Συστήματα Παράλληλης Επεξεργασίας

Ομάδα 6
Βασίλειος Βρεττός - el18126,
Ανδρέας Βατίστας - el18020,
Αλέξανδρος Τσάφος - el18211

Σχολή Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών, Εθνικό Μετσόβιο Πολυτεχνείο

1 Άσκηση 1 - Εξοικείωση με το περιβάλλον προγραμματισμού

Η συγκεκριμένη άσκηση είναι εισαγωγική με σκοπό την εξοικείωση μας με τα μηχανήματα του cslab στα οποία εκτελούμε τις εργασίες. Για να δοκιμάσουμε την κατανόησή μας, ζητήθηκε να παραλληλοποιήσουμε το Conway's Game of Life, ένα απλό παιχνίδι το οποίο "παίζεται" πάνω σε ένα ταμπλό (στην δική μας περίπτωση, διαστάσεων N x N). Το παιχνίδι τρέχει για K γύρους (χρονικά διαστήματα). Το κάθε σημείο του ταμπλό έχει 2 καταστάσεις, είτε ζωντανό ή νεκρό. Οι 2 κύριοι κανόνες του παιχνιδιού είναι:

1. Αν ένα ζωντανό σημείο έχει περισσότερους από 3 γείτονες ή λιγότερους από 2, τότε γίνεται νεκρό στον επόμενο γύρο. Αν έχει ακριβώς 2 ή 3, παραμένει ζωντανό.
2. Αν ένα νεκρό σημείο έχει ακριβώς 3 γείτονες, γίνεται ζωντανό στον επόμενο γύρο

Τρέχουμε το παραπάνω σενάριο για 1000 γενιές (γύρους) σε πίνακες διαστάσεων 64x64, 1024x1024 και 4096x4096.

Ο κώδικας ο οποίος ζητείται να παραλληλοποιηθεί είναι ο εξής.

```
for ( t = 0 ; t < T ; t++ ) {
       #pragma omp parallel for shared (previous, current) private (i,j,nbrs)
        for ( i = 1 ; i < N-1 ; i++ )
           for ( j = 1 ; j < N-1 ; j++ ) {
               nbrs = previous[i+1][j+1] + previous[i+1][j] + previous[i+1][j-1] \
                     + previous[i][j-1] + previous[i][j+1] \
                     + previous[i-1][j-1] + previous[i-1][j] + previous[i-1][j+1];
               if ( nbrs == 3 || ( previous[i][j]+nbrs ==3 ) )
                   current[i][j]=1;
               else
```

```
                current[i][j]=0;
        }

        #ifdef OUTPUT
        print_to_pgm(current, N, t+1);
        #endif
        //Swap current array with previous array
        swap=current;
        current=previous;
        previous=swap;
    }
```

Το πρώτο for-loop, το οποίο είναι η αλλαγή των γενιών, δεν έχει νόημα να παραλληλοποιηθεί αφού χρειαζόμαστε την τιμή της προηγούμενης γενιάς για να υπολογίσουμε τις καταστάσεις των κελιών για την επόμενη γενιά.

Η δική μας προσθήκη στον κώδικα είναι η γραμμή.

```
#pragma omp parallel for shared (previous, current) private (i,j,nbrs)
```

Το συγκεκριμένο compiler directive (pragma) είναι χαρακτηριστικό του OpenMP. Είναι μια "οδηγία" προς τον μεταγλωττιστή η οποία του ζητάει να παραλληλοποιηθούν τα εσωτερικά for-loops με νήματα ορισμένα από ένα environmental variable. Στο συγκεκριμένο directive, επίσης ζητάμε:

1. Οι δείκτες previous, current να μοιράζονται μεταξύ των νημάτων. Η διάσταση N μοιράζεται by default.
2. Οι μεταβλητές i,j,nbrs να είναι ξεχωριστές για κάθε νήμα.

Την απόδοση του πολυνηματισμού θα εξετάσουμε με χρήση την μετρική του χρόνου, και κατά συνέπεια, του speedup $S = \frac{T_s}{T_p}$.

```
GameOfLife: Size 64 Steps 1000 Time 0.023134 Threads: 1
GameOfLife: Size 64 Steps 1000 Time 0.013553 Threads: 2
GameOfLife: Size 64 Steps 1000 Time 0.010247 Threads: 4
GameOfLife: Size 64 Steps 1000 Time 0.009112 Threads: 6
GameOfLife: Size 64 Steps 1000 Time 0.009294 Threads: 8
GameOfLife: Size 1024 Steps 1000 Time 10.969249 Threads: 1
GameOfLife: Size 1024 Steps 1000 Time 5.452860 Threads: 2
GameOfLife: Size 1024 Steps 1000 Time 2.724507 Threads: 4
GameOfLife: Size 1024 Steps 1000 Time 1.828835 Threads: 6
GameOfLife: Size 1024 Steps 1000 Time 1.376873 Threads: 8
GameOfLife: Size 4096 Steps 1000 Time 175.911689 Threads: 1
GameOfLife: Size 4096 Steps 1000 Time 88.237651 Threads: 2
GameOfLife: Size 4096 Steps 1000 Time 44.518161 Threads: 4
GameOfLife: Size 4096 Steps 1000 Time 37.291715 Threads: 6
GameOfLife: Size 4096 Steps 1000 Time 36.185633 Threads: 8
```

Figure 1: Έξοδος Game of Life

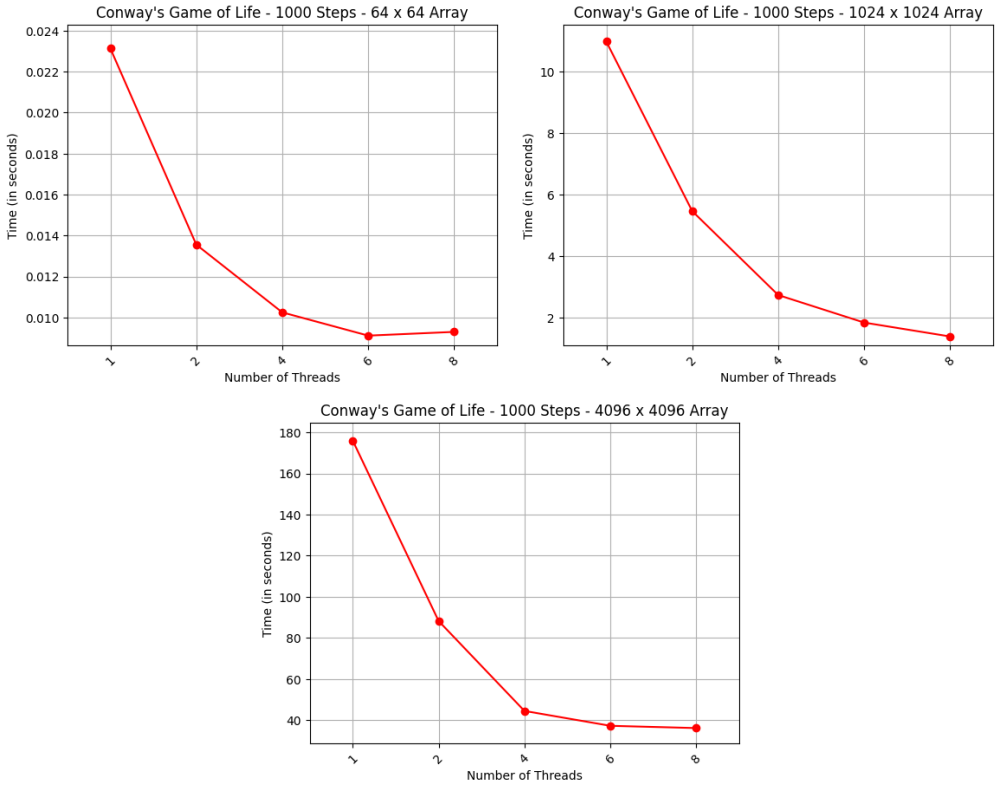


Figure 2: Χρόνος Εκτέλεσης Game of Life

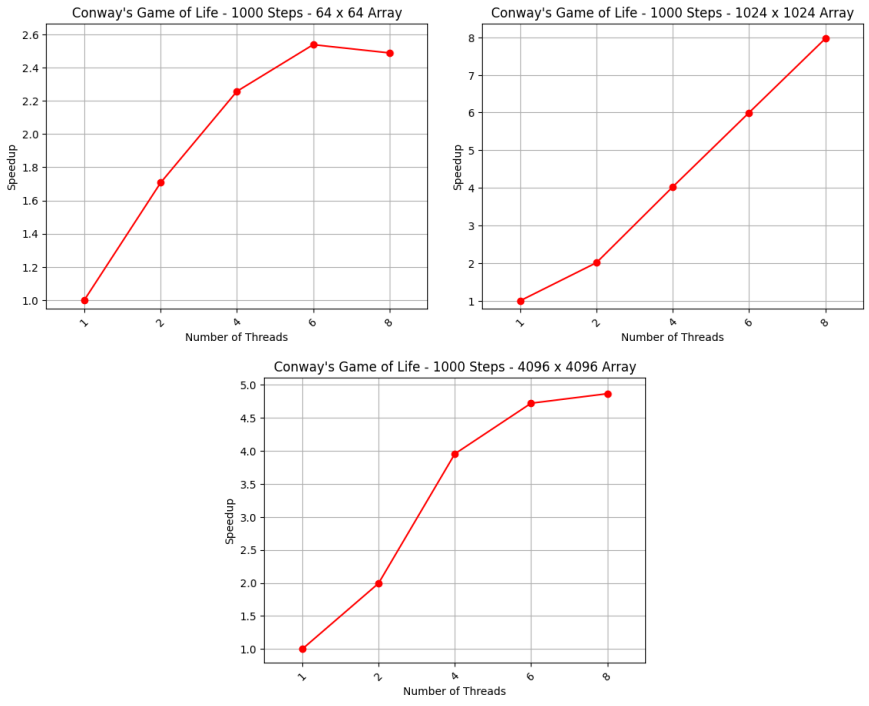


Figure 3: Speedup Game of Life

Συμπεράσματα

Για διαστάσεις 64x64

Παρατηρούμε ότι η μείωση του χρόνου δεν είναι ακριβώς ανάλογη των αριθμών των νημάτων. Από 1 σε 2 νήματα ή από 2 σε 4 νήματα βλέπουμε σημαντική βελτίωση της απόδοσης. Η εναλλαγή από 4 σε 6 νήματα προσφέρει πολύ μικρότερη αύξηση απόδοσης. Τέλος, από 6 σε 8 νήματα παρατηρούμε **ΑΥΞΗΣΗ** του χρόνου εκτέλεσης (μηδαμινή).

Η συγκεκριμένη αύξηση οφείλεται στο overhead που υπάρχει με την χρήση πολυνηματισμού σε μια διεργασία (γέννηση νημάτων, επικοινωνία κ.λ.π.). Ο συγκεκριμένος πίνακας είναι τόσο μικρός που η παραλληλοποίησή του περαιτέρω δεν βγάζει νόημα. Αν είχαμε την δυνατότητα να εξετάσουμε μεγαλύτερο αριθμό νημάτων, πολύ πιθανό να βλέπαμε περαιτέρω αύξηση του χρόνου εκτέλεσης

Για διαστάσεις 1024x1024

Παρατηρούμε ότι το speedup τείνει να είναι πλήρως γραμμικό. Ο χρόνος υποδιπλασιάζεται με κάθε διπλασιασμό νημάτων. Το συγκεκριμένο ταμπλό φαίνεται ιδανικό για παραλληλοποίηση. Η επικοινωνία μεταξύ νημάτων δεν περιορίζει την απόδοσή τους

Για διαστάσεις 4096x4096

Στους πρώτους 2 διπλασιασμούς των threads, βλέπουμε γραμμική μείωση του χρόνου (υποδιπλασιασμός) και κατά συνέπεια γραμμική αύξηση του speedup. Η λογική αυτή σταματάει να ισχύει για περαιτέρω αύξηση των νημάτων. Το συγκεκριμένο φαινόμενο δικαιολογείται από την ύπαρξη μεγάλου φόρτου (συμφόρησης) στον διάδρομο μνήμης. Υπάρχει συχνός διαμοιρασμός δεδομένων μεταξύ νημάτων καθώς αυξάνονται οι τιμές που πρέπει να ελεγχθούν για να υπολογιστούν οι γείτονες.

2 Άσκηση 2 - Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε αρχιτεκτονικές κοινής μνήμης

Στην συγκεκριμένη άσκηση, σκοπός είναι η παραλληλοποίηση 2 διαφορετικών εκδόσεων των αλγορίθμων K-means και Floyd-Warshall.

2.1 Αλγόριθμος K-means

Ο αλγόριθμος k-means διαχωρίζει N αντικείμενα σε k μη επικαλυπτόμενες ομάδες. Ο παρακάτω ψευδοκώδικας περιγράφει τον αλγόριθμο.

```
until convergence (or fixed loops)
    for each object
        find nearest cluster
    for each cluster
        calculate new cluster center coordinates.
```

Στο συγκεκριμένο πρόβλημα μελετάμε 2 διαφορετικές εκδόσεις. Στην 1η, ο πίνακας των συντεταγμένων μοιράζεται από τα νήματα (shared cluster). Στην 2η υλοποίηση, δίνουμε στο κάθε νήμα μια τοπική έκδοση του πίνακα. Έτσι, τα νήματα μπορούν να κάνουν πράξεις χωρίς να υπάρχουν προβλήματα διαμοιρασμού (cache invalidation e.t.c.). Στο τέλος, κάθε νήμα ενημερώνει το master thread με τα δικα του αποτελέσματα (reduction).

2.1.1 K-means - Shared Clusters

Το μειονέκτημα της συγκεκριμένης υλοποίησης είναι ότι εφόσον λειτουργούμε πάνω σε διαμοιραζόμενα δεδομένα, υπάρχει ανάγκη η ανανέωση των shared variables **newClusterSize** και **newClusters** να γίνεται ατομικά.

Τα κλειδώματα για την ανανέωση των μεταλητών έγινε με την χρήση του **atomic** pragma.

Το πρόγραμμα λήγει για συγκεκριμένα iterations ή μέχρι να συγκλίνει. Τα παρακάτω δεδομένα είναι για τον συνδυασμό {Size, Coords, Clusters, Loops} = {256, 16, 16, 10}.

```
#pragma omp parallel for default(shared) shared(newClusterSize, newClusters)
    private(i, j, index)
for (i=0; i<numObjs; i++){
/*
 *
 */
#pragma omp atomic
newClusterSize[index]++;
/*
 *
 */
 #pragma omp atomic
 newClusters[index*numCoords + j] += objects[i*numCoords + j];
}
```

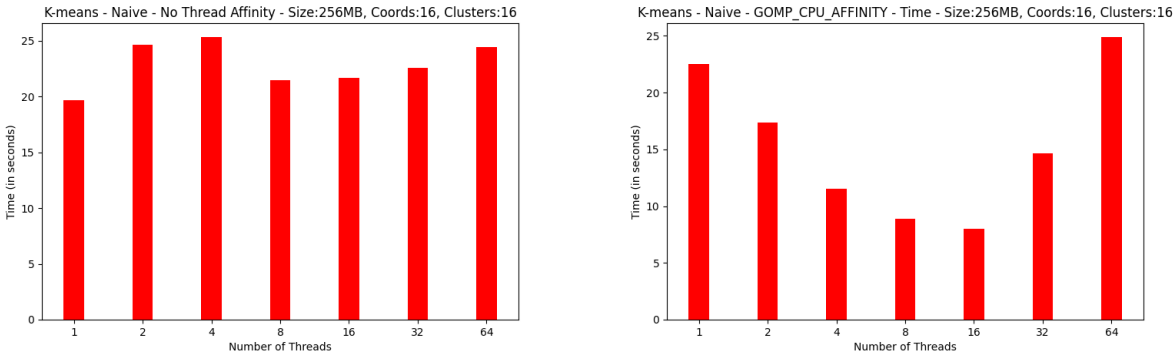


Figure 4: Χρόνος Εκτέλεσης K-Means Naive

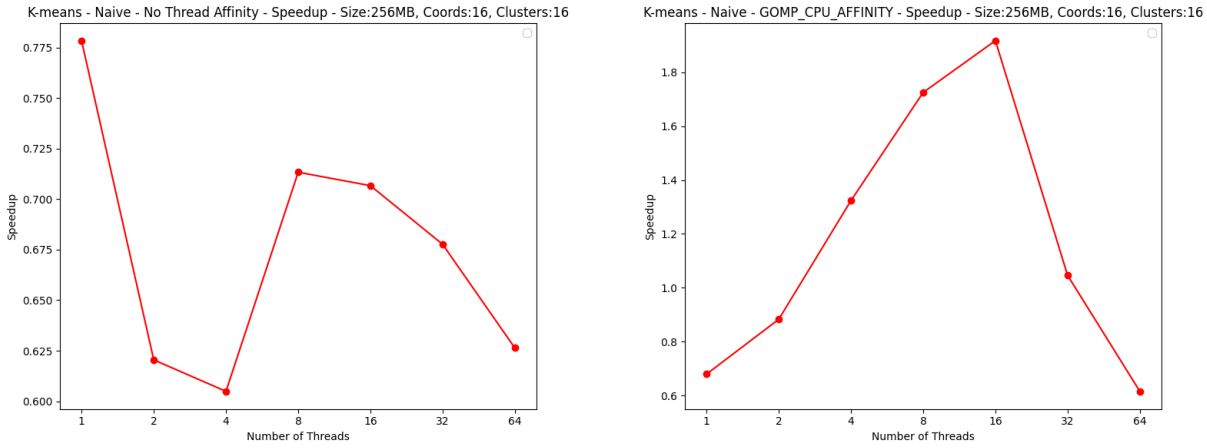


Figure 5: Speedup K-Means Naive

Σημείωση: Για το speedup θεωρούμε σειριακό χρόνο εκτέλεσης την έκδοση ΧΩΡΙΣ κλειδώματα. Εδώ το speedup με 1 νήμα είναι μικρότερο της μονάδας γιατί έχουμε κλειδώματα.

Το άμεσο συμπέρασμα είναι ότι το πρόγραμμα δεν βελτιώνεται με χρήση παραπάνω νημάτων. Συγκεκριμένα, η αύξηση των διαθέσιμων νημάτων αυξάνει τον χρόνο εκτέλεσης.

Σαν προσπάθεια να βελτιώσουμε τα αποτελέσματα, χρησιμοποιήσαμε την μεταβλητή περιβάλλοντος GOMP_CPU_AFFINITY. Η λειτουργία αυτής της μεταβλήτης είναι ότι προσδένει τα "λογικά" νήματα σε hardware νήματα των επεξεργαστών βάσει μιας ακολουθίας. Γνωρίζοντας ότι οι επεξεργαστές του Sandman θεωρούν ότι π.χ. στο Physical Core 0 υπάρχουν τα Logical Cores (0,32), επιλέξαμε τα νήματα να προσδένονται ακολουθιακά στους φυσικούς πυρήνες των επεξεργαστών.

Παρατηρήσαμε βελτίωση των χρόνων για τα πρώτα 32 νήματα. Φυσικό ήταν να μην βελτιωθεί η κατάσταση για τα 64 νήματα.

2.1.2 K-means - Copied Clusters and Reduce

2.2 Αλγόριθμος Floyd-Warshall

Έστω πίνακας "γειτνίασης" A. Ο κώδικας σε C για τον αλγόριθμο Floyd-Warshall είναι ο εξής:

```
for (k=0; k<N; k++)
    for (i=0; i<N; i++)
       for (j=0; j<N; j++)
            A[i][j] = min(A[i][j], A[i][k]+A[k][j]);
```

Το κομμάτι το οποίο μπορεί να παραλληλοποιηθεί είναι οι 2 εσωτερικά-φωλιασμένοι βρόγχοι.

2.2.1 Floyd-Warshall - Standard Edition

Η standard έκδοση του αλγορίθμου (σε C με OpenMP) είναι ως εξής:

```
for(k=0;k<N;k++)
    #pragma omp parallel for shared(A) private(i,j)
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            A[i][j]=min(A[i][j], A[i][k] + A[k][j]);
```

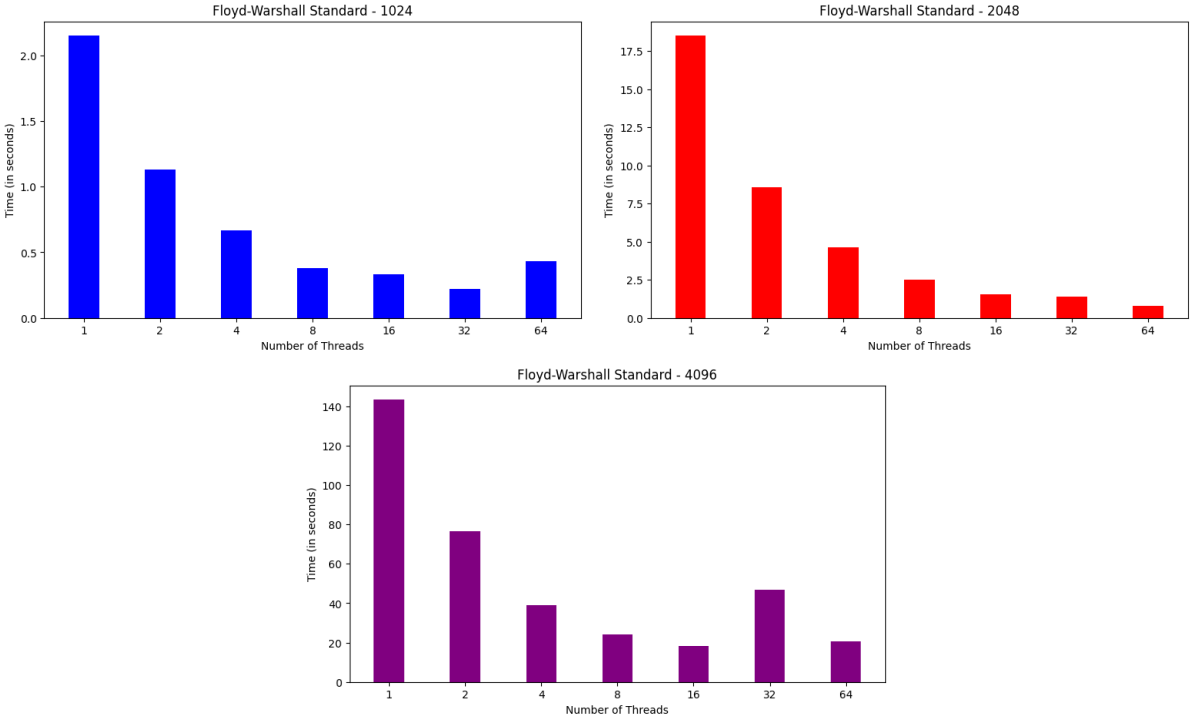


Figure 6: Χρόνος Εκτέλεσης FW Standard

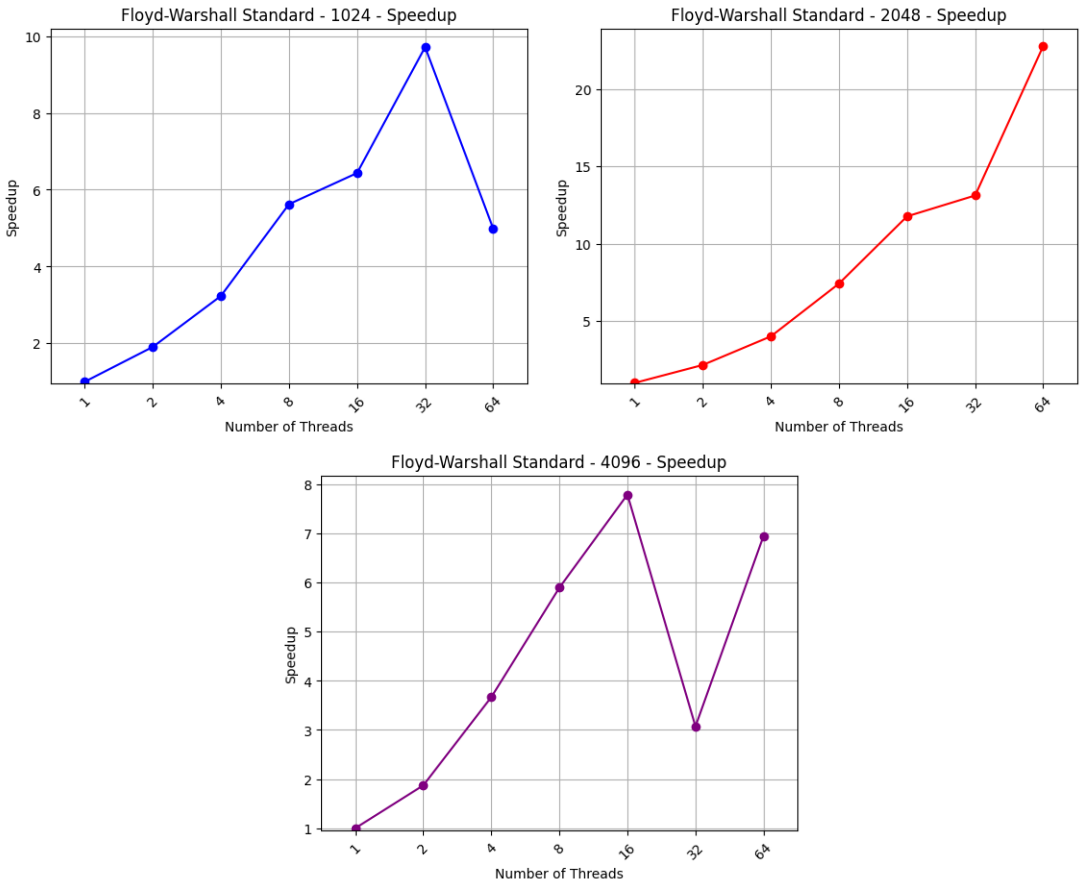


Figure 7: Speedup FW Standard

Παρατηρούμε ότι τα συμπεράσματα πέρι κλιμάκωσης είναι ολόιδια με το πρόβλημα του Game Of Life. Για μικρούς πίνακες, υπάρχει μεγάλη δυνατότητα για κλιμάκωση, αλλά αυξάνοτας τα νήματα, η το overhead που εισάγεται λόγω της επικοινωνίας μεταξύ τους περιορίζει την αύξηση της απόδοσης

Για μεσαίους πίνακες, οι οποίοι χωράνε στην cache των πυρήνων, έχουμε ακόμα μεγαλύτερη δυνατότητα για κλιμάκωση.

Για μεγάλους πίνακες, το πρόβλημα γίνεται πιο περίπλοκο καθώς πλέον δεν χωράει ο πίνακας στις cache μνήμες του επεξεργαστή. Αυξάνονται οι ανάγκες για μεταφορά cache lines. Άρα πλέον το πρόβλημα είναι memory bound.

2.2.2 Floyd-Warshall - Recursive/Task Based

Το νόημα της συγκεκριμένης έκδοσης του αλγορίθμου είναι να τρέχει αναδρομικά μέχρι να συναντήσει πίνακα μεγέθους, επιλεγμένο από εμας, βολικό για τον επεξεργαστή. Το μέγεθος αυτό ονομάζουμε block size. Δηλαδή, προσπαθούμε να χωρέσουμε ολόκληρο τον πίνακα στην cache. Παρακάτω παρουσιάζουμε το task graph της συνάρτησης αυτής για την πρώτη φορά που καλείται.

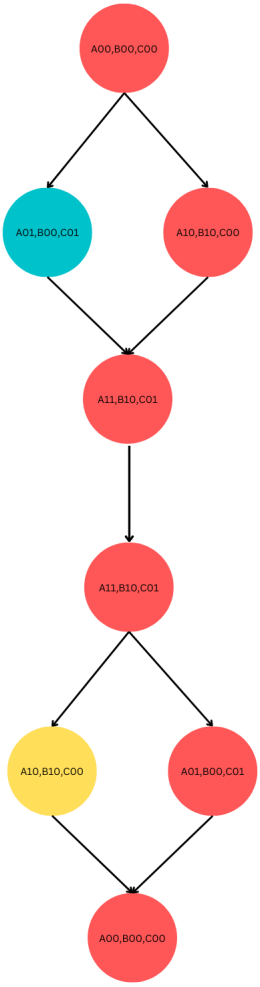
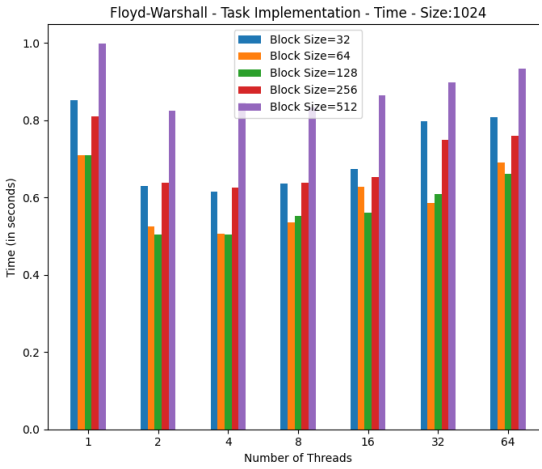


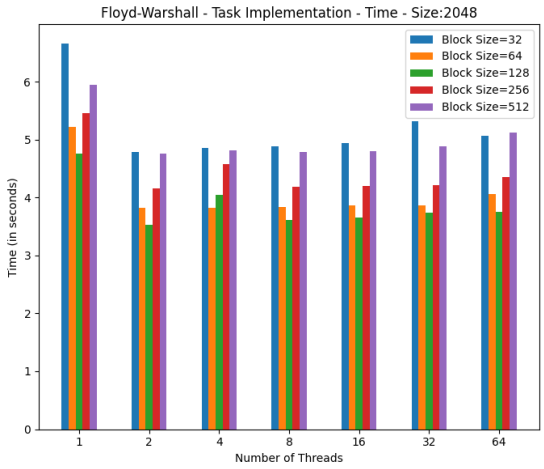
Figure 8: FW Recursive Task Graph - 1st iteration

Η παραλληλοποίηση αυτής της έκδοσης γίνεται με χρήση task, ώστε να μπορεί να είναι αναδρομική. Η OpenMP προσφέρει αυτή την δυνατότητα με την χρήση του

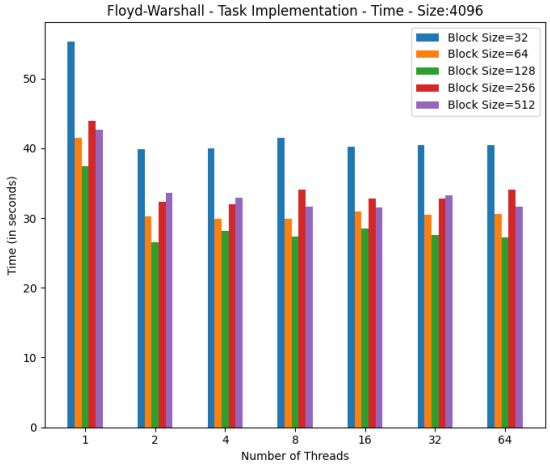
```
#pragma omp task
```



(a) FW Recursive Size 1024

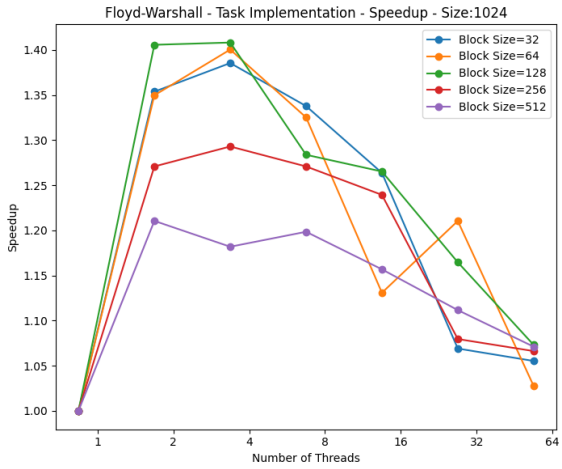


(b) FW Recursive Size 2048

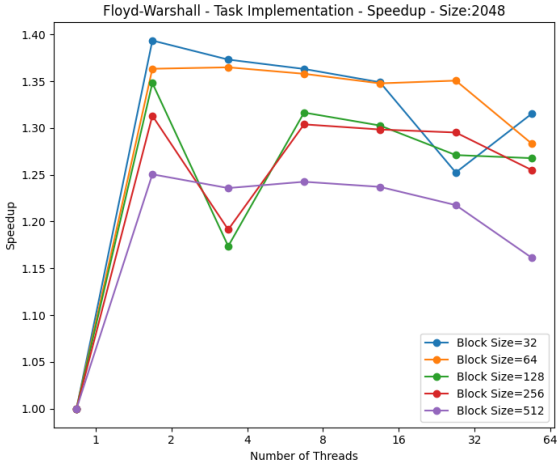


(c) FW Recursive Size 4096

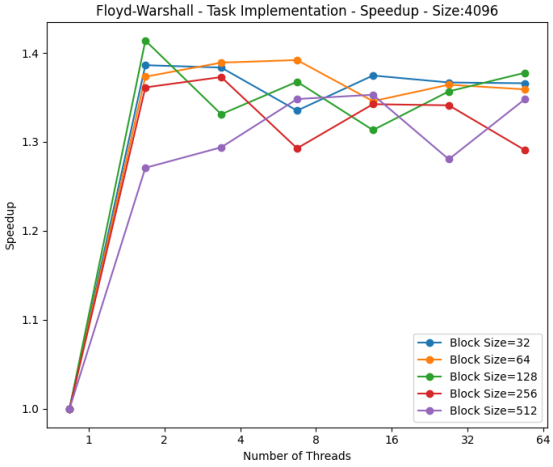
Figure 9: Χρόνος Εκτέλεσης FW Recursive



(a) FW Recursive Size 1024



(b) FW Recursive Size 2048



(c) FW Recursive - Size 4096 - Speedup

Figure 10: Speedup FW Recursive

Η πρώτη παρατήρηση είναι ότι το speedup δεν ξεπερνάει το 1.4. Σύμφωνα και με το task graph, η μέγιστη παραλληλοποίησή γίνεται με 2 νήματα σε κάθε επίπεδο αναδρομής. Ως προεπιλογή, το OpenMP **ΔΕΝ** επιτρέπει φωλιασμένη παραλληλοποίηση. Όταν ένα νήμα εισέρχεται σε παράλληλη περιοχή μέσα σε άλλη παράλληλη περιοχή, τότε δεν καλεί παραπάνω νήματα για την εκτέλεση αυτής της περιοχής. Άρα υπάρχουν περιοχές στο πρόγραμμα που εκτελούνται με το πολύ 1 εργάτη.

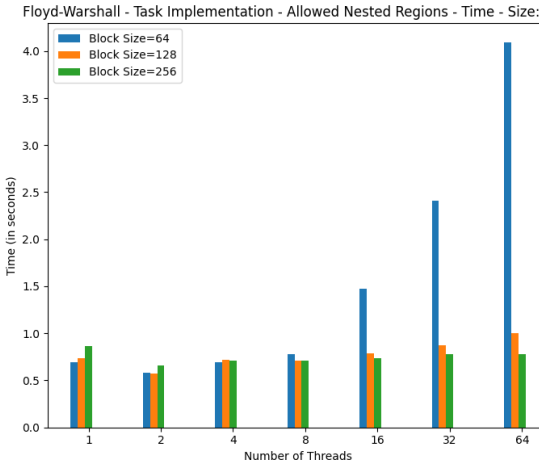
Η λύση για το παραπάνω είναι να αλλάξουμε την συμπεριφορά του OpenMP. Γίνεται να προσπεράσουμε αυτόν τον περιορισμό θέτοντας ένα environmental variable, το **OMP_NESTED** σε **TRUE**.

Η λύση αυτή δεν επαρκεί καθώς μετά μπορεί να επιφέρει αντίθετα αποτελέσματα από τα επιθυμητά. Επιτρέποντας την εκτέλεση φωλιασμένων παράλληλων περιοχών, υπάρχει ο κίνδυνος να καλέσουμε πολλά νήματα για την δημιουργία των task αλλά να μην υπάρχουν διαθέσιμα

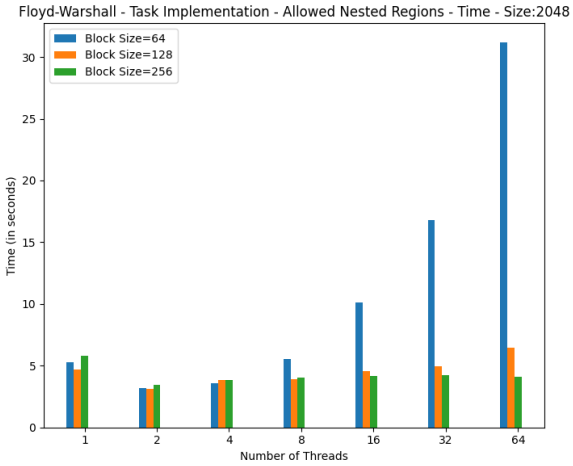
νήματα για την ίδια την εκτέλεση των πράξεων.

Χρειάζονται περαιτέρω περιορισμοί. Αυτοί δίνονται από το OpenMP με την μορφή μεταβλητών περιβάλλοντος.
Οι **SUNW_MP_MAX_POOL_THREADS** και **SUNW_MP_MAX_NESTED_LEVELS** περιορίζουν τον αριθμό των διαθέσιμων νημάτων/δούλων (δηλαδή όλων εκτός του νήματος αφέντη) και το μέγιστο βάθος που επιτρέπονται από τα νήματα να καλούν άλλα νήματα αντίστοιχα.

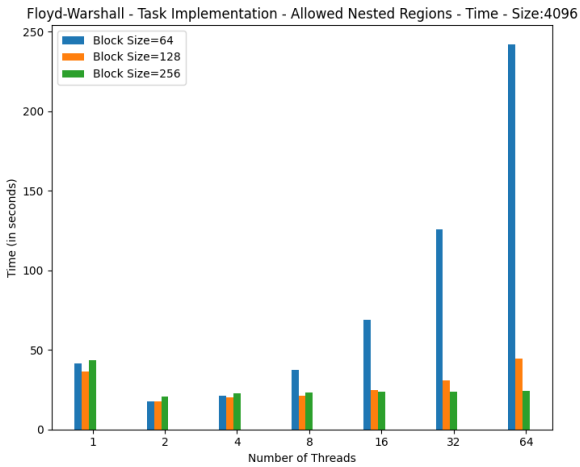
Εφαρμόζοντας τα παραπάνω, βγάζουμε τα παρακάτω αποτελέσματα:



(a) FW Recursive + Nested Size 1024

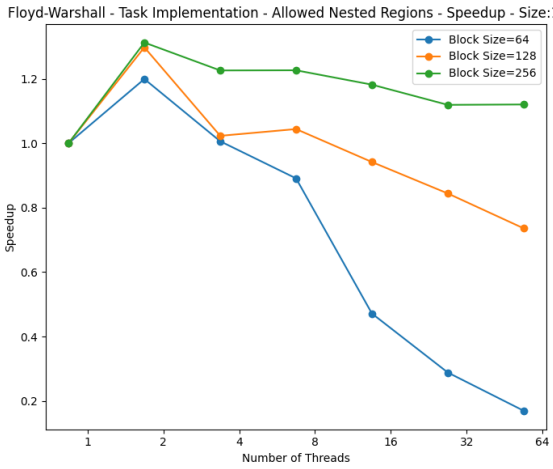


(b) FW Recursive + Nested Size 2048

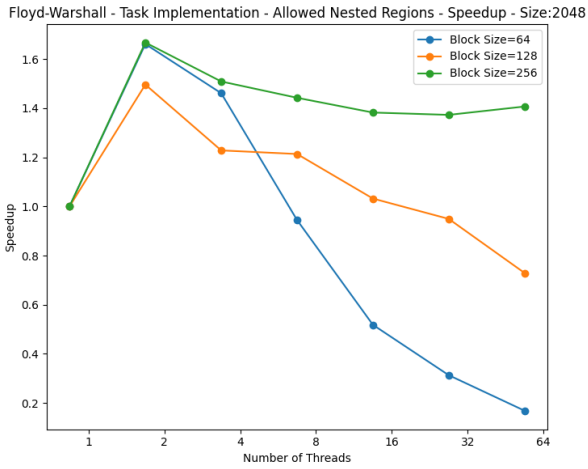


(c) FW Recursive + Nested Size 4096

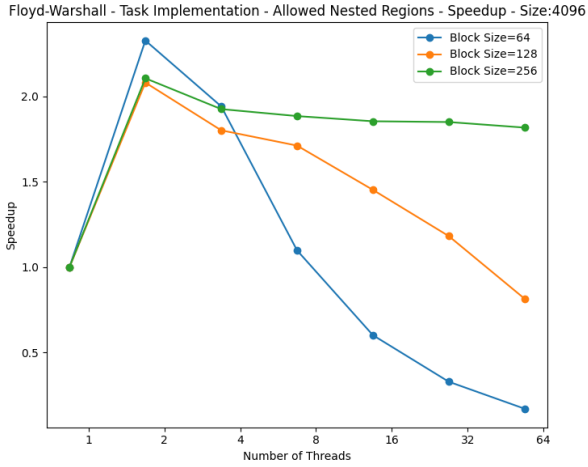
Figure 11: Χρόνος Εκτέλεσης FW Recursive/Nested



(a) FW Recursive - Size 1024 - Speedup



(b) FW Recursive - Size 2048 - Speedup

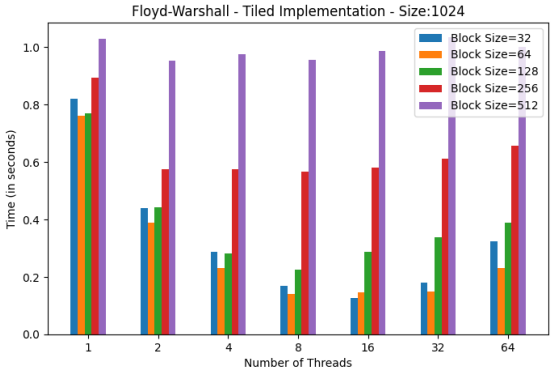


(c) FW Recursive - Size 4096 - Speedup

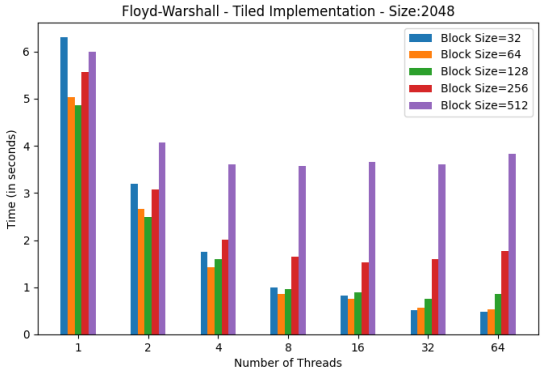
Figure 12: Speedup FW Recursive/Nested

2.2.3 Floyd-Warshall - Tiled

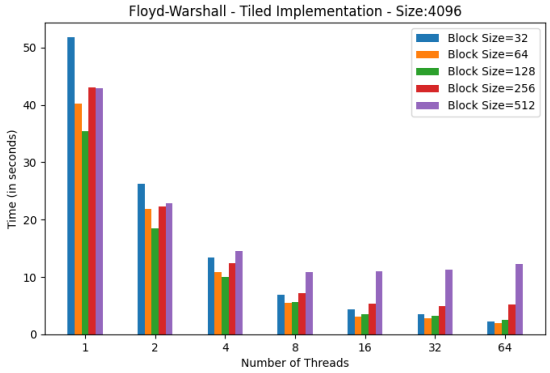
Στην συγκεκριμένη έκδοση,



(a) FW Tiled - Size 1024

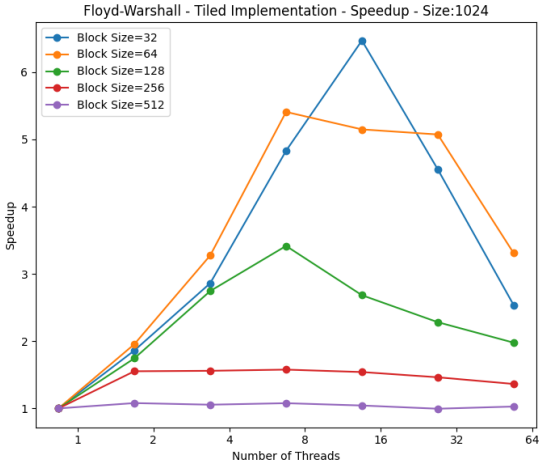


(b) FW Tiled - Size 2048

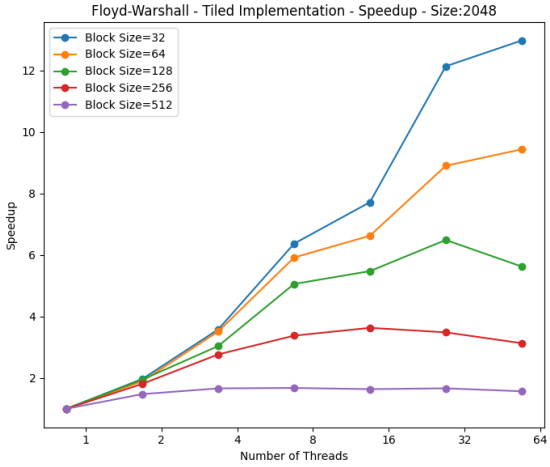


(c) FW Tiled - Size 4096

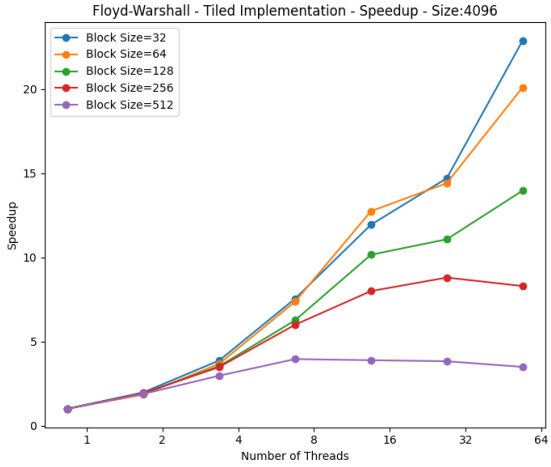
Figure 13: Χρόνος Εκτέλεσης FW Tiled



(a) FW Tiled - Size 1024 - Speedup



(b) FW Tiled - Size 2048 - Speedup



(c) FW Tiled - Size 4096 - Speedup

Figure 14: Speedup FW Tiled