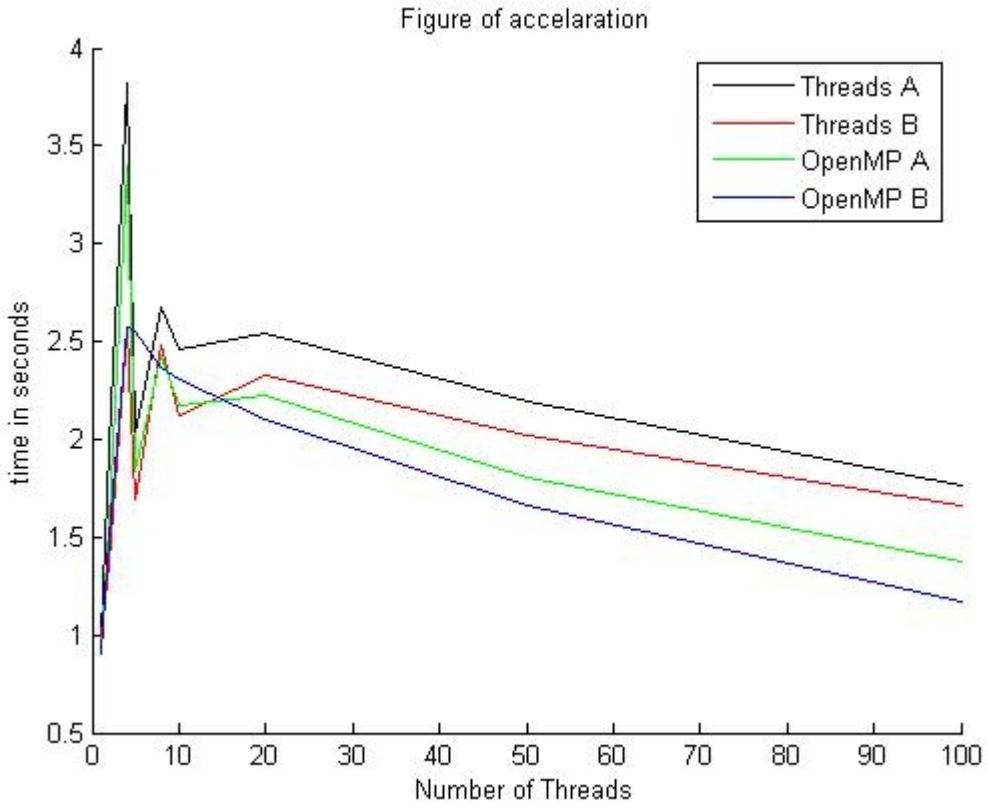
	accelaration			
No of Threads	Threads A	Threads B	OpenMP A	OpenMP B
1	0,99625	0,99634	0,90232	0,90324
2	1,9606	1,3617	1,5726	1,5019
4	3,8164	2,5708	3,4005	2,5748
5	2,0334	1,6914	1,8333	2,5474
8	2,6775	2,4775	2,4307	2,3677
10	2,4576	2,1258	2,1758	2,3017
20	2,5433	2,3254	2,2263	2,1007
50	2,192	2,015	1,8077	1,66
100	1,763	1,6657	1,37	1,1667

Συγκεντρωτικές γραφικές αποτελεσμάτων

Γραφική παρουσίαση χρόνων εκτελέσεων



Γραφική παρουσίαση των επιταχύνσεων



Όπως φαίνεται και από τους παραπάνω πίνακες και διαγράμματα, καλύτεροι χρόνοι επιτυγχάνονται με παραλληλοποίηση μέσω 4 νημάτων. Αυτό διότι το σύστημα Ετεοκλής διαθέτει 2 διπύρηνους επεξεργαστές (δηλαδή στο σύνολο τους 4) και έτσι χρησιμοποιώντας 4 νήματα διαμοιράζουμε αποδοτικότερα τις εργασίες στους διαθέσιμους επεξεργαστές. Επίσης, για τον ίδιο λόγο, γενικότερα καλύτεροι χρόνοι επιτυγχάνονται με πλήθος νημάτων πολλαπλάσιο του 4. Ένα παράδειγμα αυτού φαίνεται ξεκάθαρα στους χρόνους μεταξύ 4, 5 και 8 νημάτων.
Παρατηρώντας τους χρόνους μεταξύ ThreadsA και ThreadsB, διαπιστώνουμε ότι τα πρώτα είναι πιο γρήγορα σε σχέση με τα δεύτερα, πράγμα που διαπιστώνεται εύκολα παρατηρώντας την υλοποίηση τους. Συγκεκριμένα, από την αρχή της δημιουργίας των νημάτων στα ThreadsA ανατίθεται συγκεκριμένο πεδίο εργασίας, εν αντιθέσει με τα ThreadsB που κάθε φορά πρέπει να βρίσκουμε με πιο περίπλοκο τρόπο την επόμενη γραμμή προς επεξεργασία από το εκάστοτε νήμα (βλέπε i=t_i+k*N_THREADS;). Επίσης, θα μπορούσαμε να υποθέσουμε ότι εκτελείται περισσότερες φορές ο βρόχος του if που ελέγχει να βρει το μεγαλύτερο diff_thread, παρ'όλα αυτά δεν μπορούμε να στηριχτούμε σε αυτό και λόγω του μηδαμινού χρόνου της ανάθεσης αλλά και λόγω του ότι αυτό εξαρτάται καθαρά από τα δεδομένα μας.
Παρατηρώντας τους χρόνους μεταξύ OpenMPA και OpenMPB, διαπιστώνουμε ότι υπάρχουν διαφοροποιήσεις μεταξύ τους, πράγμα που δεν μας αφήνει να καταλήξουμε ποιο από τα δύο είναι πιο γρήγορο. Τα OpenMPB παρατηρούμε πως είναι πιο γρήγορα για αριθμό νημάτων μη πολλαπλάσιο του 4 λόγω αποδοτικότερης διαμοίρασης εργασιών ανάμεσα στα νήματα (δυναμικά). Για

όλους τους άλλους αριθμούς νημάτων η στατική εκτέλεση αποδίδει καλύτερα. Τέλος, καταλήγουμε στο συμπέρασμα ότι καλύτερη υλοποίηση είναι αυτή των ThreadsA. Να σημειώσουμε ότι τα OpenMP παράχτηκαν με μεγαλύτερη ευκολία λόγω της αυτοματοποιημένης διαδικασίας ανάθεσης εργασιών στα διάφορα νήματα.