Parallel Implementation of John Conway's Game of Life

Μπεχτσούδης Ανέστης - *mpechtsoud@ceid.upatras.gr*

Abstract:

Στην παρούσα εργασία πραγματοποιήθηκαν παράλληλες υλοποιήσεις σε μοντέλο κοινής μνήμης του κυτταρικού αυτομάτου Game Of Life (John Conway). Οι κώδικες για τις δύο παράλληλες εκδοχές γράφτηκαν σε γλώσσα C και έγινε χρήση των Posix Threads (Pthreads) και OpenMP APIs αντίστοιχα. Προκειμένου να γίνει αξιολόγηση της απόδοσης και της βελτίωσης των υλοποιήσεων, πραγματοποιήθηκαν μετρήσεις του χρόνου εκτέλεσης σε 2 συστήματα, ένα προσωπικό φορητό υπολογιστή και το σύστημα που μας παρέχετε από το HPCLab (Xanthos). Αρχικά στο πρώτο κομμάτι θα παρουσιάσουμε πως προσεγγίσαμε το θέμα του παραλληλισμού για το κυτταρικό αυτόματο και θα παρουσιάσουμε τις τεχνικές λεπτομέρειες για τις 2 υλοποιήσεις. Στο δεύτερο κομμάτι θα παρουσιάσουμε τις μετρήσεις που έγιναν για τα 2 συστήματα και θα σχολιάσουμε την απόδοση μέσα από αυτά και κάποια βοηθητικά γραφήματα.

Game Of Life Rules:

Σε μία δυσδιάστατη εκδοχή του κυτταρικού αυτομάτου έχουμε ένα πίνακα MxN, όπου κάθε θέση του πίνακα αποτελεί ένα κελί (cell). Κάθε κελί έχει 2 πιθανές καταστάσεις, "νεκρό" ή "ζωντανό". Κατά την φάση αρχικοποίησης σε κάθε κελί δίνεται μία αρχική κατάσταση και ξεκινάει η εκτέλεση. Σε κάθε εκτέλεση (γύρω) κάθε κελί ελέγχει την κατάσταση των 8 γειτόνων του και με βάση κάποιους κανόνες προσδιορίζει την κατάσταση στην οποία θα μεταβεί. Οι κανόνες είναι οι εξής:

- Κάθε ζωντανό κελί με λιγότερο από 2 ζωντανούς γείτονες πεθαίνει
- Κάθε ζωντανό κελί με περισσότερους από 3 ζωντανούς γείτονες πεθαίνει
- Κάθε ζωντανό κελί με 2 ή 3 ζωντανούς γείτονες, παραμένει ζωντανό
- Κάθε νεκρό κελί με ακριβώς 3 ζωντανούς γείτονες ζωντανεύει.

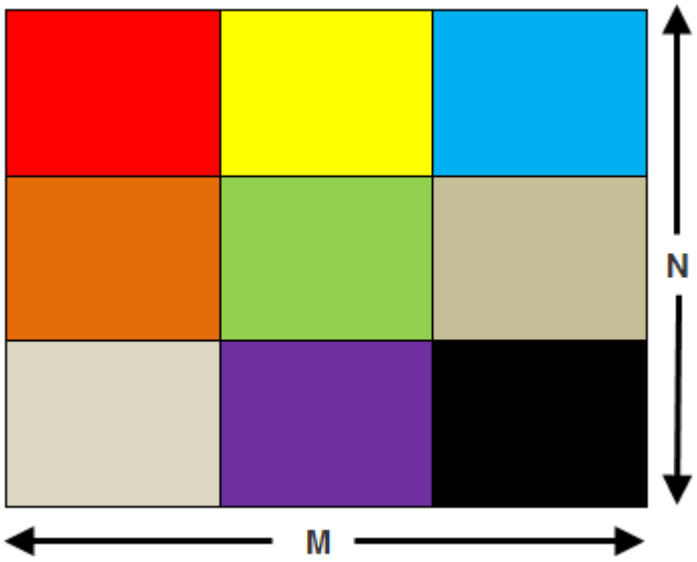
Μέρος 1ο – Λεπτομέρειες Υλοποίησης

1. Παράλληλη υλοποίηση του Game of Life

- *Χωρισμός Παράλληλων Περιοχών*

Το Game Of Life είναι ένα εγγενές πρόβλημα παραλληλισμού δεδομένων (data-parallel), όπου η επόμενη κατάσταση του στοιχείου εξαρτάτε από την τρέχουσα κατάσταση των γειτονικών στοιχείων. Έχουμε λοιπόν έναν πίνακα εξαρτώμενων στοιχείων και επεξεργασία που πρέπει να γίνει πάνω σε αυτά. Στο πολυνηματικό μοντέλο κοινής μνήμης που εξετάζουμε τα πράγματα είναι σχετικά απλά. Έχουμε στη διάθεση μας ένα αριθμό νημάτων τα οποία τρέχουν "ψευδό" παράλληλα, στα οποία μπορούμε να αναθέσουμε την επεξεργασία από ένα κομμάτι του πίνακα. Το πρόβλημα είναι πως θα χωρίσουμε βέλτιστα τον πίνακα.

Μία πρώτη λογική σκέψη θα ήταν απλά να διαιρέσουμε τον αρχικό πίνακα σε ισομερή τμήματα ή όσο το δυνατόν πιο κοντά στο ισομερές γίνεται εάν δεν πραγματοποιούνται ακριβώς οι διαιρέσεις με τις διαστάσεις. Κάπως έτσι δηλαδή:

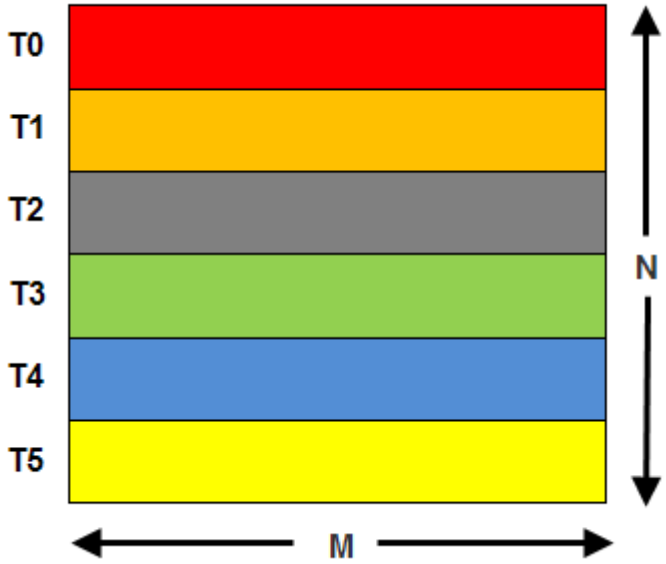


Ο παραπάνω διαχωρισμός εκ πρώτης όψεως φαίνεται αρκετά σωστός και βέλτιστος μιας και στοχεύει σε ισομερή κατανομή των μεγεθών στα νήματα επεξεργασίας, οπότε και θα έχουμε το καλύτερο δυνατό αποτέλεσμα. Μια προσεκτικότερη όμως και πιο τεχνική ματιά θα μας δείξει ότι πράγματα δεν είναι έτσι.

Γνωρίζουμε ότι σε όλα τα σύγχρονα υπολογιστικά συστήματα το κομμάτι της κύριας μνήμες έχει υλοποιηθεί με τεχνολογία SDRAM. Από την υλοποίηση των SDRAM ξέρουμε ότι είναι οργανωμένη σε Banks (συνήθως 4) όπου η προσπέλαση σε κάθε bank γίνεται δίνοντας διεύθυνση γραμμής και στήλης. Η επιλογή του bank φτάνει από ξεχωριστούς ακροδέκτες στο chip της μνήμης, ενώ για την επιλογή της γραμμής και στήλης γίνεται χρήση των ίδιων ακροδεκτών. Έχουμε δηλαδή μία

πολυπλεγμένη διεύθυνση όπου πρώτα αποστέλλεται η διεύθυνση της γραμμής και έπειτα η διεύθυνση της στήλης. Σε μία burst λειτουργία όπου θέλουμε να διαβάσουμε διαδοχικά πολλές θέσεις μνήμης, μας συμφέρει να τις έχουμε κατανέμει ανά στήλες διότι έτσι θα έχουμε το μικρότερο δυνατό overhead. Για περισσότερες λεπτομέρειες ανατρέξτε σε κάποιο σχετικό άρθρο για την υλοποίηση και προσπέλαση σε SDRAM μνήμες.

Εάν κάνουμε λοιπόν allocation του πίνακα ανά στήλες και χωρίσουμε τα τμήματα ανά γραμμές, ώστε να προσπελαύνονται ανά στήλες θα έχουμε την καλύτερη δυνατή απόδοση για το μοντέλο κοινής μνήμης των νημάτων. Ο διαχωρισμός λοιπόν γίνεται όπως την παρακάτω εικόνα:



Ο παραπάνω τρόπος χωρισμού παράλληλων τμημάτων ακολουθείται τόσο στην PThread υλοποίηση όσο και στην OpenMP. Ο καθορισμός των ορίων γίνεται ως εξής: Όπως γνωρίζουμε κάθε νήμα έχει ένα μοναδικό ακέραιο ID με την αρίθμηση να ξεκινάει από το 0. Οπότε το τμήμα που επεξεργάζεται το κάθε νήμα προκύπτει από τις εξής μαθηματικές σχέσεις:

*Start = ID * height / NThreads* - Αρχή τμήματος

Finish = Start + height / NThreads - Τέλος τμήματος

Όπου *NThreads* είναι ο αριθμός των νημάτων και height το ύψος (αριθμός γραμμών) του πίνακα.

Και τέλος επειδή δεν θέλουμε το περίγραμμα του πίνακα (κελιά περιφέρειας) να συμμετέχουν στο παιχνίδι (η κατάσταση τους παραμένει ίδια με την αρχική για όλη τη διάρκεια εκτέλεσης), κάνουμε τους ανάλογους ελέγχους μέσα στο κώδικα μας (τα σημεία είναι εμφανή από το σχολιασμό του πηγαίου αρχείου).

- *Συγχρονισμός*

Τα νήματα γίνονται schedule από το λειτουργικό με βάση το πρωτόκολλο που υλοποιεί και δεν είμαστε σε θέση εμείς να γνωρίζουμε τη σειρά με την οποία θα εκτελεστούν τα νήματα. Για αυτό το λόγο πρέπει να εξασφαλίσουμε ότι υπάρχει συγχρονισμός των νημάτων, εφόσον τα μερικά από τα δεδομένα που διαχειρίζεται το κάθε νήμα, πιο συγκεκριμένα τα στοιχεία που γειτονεύουν με τα στοιχεία του επόμενου νήματος, και έχουμε συνεπώς εξάρτηση των δεδομένων. Για να επιτύχουμε αυτό το συγχρονισμό κάνουμε χρήση των συναρτήσεων barrier που περιέχονται στην βιβλιοθήκη <*pthread.h*>.

Τα PThread barriers δουλεύουν με την εξής λογική: όταν ένα νήμα συναντήσει μία συνάρτηση pthread_barrier_wait(), αναστέλλει την εκτέλεση του, μέχρι να ικανοποιηθεί το φράγμα του αριθμού των νημάτων που έχουν επίσης καλέσει την pthread_barrier_wait(). Το πιο θα είναι αυτό το όριο το καθορίζουμε κατά την αρχικοποίηση του barrier.

Εύλογα μπορεί να δει λοιπόν κάποιος ότι βάζουμε ένα barrier στο τέλος κάθε γύρου εκτέλεσης, ώστε να εξασφαλίσουμε ότι κανένα νήμα δεν θα μεταβεί στον επόμενο γύρο, εάν πρώτα όλα τα νήματα δεν έχουνε τελειώσει την εκτέλεση του τρέχον γύρου. Έτσι αποφεύγουμε ασυνέπεια στα δεδομένα μας και κατά συνέπεια έχουμε valid ελέγχους κατά την επεξεργασία.

2. Λεπτομέρειες Υλοποίησης

Η υλοποίηση μας είναι πλήρως δυναμική, τόσο όσον αφορά τις διαστάσεις του πίνακα όσο και τον αριθμό των νημάτων, ενώ ταυτόχρονα υπάρχουν κάποιες default τιμές (80x25 και 1 νήμα) εάν δεν οριστούν τιμές από το χρήστη. Εφόσον λοιπόν μέσα στη main φτάσουμε σε στάδιο όπου γνωρίζουμε τα μεγέθη του πίνακα, κάνουμε memory allocation (*malloc()*) 2 πίνακες των 2 διαστάσεων ο καθένας. Χρειαζόμαστε 2 πίνακες προκειμένου ο ένας να περιέχει την τρέχουσα κατάσταση, και στον άλλο να αποθηκεύουμε τη νέα κατάσταση που έχει προκύψει σε κάθε γύρω για το κάθε κελί. Οπότε στο τέλος του κάθε γύρου αρκεί να αντιγράψουμε τα περιεχόμενα του δεύτερου πίνακα (επόμενη κατάσταση) στον πρώτο (τρέχουσα κατάσταση).

Για να αποφύγουμε τόσα πολλά και περιττά read και write, έχουμε 2 δείκτες προς (current και next) οι οποία δείχνουν αρχικά στους 2 πίνακες όπως τους ορίσαμε, και στο τέλος κάθε γύρου τους κάνουμε swap προκειμένου να έχουν τις νέες τιμές. Έτσι αποφεύγουμε το μεγάλο overhead τις προσπέλασης και αντιγραφής τόσων στοιχείων.

- **Operation Modes (Play – Bench) (Default – Non-Default)**

Όσον αφορά την εκτέλεση του Game Of Life, έχουμε ορίσει 2 mode για την συνολική εκτέλεση και 2 mode για το είδος των διαστάσεων του πίνακα. Για την συνολική εκτέλεση έχουμε το play και bench mode. Κατά το play mode, εμφανίζεται στο terminal η τρέχουσα κατάσταση του παιχνιδιού και κάποιες συνοδευτικές πληροφορίες. Για να έχει νόημα μία οπτική απεικόνιση της εκτέλεσης, πρέπει τα όρια του πίνακα να είναι εντός κάποιων λογικών ορίων. Εμείς θέσαμε αυτά τα όρια σε

100x50. Οπότε για μεγαλύτερα μεγέθη πίνακα δεν επιτρέπεται το play mode από το χρήστη. Στο bench mode έχουμε μια λειτουργία μέτρησης του χρόνου εκτέλεσης με χρήση της *gettimeofday()* συνάρτησης. Στο mode αυτό δεν γίνεται εκτύπωση της κατάστασης του πίνακα, αλλά παρουσιάζονται στατιστικά για την εκτέλεση που έγινε (διαστάσεις, αριθμός νημάτων, χρόνος εκτέλεσης σε us).

Όσον αφορά τώρα τις διαστάσεις του πίνακα έχουμε το default και το non-default mode. Κατά το default mode οι διαστάσεις είναι 80x25 και η αρχική κατάσταση δίνεται από αρχείο εισόδου με ίδια μεγέθη. Εμείς παραθέσαμε 2 ενδεικτικά αρχεία εισόδου Shapes, Random όπου το πρώτο περιλαμβάνει κάποια επαναλαμβανόμενα pattern και το δεύτερο είναι μία τυχαία αρχική κατάσταση. Στον non-default mode ο χρήστης μπορεί να ορίσει αυτός τις διαστάσεις του πίνακα και η αρχική κατάσταση διαμορφώνεται τυχαία με χρήση της *rand()* συνάρτησης.

Για τον καθορισμό των modes υπάρχουν 2 μεταβλητές-flag *bench_flag* και *dflag* για τα modes που αναφέρθηκαν αντίστοιχα.

- **Κοινές Συναρτήσεις σε PThread κ΄ OpenMP**

Στη συνέχεια ας δούμε κάποιες βασικές συναρτήσεις που υλοποιήθηκαν και είναι κοινές και για τις 2 υλοποιήσεις, ενώ αργότερα θα δούμε που διαφέρουν οι 2 κώδικες.

Function	Arguments	Return	Operation
initialize_board()	int **current, int dflag	void	Αν το default flag είναι 1 ο πίνακας αρχ. με 0, διαφορετικά με τυχαία 0,1.
read_file()	int **current, char *filename	void	Ανάγνωση της αρχικής κατάστασης από αρχείο.
copy_region()	int **current, int **next	void	Αντιγραφή της περιφέρειας και στο δεύτερο πίνακα.
adjacent_to()	int **current, int i, int j	int	Επιστρέφει τον αριθμό των ζωντανών γειτόνων του κελιού του current με θέση(i,j).
play()	int **current, int **next, int start, finish	void	Εφαρμογή των κανόνων του παιχνιδιού, και ανανέωση των τιμών στον πίνακα επόμενη κατάστασης (pointer next).
print()	int **current	void	Εκτύπωση της κατάστασης του current πίνακα
arg_check	int argc, char *argv[]	int	Έλεγχος των παραμέτρων με τις οποίες καλέστηκε το εκτελέσιμο. Επιστρέφει 0 σε επιτυχή έλεγχο, αλλιώς ένα διφορετικό int.
print_help()	void	void	Εμφάνιση των δυνατών παραμέτρων καλέσματος και ερμηνεία αυτών.

2.1 PThreads Implementation

Μέσα στη main εφόσον γίνει γνωστός ο αριθμός των νημάτων (default=1 or user specified) αρχικοποιείται αρχικά το barrier με όριο αριθμού νημάτων όλα τα νήματα. Αυτό γίνεται γιατί θέλουμε όλα τα νήματα στα οποία θα ανατεθεί από ένα τμήμα του πίνακα να συμμετέχουν στο συγχρονισμό όπως αναφέρθηκε παραπάνω. Έπειτα γίνονται create τα νήματα με συνάρτηση εισόδου την entry_function().

Στην αρχή της συνάρτησης εισόδου γίνεται καθορισμός των ορίων του τμήματος που θα επεξεργαστεί το κάθε νήμα με βάση τον τρόπο που αναφέρθηκε παραπάνω. Στη συνέχεια έχουμε ένα κυρίως loop των 100 επαναλήψεων το οποίο αποτελεί τους 100 γύρους εκτέλεσης του Game Of Life.

Στην αρχή κάθε γύρου εκτελείται η κυρίως συνάρτηση play με ορίσματα τους δείκτες για τους 2 πίνακες (current & next) και τα όρια του κομματιού του πίνακα που στα οποία θα εφαρμοσθούν οι κανόνες του παιχνιδιού. Η play() λοιπόν εφαρμόζει τους κανόνες και ανανεώνει τις τιμές του πίνακα στον οποίο δείχνει ο δείκτης next. Μέσα στην play υπάρχει ο κατάλληλος έλεγχος ώστε τα περιφερειακά στοιχεία να μείνουν ανεπηρέαστα.

Μετά την εκτέλεση της play ακολουθεί ένα barrier_wait() προκειμένου όλα τα νήματα να έχουν εφαρμόσει τους κανόνες και να έχουν ανανεώσει τις τιμές στον πίνακα της επόμενης κατάστασης. Στη συνέχεια το νήμα με ID==0 αναλαμβάνει να κάνει swap τους pointer προκειμένου ο current στον επόμενο γύρο να δείχνει στον next του τρέχον γύρου, και συνεπώς οι ανανεωμένες τιμές του τρέχον γύρου να γίνουν οι τρέχον τιμές στον επόμενο γύρο. Επίσης το νήμα με ID==0 αναλαμβάνει να κάνει και κάποιες εκτυπώσεις στην κονσόλα εάν είμαστε σε play mode προκειμένου να παρουσιαστεί η κατάσταση του πίνακα κατά τον τρέχον γύρο. Μετά το swap των δεικτών ακολουθεί άλλο ένα barrier έτσι ώστε να εξασφαλίσουμε ότι κανένα νήμα δεν θα μεταβεί στον επόμενο γύρω εάν δεν έχει ολοκληρωθεί πρώτα η εναλλαγή των δεικτών.

Με το πέρας των 100 γύρων το νήμα τελειώνει την επεξεργασία του και επιστρέφει.

2.2 OpenMP Implementation

Στην OpenMP υλοποίηση οι διαφορές δεν είναι και τόσο μεγάλες. Μέσα στη main εφόσον τελειώσουμε με τους ελέγχους και τις αρχικοποιήσεις ορίζουμε μία παράλληλη περιοχή με την κατάλληλη openmp directive. Ορίζουμε σαν κοινές μεταβλητές τους δείκτες προς τους 2 πίνακες καθώς και των αριθμό των νημάτων. Σαν ιδιωτικές μεταβλητές ορίζουμε το ID του νήματος, τα όρια επεξεργασίας και μία βοηθητική μεταβλητή. Επίσης μέσα από το option num_threads() ορίζουμε με πόσα νήματα θέλουμε να εκτελεστεί η παράλληλη περιοχή.

Στη συνέχεια η φιλοσοφία είναι ίδια με αυτή των PThread. Κάθε νήμα ορίζει τα όρια του με βάση το ID του (όπως αναφέρθηκε και παραπάνω) και ξεκινάει την κυρίως λειτουργία για 100 γύρους. Σε κάθε γύρο μετά την εκτέλεση της play έχουμε ένα barrier (ανάλογη directive) για να έχουμε συγχρονισμό. Και εδώ το νήμα με ID==0 αναλαμβάνει να κάνει swap τους δείκτες και να εμφανίσει την κατάσταση του

πίνακα εάν είμαστε σε play mode. Στο τέλος του γύρου ακολουθεί ένα ακόμα barrier προκειμένου και εδώ να βεβαιωθούμε ότι έχει γίνει το swap πριν πάμε στον επόμενο γύρο.

Μετά το πέρας των 100 γύρων κλείνουμε την παράλληλη περιοχή και εμφανίζουμε τα ανάλογα μηνύματα προτού τερματίσει το πρόγραμμα.

Μέρος 2ο – Μετρήσεις

1. Συστήματα Εκτέλεσης

Όπως αναφέρθηκε και παραπάνω οι μετρήσεις πραγματοποιήθηκαν σε 2 σύστημα με διαφορετικές αρχιτεκτονικές προκειμένου να έχουμε μία πληρέστερη εικόνα για την συμπεριφορά των παράλληλων εκδοχών. Αρχικά ας δούμε τα βασικά χαρακτηριστικά των 2 συστημάτων.

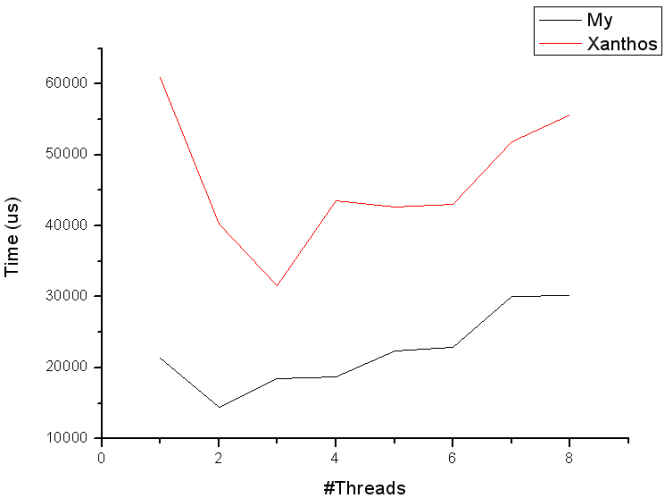
Characteristics	My System	Xanthos
Cpu Model	Inter Core 2 Duo T6400	AMD Opteron(tm) Processor 2210 HE
Frequency	2.00 GHz	1.80 GHz
#Cores	2	4
L2 Cache/Core	2048 KB	1024 KB
#Threads	2	4
Main Memory	3.00 GB	20 GB
Swap	3473 MB	20 GB

2. Αποτελέσματα - Σχολιασμός

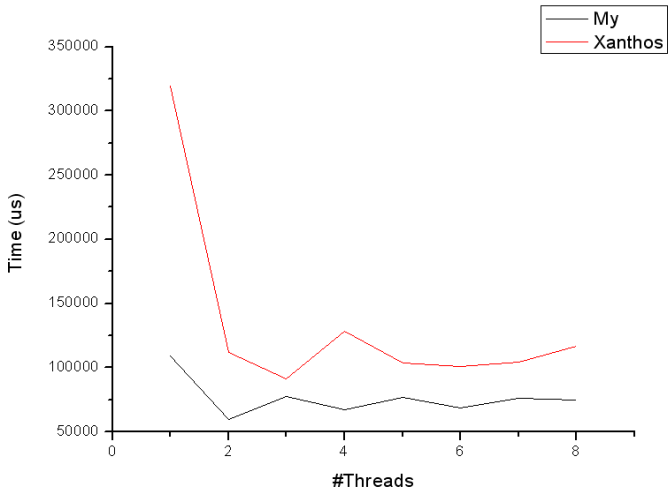
Αναλυτικοί πίνακες υπάρχουν στο συνοδευτικό φύλο excel. Εμείς εδώ θα παραθέσουμε όποια κομμάτια είναι αναγκαία για να κάνουμε το σχολιασμό που θέλουμε.

Αρχικά ας δούμε κάποιες μετρήσεις για την υλοποίηση με PThreads για διάφορα μεγέθη πίνακα:

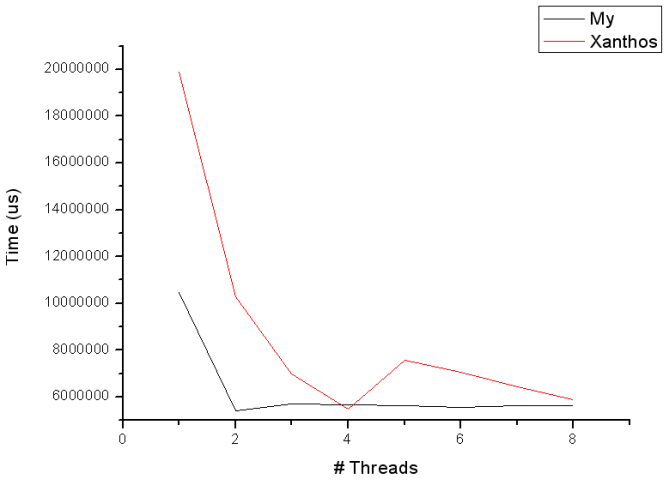
PThreads 80x25 – Random Input			
Nthreads	My System	Xanthos	
	Time (us)	Time (us)	Diff(M
1	21348	60926	
2	14379	40243	
3	18402	31597	
4	18757	43582	
5	22359	42599	
6	22856	42963	
7	30019	51860	
8	30170	55658	



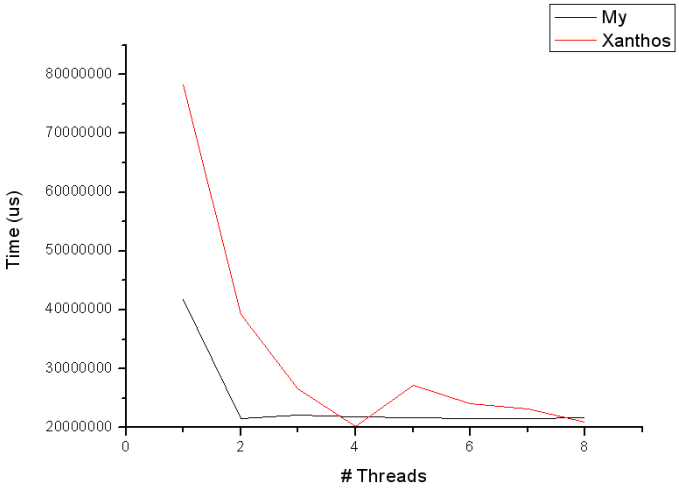
PThreads 200x50 – Random Input			
Nthreads	My System	Xanthos	
	Time (us)	Time (us)	Diff(My-Xanthos
1	109006	319621	-210615
2	59771	111583	-51812
3	77924	91070	-13146
4	67249	128418	-61169
5	77135	103729	-26594
6	69014	101019	-32005
7	76460	104758	-28298
8	74866	116670	-41804



PThreads 2000x500 – Random Input			
Nthreads	My System	Xanthos	
	Time (us)	Time (us)	Diff(My-Xanthos
1	10477616	19892058	-9414442
2	5397030	10298023	-4900993
3	5698842	6975674	-1276832
4	5670513	5481174	189339
5	5637657	7562911	-1925254
6	5545523	7050784	-1505261
7	5636890	6422875	-785985
8	5615205	5900633	-285428



PThreads 2000x2000 – Random Input			
Nthreads	My System	Xanthos	
	Time (us)	Time (us)	Diff(My-Xanthos
1	41790751	78177419	-36386668
2	21575731	39150984	-17575253
3	22081852	26587961	-4506109
4	21830961	20241626	1589335
5	21619917	27239080	-5619163
6	21454325	23992932	-2538607
7	21558108	23212022	-1653914
8	21597983	20926997	670986

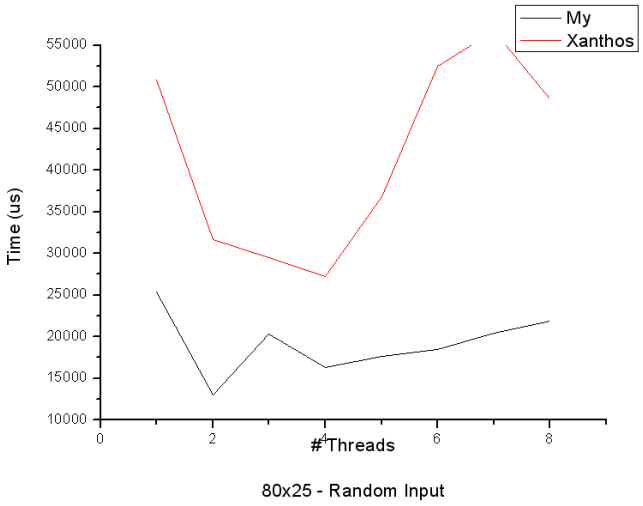


Παρατηρούμε ότι για μικρά μεγέθη πίνακα υπάρχει ένα όριο βελτίωσης όσο αυξάνουμε τον αριθμό των νημάτων. Από αυτό το όριο και μετά ο χρόνος εκτέλεσης

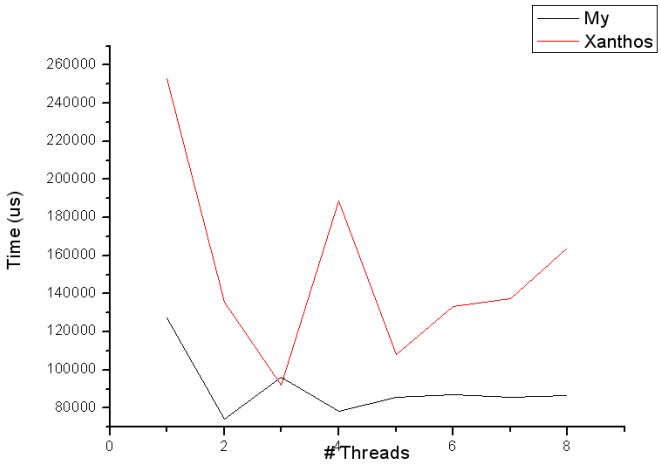
αντί να μειώνεται αυξάνεται. Αυτό είναι λογικό διότι το overhead για την εναλλαγή των νημάτων στα cores είναι τάξη μεγέθους μεγαλύτερο από τον ίδιο το χρόνο εκτέλεσης του κομματιού του κάθε νήματος. Επίσης για μικρά μεγέθη πίνακα βλέπουμε ότι η βελτίωση είναι ελάχιστη σε σχέση με το σειριακό. Αντίθετα για μεγάλα μεγέθη πίνακα η βελτίωση είναι αισθητά μεγαλύτερη όσο αυξάνεται ο αριθμός των νημάτων. Επίσης παρατηρούμε ότι έχουμε ένα άνω όριο στη βελτίωση που μπορούμε να επιτύχουμε. Από εκείνο το όριο και μετά όσο και να αυξάνουμε τα νήματα ο χρόνος εκτέλεσης παραμένει ίδιος ή και μεγαλύτερος όπως βλέπουμε για τα μικρά μεγέθη πίνακα.

Χαρακτηριστικό είναι ότι το βέλτιστο χρόνο στα δύο συστήματα τον έχουμε όταν ο αριθμός των νημάτων είναι ίδιος με αυτόν που μπορεί να δουλέψει ταυτόχρονα το εκάστοτε σύστημα (2 για το δικό μας και 4 για το Xanthos αντίστοιχα).

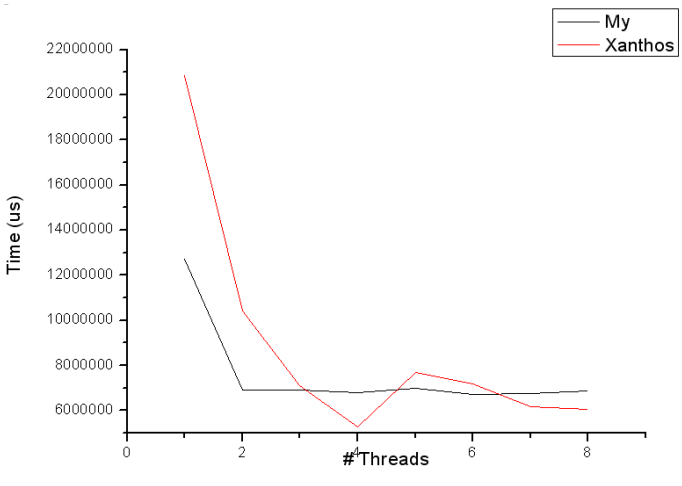
Τα παραπάνω ήταν για την PThread υλοποίηση, πάμε να δούμε τώρα τα αντίστοιχα γραφήματα για την OpenMP υλοποίηση. Τους πίνακες με τα νούμερα δεν τους συμπεριλάβαμε στην αναφορά, αλλά μπορούν να βρεθούν στο συνοδευτικό φύλλο excel.



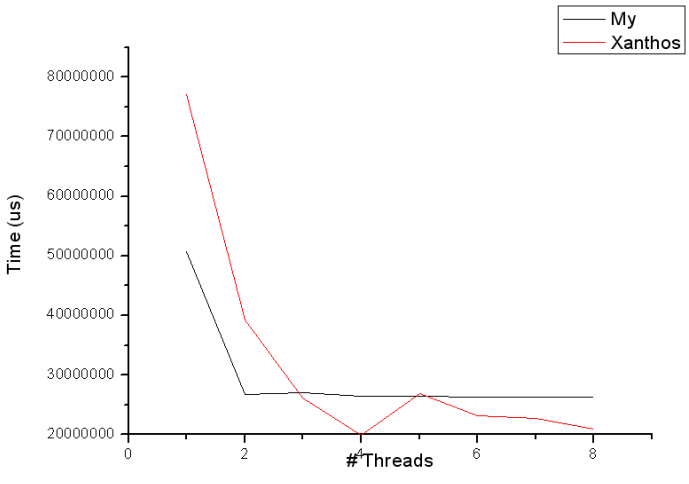
80x25 - Random Input



200x50 - Random Input



2000x500 - Random Input

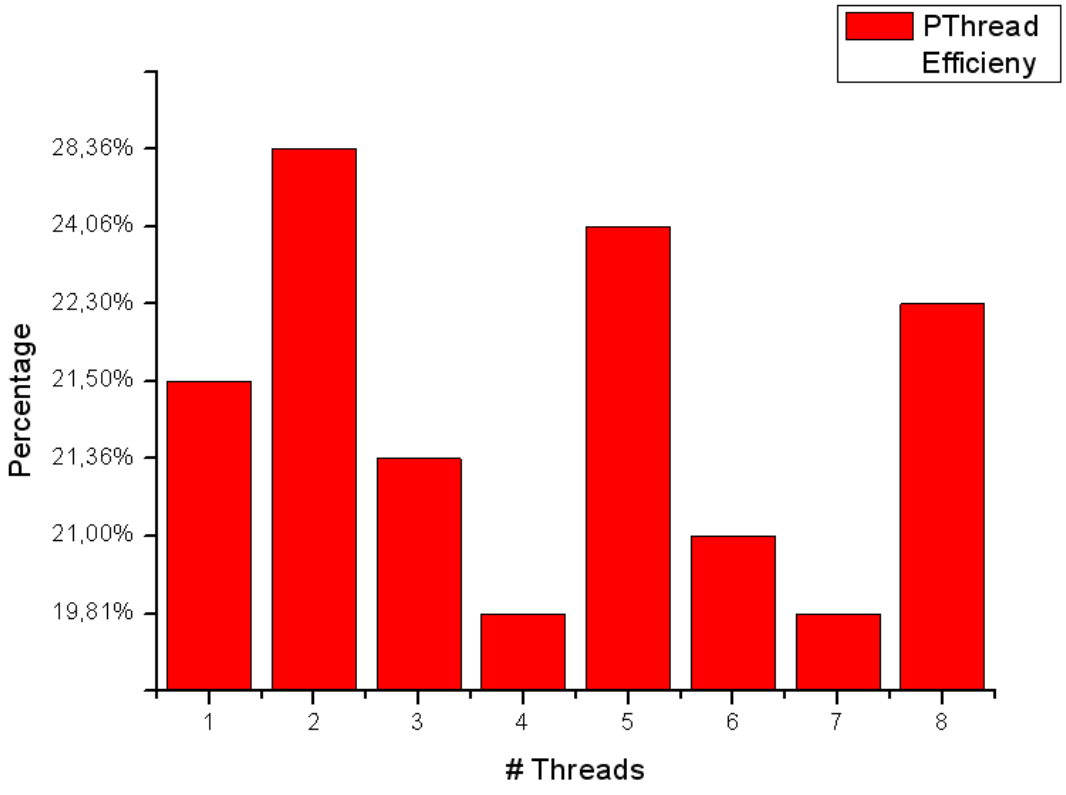


2000x2000 - Random Input

Ανάλογες παρατηρήσεις με τα PThread μπορούμε να κάνουμε και στο OpenMP. Βλέπουμε ότι για μικρά μεγέθη πίνακα υπάρχει ένα όριο βελτίωσης όσο αυξάνουμε τον αριθμό των νημάτων. Πέρα από αυτό το όριο εάν αυξήσουμε τον αριθμό των νημάτων ο χρόνος εκτέλεσης αυξάνεται αντί να μειώνεται για το λόγο που αναφέραμε και παραπάνω. Αντίθετα για μεγάλα μεγέθη πίνακα μετά από ένα όριο νημάτων και μετά ο χρόνος εκτέλεσης τείνει να γίνει σταθερός όσο και να αυξάνουμε τον αριθμό των νημάτων. Φυσικά και εδώ εάν το υπερβούμε ένα λογικό όριο αριθμού νημάτων, το overhead της εναλλαγής θα είναι και πάλι πολύ μεγάλο και ο συνολικός χρόνος εκτέλεσης θα αυξηθεί.

Επίσης παρατηρούμε ότι στην PThread υλοποίηση το δικό μας 2-Core σύστημα παρουσίαζε πάντα καλύτερη απόδοση ενώ για μεγάλα μεγέθη ο χρόνος εκτέλεσης των 2 συστημάτων συγκλίνει. Αντίθετα στην OpenMP υλοποίηση για μεγάλα μεγέθη είναι αισθητή η υπεροχή του συστήματος Xanthos, πράγμα που μας οδηγεί στο συμπέρασμα ότι γίνεται καλύτερη αξιοποίηση από την αρχιτεκτονική του Xanthos.

Στο επόμενο γράφημα βλέπουμε το efficiency της PThread υλοποίησης έναντι της OpenMP για το δικό μας σύστημα:



Βλέπουμε λοιπόν ότι η PThread υλοποίηση είναι καλύτερη από την OpenMP, πράγμα απολύτως δικαιολογημένο μιας και το OpenMP είναι ένα υψηλότερου επιπέδου αφαιρετικότητας API για υλοποίηση παράλληλων εφαρμογών, σε σχέση με τα PThtread τα οποία αποτελούν μια πιο low-level προσέγγιση. Φυσικά εδώ αξίζει να σημειωθεί η ανεξαρτησία πλατφόρμας που παρουσιάζει το OpenMP σε σχέση με τα Posix Threads.

Πάμε να δούμε τώρα μερικές μετρήσεις όσον αφορά τα μεγέθη του πίνακα.

Παίρνουμε 2 σταθερές τιμές για το ύψος 50 και 1000 και για αυτές παίρνουμε διάφορες μετρήσεις για διάφορες τιμές του πλάτους. Αρχικά παρατηρούμε ότι όσο αυξάνονται τα μεγέθη του πίνακα, ο χρόνος εκτέλεσης αυξάνεται γραμμικά.

Με μία προσεκτικότερη ματιά στα δεδομένα του γραφήματος βλέπουμε ότι για μεγάλο ύψος (γραμμές) έχουμε κατά πολύ μεγαλύτερη γωνία αύξησης σε σχέση με μικρές τιμές του ύψους. Αυτό έρχεται να επιβεβαιώσει τα λεγόμενα μας για μικρότερο overhead μεταφοράς δεδομένων από τη μνήμη όταν η προσπέλαση γίνεται κατά στήλες. Για αυτό θα ήταν πιο αποδοτικό εάν πάντα η μεγάλη διάσταση του πίνακα είναι το πλάτος (στήλες) και η μικρή το ύψος (γραμμές).

Παράρτημα:

Περιεχόμενα αρχεία:

- gol_threads.c – Source file Pthread implementation
- gol_openmp.c – Source file OpenMP implementation
- Makefile – Makefile to compile sources
- Random – Random input board’s state file
- Shapes – Pattern input board’s state file
- Measures.xls – Excel sheet with measurements
- Readme – Short txt file with executable parameters details
- GOLReport.pdf – Project report