

ΣΥΣΤΗΜΑΤΑ ΠΑΡΑΛΛΗΛΗΣ ΕΠΕΞΕΡΓΑΣΙΑΣ

1η ΑΣΚΗΣΗ

Παραλληλοποίηση αλγορίθμων σε πολυπύρηνες αρχιτεκτονικές κοινής μνήμης

Ακ. έτος 2020-2021, 9ο Εξάμηνο, Σχολή ΗΜ&ΜΥ

1.4 ΖΗΤΟΥΜΕΝΑ

1.4.1

Εξοικειωθείτε με τον τρόπο μεταγλώττισης και υποβολής εργασιών στις συστοιχίες. Εξηγούμε αναλυτικά τα βήματα μεταγλώττισης και υποβολής των εργασιών στο 1.4.3.

1.4.2

Αναπτύξτε παράλληλο πρόγραμμα στο μοντέλο κοινού χώρου διευθύνσεων (*shared address space*) με τη χρήση του *OpenMP*.

Το κομμάτι του κώδικα που έχει το μεγαλύτερο φορτίο φαίνεται παρακάτω πως το παραλληλοποιήσαμε:

```
for ( t = 0 ; t < T ; t++ ) {  
    #pragma omp parallel for schedule(static) private(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;  
        }  
}
```

Παρατηρήσαμε ότι τα δύο εσωτερικά loops είναι πλήρως ανεξάρτητα, άρα και παραλληλοποιήσιμα. Αυτό συμβαίνει, διότι οι τιμές στα κελιά *current*, εξαρτώνται μόνο από τα γειτονικά τους κελιά στο πίνακα *previous*, και δεν υπάρχει καμία εξάρτηση μεταξύ τους.

Επίσης, δεν υπάρχουν *race conditions*, διότι το κάθε *thread* θα γράφει σε ξεχωριστή γραμμή του κοινού πίνακα και έχουμε δηλώσει τις μεταβλητές *nbrs* και *j* ως *private* ώστε να μην επηρεάζονται οι τιμές τους μεταξύ των *threads*.

1.4.3

Πραγματοποιήστε μετρήσεις επίδοσης σε έναν από τους κόμβους της συστοιχίας των clones για 1,2,4,6,8 πυρήνες και μεγέθη ταμπλώ 64×64, 1024×1024 και 4096×4096 (σε όλες τις περιπτώσεις τρέξτε το παιχνίδι για 1000 γενιές).

Για την πραγματοποίηση των μετρήσεων, χρησιμοποιούμε για την μεταγλώττιση το script `make_on_queue.sh` το οποίο φαίνεται παρακάτω:

```
#!/bin/bash

## Give the Job a descriptive name
# PBS -N make_omp_Game_Of_Life

## Output and error files
# PBS -e make_omp_Game_Of_Life.err
# PBS -o make_omp_Game_Of_Life.out

## How many machines should we get?
# PBS -l nodes=1:ppn=1

##How long should the job run for?
# PBS -l walltime=00:10:00

## Start
## Run make in the src folder (modify properly)

module load openmp
cd /home/parallel/parlab07/a1/src/Game_Of_Life
make
```

Για την εκτέλεση των μετρήσεων για κάθε συνδυασμό πυρήνων και μεγεθών ταμπλώ, χρησιμοποιούμε το script `run_on_queue.sh`, το οποίο φαίνεται παρακάτω. Στην ουσία, ορίζουμε ένα πίνακα για τα threads και έναν για τα μεγέθη των ταμπλό. Έπειτα για κάθε size και κάθε thread, κάνουμε export τα threads που θέλουμε μέσω της environment variable `OMP_NUM_THREADS` και τρέχουμε το εκτελέσιμο πρόγραμμα.

```
# !/bin/bash

## Give the Job a descriptive name
# PBS -N run_omp_Game_Of_Life

## Output and error files
# PBS -o run_omp_Game_Of_Life.out
# PBS -e run_omp_Game_Of_Life.err

## How many machines should we get?
# PBS -L nodes=1:ppn=8

##How long should the job run for?
# PBS -L walltime=00:01:00

## Start
## Run make in the src folder (modify properly)

module load openmp
cd /home/parallel/parlab07/a1/src/Game_Of_Life

FILE=metrics.txt
nthreads=( 1 2 4 6 8 )
sizes=( 64 1024 4096)

if [ -f "$FILE" ]; then
    rm $FILE
fi
for size in "${sizes[@]}";
do
    for nthread in "${nthreads[@]}";
    do
        export OMP_NUM_THREADS=${nthread};
        echo -n "Threads ${nthread} " >> $FILE
        ./Game_Of_Life ${size} 1000 >> $FILE
    done
    echo >> $FILE
done
```

Τα αποτελέσματα αποθηκεύονται στο αρχείο `metrics.txt` και φαίνονται παρακάτω:

```
Threads 1 GameOfLife: Size 64 Steps 1000 Time 0.023125
Threads 2 GameOfLife: Size 64 Steps 1000 Time 0.013745
Threads 4 GameOfLife: Size 64 Steps 1000 Time 0.009466
Threads 6 GameOfLife: Size 64 Steps 1000 Time 0.009147
Threads 8 GameOfLife: Size 64 Steps 1000 Time 0.921568

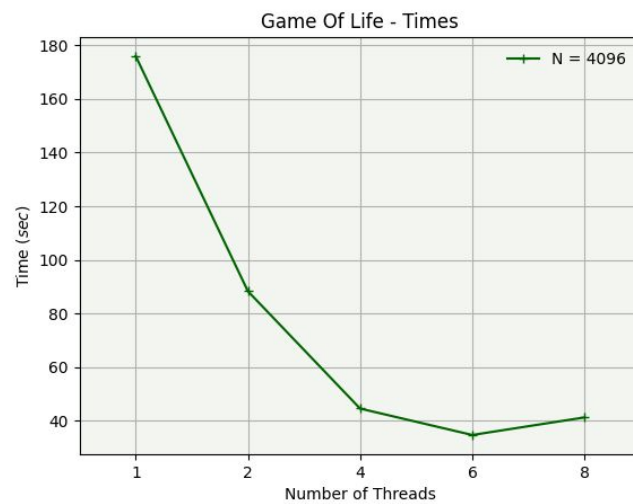
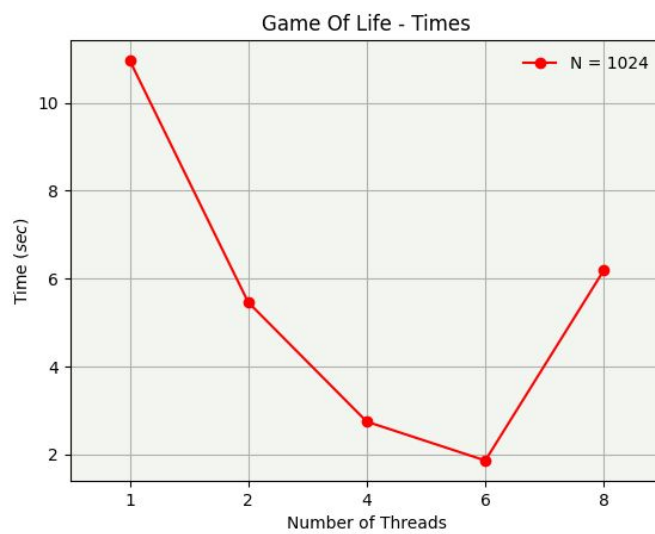
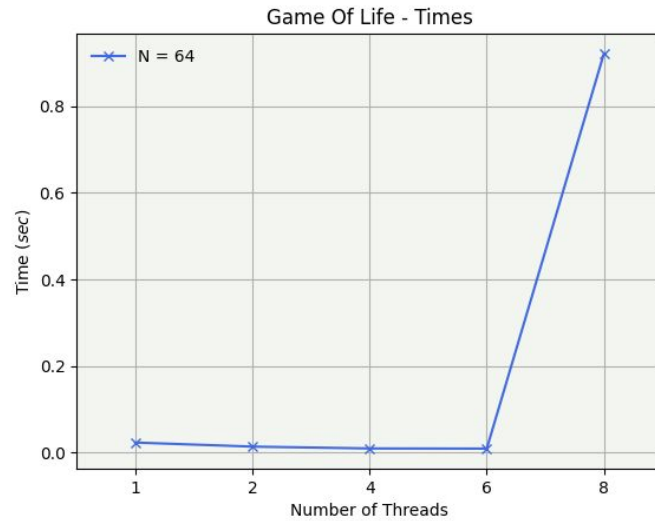
Threads 1 GameOfLife: Size 1024 Steps 1000 Time 10.965569
Threads 2 GameOfLife: Size 1024 Steps 1000 Time 5.458503
Threads 4 GameOfLife: Size 1024 Steps 1000 Time 2.741920
Threads 6 GameOfLife: Size 1024 Steps 1000 Time 1.856764
Threads 8 GameOfLife: Size 1024 Steps 1000 Time 6.187554

Threads 1 GameOfLife: Size 4096 Steps 1000 Time 175.871665
Threads 2 GameOfLife: Size 4096 Steps 1000 Time 88.288967
Threads 4 GameOfLife: Size 4096 Steps 1000 Time 44.582814
Threads 6 GameOfLife: Size 4096 Steps 1000 Time 34.739006
Threads 8 GameOfLife: Size 4096 Steps 1000 Time 41.281937
```

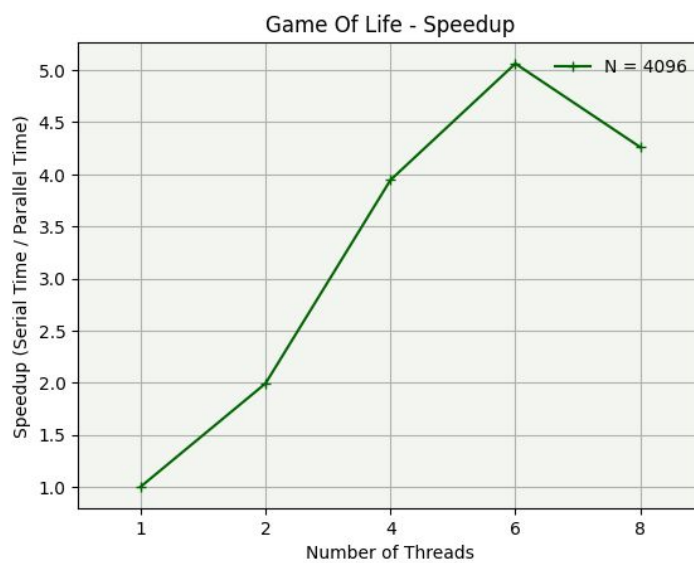
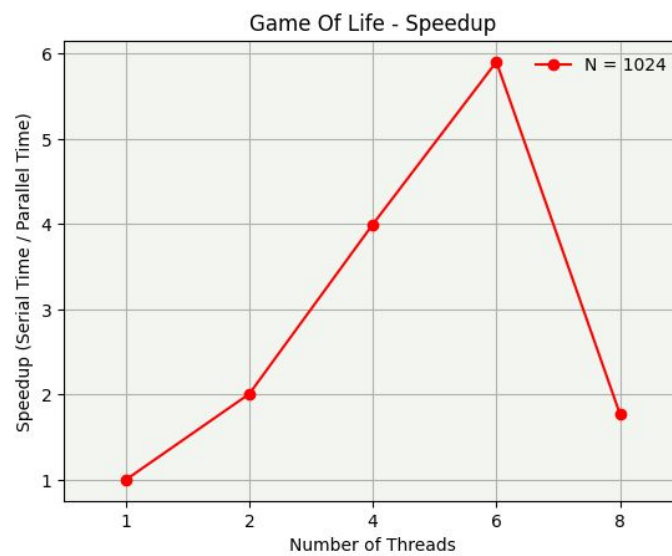
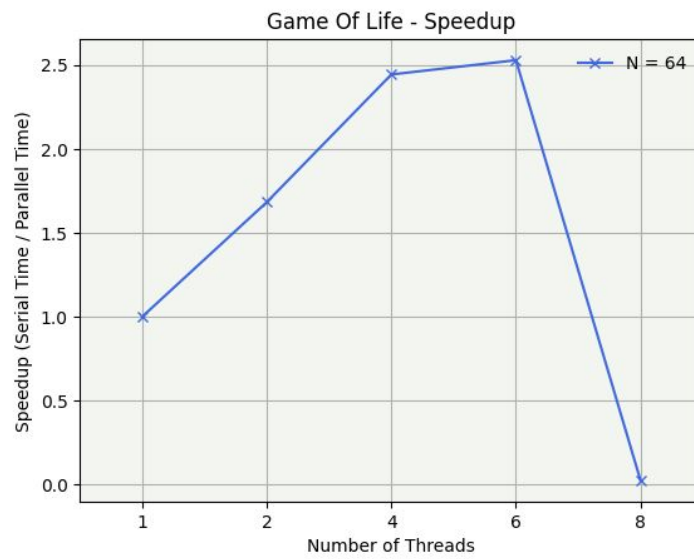
1.4.4

Συγκεντρώστε τα αποτελέσματά σας σε γραφήματα για το χρόνο και το *speedup*, καθώς και τα σχόλιά σας, στην ενδιάμεση αναφορά.

Διαγράμματα Χρόνου



Διαγράμματα Speedup



Συμπεράσματα

Παρατηρούμε ότι το πρόγραμμα μας κλιμακώνει σωστά μέχρι τους 6 πυρήνες, ενώ για τους 8 πυρήνες παρατηρούμε ότι έχουμε μια απότομη αύξηση στο χρόνο εκτέλεσης, για όλα τα μεγέθη ταμπλό.

Για αυξανόμενα μεγέθη ταμπλό, παρατηρούμε ότι η παραπάνω συμπεριφορά έχει σταδιακά μικρότερη επίδραση.

π.χ. για $N=64$, ο χρόνος εκτέλεσης στα 8 threads είναι κατά πολύ μεγαλύτερος και από τον χρόνο σε 1 thread, ενώ για $N=4096$ δεν έχουμε σημαντική διαφοροποίηση από ότι για 6 threads.

Μία πιθανή εξήγηση είναι ότι το κόστος του χρονοπρογραμματισμού των παραπάνω threads είναι μεγαλύτερο από το όφελος της περαιτέρω παραλληλοποίησης.

Τέλος, για την βέλτιστη εκτέλεση του αλγορίθμου θα επιλέγαμε 6 threads, καθώς δεν παρατηρούμε κάποια κλιμάκωση πέρα από αυτό το σημείο.

2.4 ΖΗΤΟΥΜΕΝΑ

2.4.1

Ανακαλύψτε τον παραλληλισμό του αλγορίθμου σε κάθε έκδοση και σχεδιάστε την παραλληλοποίησή του. Περιγράψτε τη στρατηγική παραλληλοποίησης στην ενδιάμεση αναφορά.

Κλασσικός Αλγόριθμος:

Για κάθε k :

- Τα στοιχεία $A[i][k]$, $A[k][j]$ (που αποτελούν την μπλε γραμμή στα σχήματα) δεν αλλάζουν τιμή, διότι:
 - $A[i][k] = \min(A[i][k], A[k][k] + A[i][k]) = A[i][k]$
 - $A[k][j] = \min(A[k][j], A[k][k] + A[k][j]) = A[k][j]$
- Άρα, μπορούμε να παραλληλοποιήσουμε πλήρως τα δύο εσωτερικά loop για το i, j .

Στον κώδικα με τα directives του OpenMP, θα είναι έτσι:

```
for(k=0; k<N; k++)  
    #pragma omp parallel for private(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]);
```


Recursive Αλγόριθμος:

Επειδή ο FW είναι memory bound και για μεγάλα N δεν χωράει ολόκληρος στην cache, υπάρχουν οι εναλλακτικές υλοποιήσεις του αλγορίθμου. Μία από αυτές είναι και ο αναδρομικός αλγόριθμος FW, ο οποίος αναλύεται παρακάτω:

Ο Αναδρομικός αλγόριθμος δέχεται ως **είσοδο**:

- 3 πίνακες (A, B, C) ίδιου μεγέθους.
 - Αρχικά μέγεθος = N
- Την γραμμή και την στήλη του πρώτου στοιχείου, για κάθε ένα από τους τρεις πίνακες (arow, acol, brow, bcol, crow, ccol).
 - Αρχικά όλα row = 0, col = 0
- Το μέγεθος του υποπίνακα προς επεξεργασία (myN).
 - Αρχικά μέγεθος = N
- Το μέγεθος του υποπίνακα για το οποίο θα σταματήσει η αναδρομή (bsize), το base case της αναδρομής δηλαδή.

Σε κάθε κλήση της αναδρομικής συνάρτησης γίνεται έλεγχος για το ποιά περίπτωση θα εκτελεστεί.

- Αν το μέγεθος του υποπίνακα που δίνεται, είναι μικρότερο ή ίσο του base case, τότε:
- **if (myN <= bsize):**
 - Για κάθε χρονικό βήμα k, υπολογίζεται για κάθε ζεύγος κόμβων i-j, αν υπάρχει συντομότερο μονοπάτι από τον i προς τον j, περνώντας από τον k.
 - Όλοι αυτοί οι υπολογισμοί γίνεται σχετικά με τις θέσεις (σειρά, στήλη) των πρώτων στοιχείων των υποπινάκων που έχουν δοθεί σε αυτήν την αναδρομική κλήση.
- Αλλιώς αν το μέγεθος του υποπίνακα που δίνεται, είναι μεγαλύτερο του base case, τότε:
- **if (myN > bsize):**
 - Γίνονται 8 αναδρομικές κλήσεις, όπου υποδιπλασιάζεται το μέγεθος του πίνακα (myN / 2), και στις αλλάζουν οι θέσεις των πρώτων στοιχείων των τριών πινάκων σύμφωνα με την σειρά που υποδεικνύει ο αλγόριθμος.
 - Στις πρώτες 4 κλήσεις, στέλνονται οι υποπίνακες που αντιστοιχούν σε:
 - Κλήσεις A: $A_{00}, A_{01}, A_{10}, A_{11}$ υποπίνακα
 - Κλήσεις B: $B_{00}, B_{00}, B_{10}, B_{10}$ υποπίνακα
 - Κλήσεις C: $C_{00}, C_{01}, C_{00}, C_{01}$ υποπίνακα
 - Στις τελευταίες 4 κλήσεις, γίνονται οι ίδιες κλήσεις με αντίστροφη σειρά.
 - Κλήσεις A: $A_{11}, A_{10}, A_{01}, A_{00}$ υποπίνακα
 - Κλήσεις B: $B_{10}, B_{10}, B_{00}, B_{00}$ υποπίνακα
 - Κλήσεις C: $C_{01}, C_{00}, C_{01}, C_{00}$ υποπίνακα
 - Κλήσεις:

Οι εξαρτήσεις που υπάρχουν είναι:

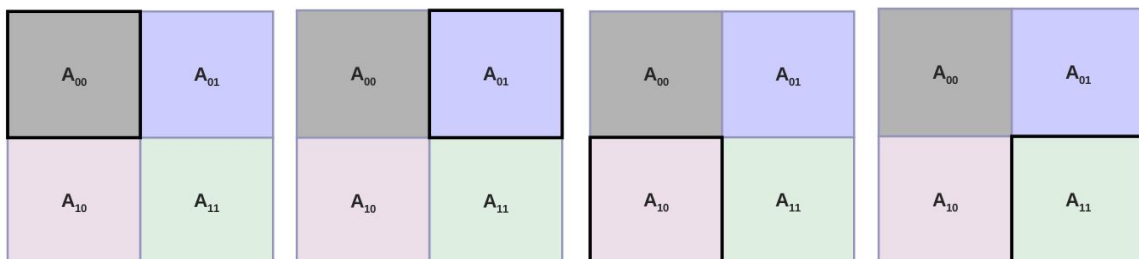
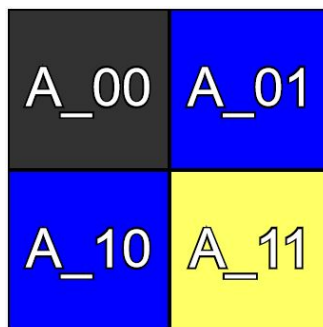
- Το μαύρο block εξαρτάται μόνο από τον εαυτό του.
 - Τα κελιά του μαύρου block, εξαρτάται μόνο από τα κελιά του μαύρου block.
- Τα block των μπλε γραμμών και στηλών είναι ανεξάρτητα μεταξύ τους (εξαρτώνται μόνο από το μαύρο block).
 - Τα κελιά του κάθε μπλε block, εξαρτώνται μόνο από τα κελιά του μαύρου block και τα κελιά του ίδιου του μπλε block.
- Τα κίτρινα tiles είναι ανεξάρτητα μεταξύ τους (εξαρτώνται μόνο από τα μπλε block του σταυρού).
 - Τα κελιά του κάθε κίτρινου block, εξαρτώνται μόνο από τα κελιά των προβολών τους στα μπλε block και του ίδιου του κίτρινου block. (π.χ. Φαίνονται αυτές οι εξαρτήσεις στο παρακάτω παράδειγμα, όπου υποθέτουμε πίνακας με 2x2 block).

Σχετικά με τις αναδρομικές κλήσεις παρατηρούμε, ότι:

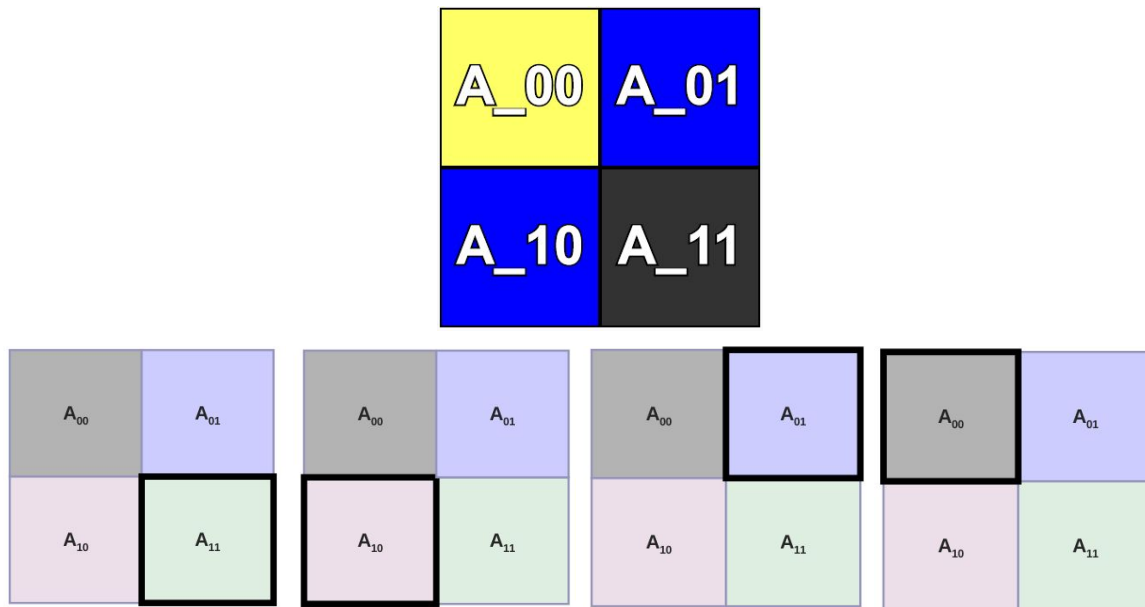
Έστω ότι έχουμε τις αναδρομικές κλήσεις με τον παρακάτω πίνακα 2x2, ώστε οι μόνες κλήσεις που θα γίνουν είναι οι 8 που περιγράψαμε (για απλότητα).

Για τις πρώτες 4 κλήσεις, θέλουμε να εκτελεστεί πρώτα:

- Το A_{00} , το οποίο εξαρτάται μόνο από τον εαυτό του, για αυτό και το καλούμε με τα B_{00}, C_{00} .
- Το A_{01} , το οποίο εξαρτάται μόνο του A_{00} και τον εαυτό του, για αυτό και το καλούμε με τα B_{00}, C_{01} .
- Το A_{10} , το οποίο εξαρτάται μόνο του A_{00} και τον εαυτό του, για αυτό και το καλούμε με τα B_{10}, C_{00} .
- Το A_{11} , το οποίο εξαρτάται μόνο του A_{00} και τον εαυτό του, για αυτό και το καλούμε με τα B_{10}, C_{01} .



Αντίστοιχα, για τις 4 κλήσεις με την αντίστροφη σειρά, ο A καλείται με τους B και C, όπως γράψαμε πάνω ανάλογα με τις εξαρτήσεις του σταυρού.



Για την παραλληλοποίηση, παρατηρούμε ότι:

- Οι κλήσεις των A_{01} , A_{10} υποπινάκων (μπλε block) είναι ανεξάρτητες μεταξύ τους, καθώς η μόνη τους εξάρτηση αφορά τα ίδια τα block και το εκάστοτε μαύρο block (είτε το A_{00} είτε το A_{11}).
- Οπότε αυτές οι κλήσεις είναι παραλληλοποιήσιμες και τις παραλληλοποιούμε μέσω tasks.
- Χρησιμοποιούμε τα κατάλληλα directives (του OpenMP) στον παρακάτω ψευδοκώδικα, για την χρήση των tasks και των tasks wait, ώστε να εξασφαλίσουμε την σειριοποιησιμότητα του προγράμματος.

```

FWR (A, B, C)
if (base case)
    FWI (A, B, C)
else
    #pragma omp task
    FWR (A_00, B_00, C_00);
    #pragma omp task wait

    #pragma omp task
    FWR (A_01, B_00, C_01);
    FWR (A_10, B_10, C_00);
    #pragma omp task wait

    #pragma omp task
    FWR (A_11, B_10, C_01);
    #pragma omp task wait

    #pragma omp task
    FWR (A_11, B_10, C_01);
    #pragma omp task wait

    #pragma omp task
    FWR (A_10, B_10, C_00);
    FWR (A_01, B_00, C_01);
    #pragma omp task wait

    #pragma omp task
    FWR (A_00, B_00, C_00);
    #pragma omp task wait

```

Tiled Αλγόριθμος:

Η τεχνική του tiling, στηρίζεται στην υψηλότερη επαναχρησιμοποίηση κώδικα μέσα στον κώδικα σε loops. Ο tiled αλγόριθμος του FW αναλύεται παρακάτω:

Ο κεντρικός αλγόριθμος:

Τρέχει από **0** έως **N** (μέγεθος πίνακα), και με βήμα **B** (μέγεθος tile, $N = p \cdot B$, $p \in \mathbb{N}$, διαιρεί τον πίνακα σε tiles μεγέθους **B**).

- Κατά την **k**-οστή block επανάληψη, ο αλγόριθμος:
 - Καλεί την συνάρτηση FW, για ανανέωση της τιμής του **k**-οστού διαγώνιου tile (CR - black tile).
 - Καλεί την συνάρτηση FW, για ανανέωση της τιμής των υπόλοιπων tiles που ανήκουν στην στήλη του **k**-οστού block (N, S - blue tiles).
 - Καλεί την συνάρτηση FW, για ανανέωση της τιμής των υπόλοιπων tiles που ανήκουν στην γραμμή του **k**-οστού block (E, W - blue tiles).
 - Καλεί την συνάρτηση FW, για ανανέωση της τιμής των υπόλοιπων tiles του πίνακα (NE, NW, SE, SW - yellow tiles).
 - Πρώτα των NW tiles.
 - Μετά των NE tiles.
 - Μετά των SW tiles.
 - Τέλος των SE tiles.

Η συνάρτηση $FW()$ δέχεται ως **είσοδο**:

- Τον δισδιάστατο Πίνακα (**A**).
 - Καλείται πάντα με τον πίνακα εισόδου **A**.
- Η τιμή **k** για το εξωτερικό loop (**K**).
 - Καλείται κάθε φορά με την δεδομένη τιμή του **k**.
- Η τιμή **i** για το εσωτερικό loop (**I**).
 - Καλείται κάθε φορά με τις δεδομένες τιμές του **i** (για N tiles) ή **i+B** (για S tiles).
- Η τιμή **j** για το εσωτερικό loop (**J**).
 - Καλείται κάθε φορά με τις δεδομένες τιμές του **j** (για W tiles) ή **j+B** (για E tiles).
- Το μέγεθος του υποπίνακα προς επεξεργασία (**N**).
 - Καλείται πάντα με το μέγεθος του tile **B**.

CR	E	E	E	NW	N	NE	NE	NW	NW	N	NE
S	SE	SE	SE	W	CR	E	E	NW	NW	N	NE
S	SE	SE	SE	SW	S	SE	SE	W	W	CR	E
S	SE	SE	SE	SW	S	SE	SE	SW	SW	S	SE

Παράδειγμα εκτέλεσης αλγορίθμου για πίνακα με 4x4 tiles.

Παρατηρούμε, ότι:

- Το μαύρο tile εξαρτάται μόνο από τον εαυτό του.
 - Τα κελιά του μαύρου tile, εξαρτάται μόνο από τα κελιά του μαύρου tile.
- Τα tiles των μπλε γραμμών και στηλών είναι ανεξάρτητα μεταξύ τους (εξαρτώνται μόνο από το μαύρο tile).
 - Τα κελιά του κάθε μπλε tile, εξαρτώνται μόνο από τα κελιά του μαύρου tile και τα κελιά του ίδιου του μπλε tile.
- Τα κίτρινα tiles είναι ανεξάρτητα μεταξύ τους (εξαρτώνται μόνο από τα μπλε tiles του σταυρού).
 - Τα κελιά του κάθε κίτρινου tile, εξαρτώνται μόνο από τα κελιά των προβολών τους στα μπλε tiles και του ίδιου του κίτρινου tile. (π.χ. Φαίνονται αυτές οι εξαρτήσεις στο παρακάτω παράδειγμα, όπου υποθέτουμε πίνακας με 4x4 tiles).

CR	E	E	E
S	SE	SE	SE
S	SE	SE	SE
S	SE	SE	SE

Η παραλληλοποίηση του αλγορίθμου θα γίνει με tasks, και τα παράλληλα κομμάτια είναι:

- Ανανέωση της τιμής του CR, k -οστού διαγώνιου tile.
- (#pragma omp task)
 - Ανανέωση της τιμής των N tiles που ανήκουν στην στήλη του k -οστού block.
- (#pragma omp task)
 - Ανανέωση της τιμής των S tiles που ανήκουν στη στήλη του k -οστού block.
- (#pragma omp task)
 - Ανανέωση της τιμής των E tiles που ανήκουν στην γραμμή του k -οστού block.
- (#pragma omp task)
 - Ανανέωση της τιμής των W tiles που ανήκουν στη γραμμή του k -οστού block.
- (#pragma omp task wait)
- (#pragma omp task)
 - Ανανέωση της τιμής των NE tiles του πίνακα.
- (#pragma omp task)
 - Ανανέωση της τιμής των NW tiles του πίνακα.
- (#pragma omp task)
 - Ανανέωση της τιμής των SW tiles του πίνακα.
- (#pragma omp task)
 - Ανανέωση της τιμής των SE tiles του πίνακα.
- (#pragma omp task wait)