9ο Εξάμηνο, Ακαδημαϊκό Έτος 2021-2022

Συστήματα Παράλληλης Επεξεργασίας

Τελική Αναφορά 2ης Εργαστηριακής Άσκησης: Εισαγωγή στην Παραλληλοποίηση σε Αρχιτεκτονικές Κοινής Μνήμης

parlab20

Ζάρα Στέλλα, 03117154
Λιάγκα Αικατερίνη, 03117208

<u>Σημειώσεις</u>:

1. *Δεν παρατίθενται ξανα τα ερωτήματα που έχουν απαντηθεί στην ενδιάμεση αναφορά.*

2. *Στις απαντήσεις μας παρατίθενται μόνο τα τμήματα του κώδικα που υφίστανται την παραλληλοποίηση / τροποποίηση και όχι ολόκληρος ο κώδικας του εκάστοτε αλγορίθμου.*

3. *Στα αρχεία που κάνουν το make και το run στην ουρά δεν ορίζεται ο επιθυμητός αριθμός δέσμευσης μηχανημάτων, γιατί αυτός ορίστηκε κατά την εκτέλεση της εντολής υποβολής των αρχείων στην ουρά του sandman:*

```
$  qsub -q serial -l nodes=sandman:ppn=64 script.sh
```

4. *Σε όλες τις παρακάτω περιπτώσεις χρησιμοποιήθηκε το εξής Makefile*

```
.phony: all clean

all: fw_tiled_parallel fw_recursive_parallel fw_p

CC=gcc
CFLAGS= -Wall -O3 -fopenmp -Wno-unused-variable

HDEPS+=%.h

OBJS=util.o

fw_tiled_parallel: fw_tiled_parallel.c
	$(CC) $(OBJS) fw_tiled_parallel.c -o fw_tiled_parallel $(CFLAGS)

fw_recursive_parallel: fw_recursive_parallel.c
	$(CC) $(OBJS) fw_recursive_parallel.c -o fw_recursive_parallel $(CFLAGS)

fw_p: fw_parallel_v1.c
	$(CC) $(OBJS) fw_parallel_v1.c -o fw_p $(CFLAGS)

%.o: %.c $(HDEPS)
	$(CC) $(CFLAGS) -c $< -o $@

clean:
	rm -f *.o fw_tiled_parallel
```

Αρχικές παραλληλοποιήσεις

Standard αλγόριθμος Floyd-Warshall

Η πρώτη μας έκδοση παραλληλοποίησης για τον standard αλγόριθμο Floyd-Warshall (που βρίσκεται στο αρχείο *fw_parallel_v1.c)* παρουσιάζεται παρακάτω.

```
for(k=0;k<N;k++)
      #pragma omp parallel for collapse(2) shared(A,k,N) 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]);
```

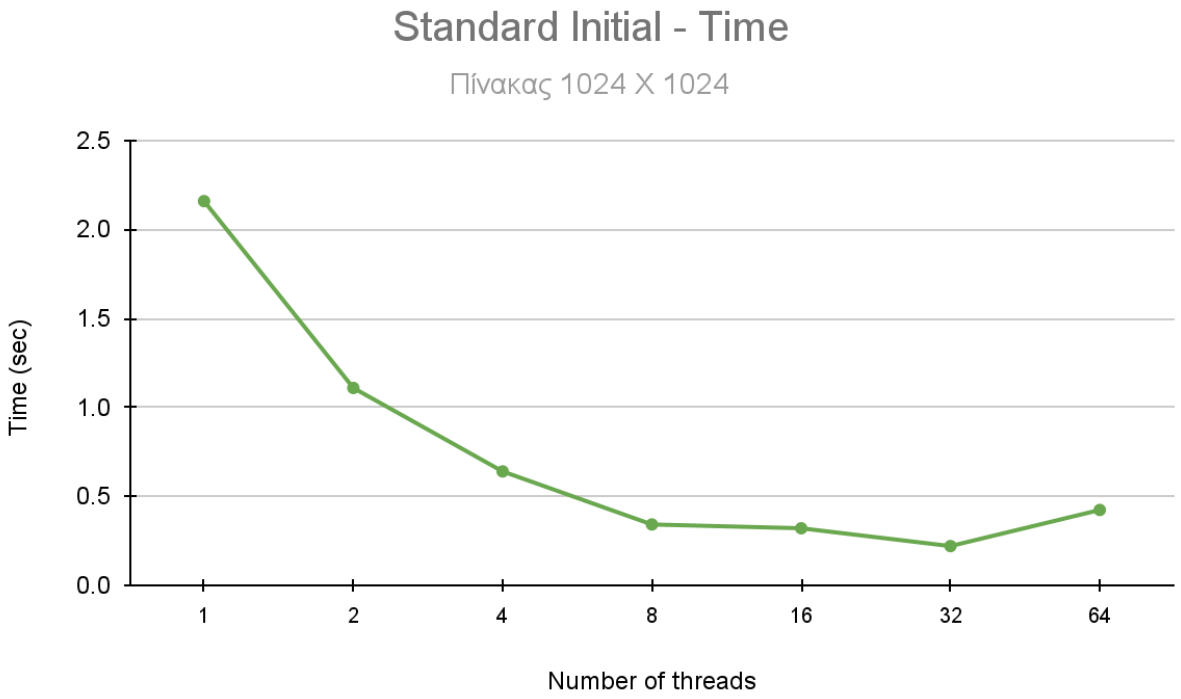
Για την παραλληλοποίηση του κώδικα χρησιμοποιήθηκε η δομή parallel for. Όπως εξηγήθηκε αναλυτικά και στην ενδιάμεση αναφορά, παραλληλοποίηση μπορεί να γίνει μόνο στα δύο εσωτερικά loops, για τα οποία ορίστηκε επιπρόσθετα το directive collapse(2), το οποίο δημιουργεί ένα ενιαίο iteration space βελτιώνοντας την εκτέλεση των δύο εμφωλευμένων βρόγχων. Ακόμα ορίστηκαν κοινές μεταβλητές ο πίνακας A και τα k, N, ενώ τα i, j είναι ιδιωτιά για κάθε thread.

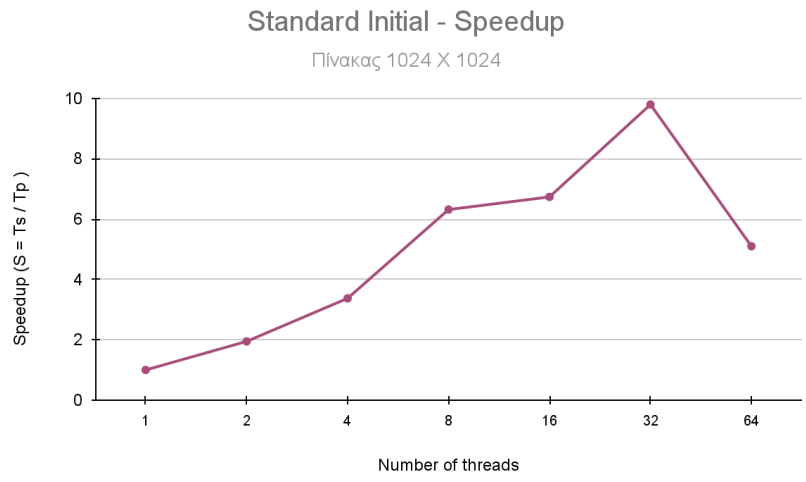
Για να μεταγλωτιστεί και να εκτελεστεί ο κώδικας αυτός δημιουργούμε τα αρχεία *make_on_queue_fw_p.sh* και *run_on_queue_fw_p.sh* τα οποία παρουσιάζονται με την σειρά παρακάτω.

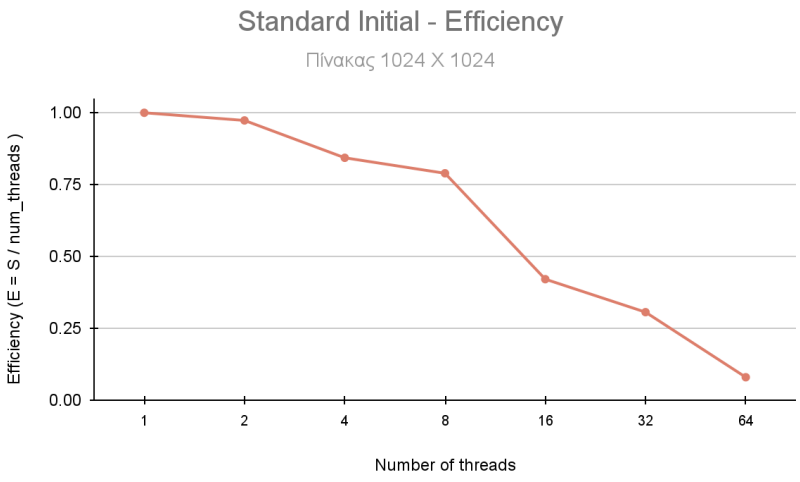
```
#!/bin/bash

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

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

##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/parlab20/a2/FW-parallel
```

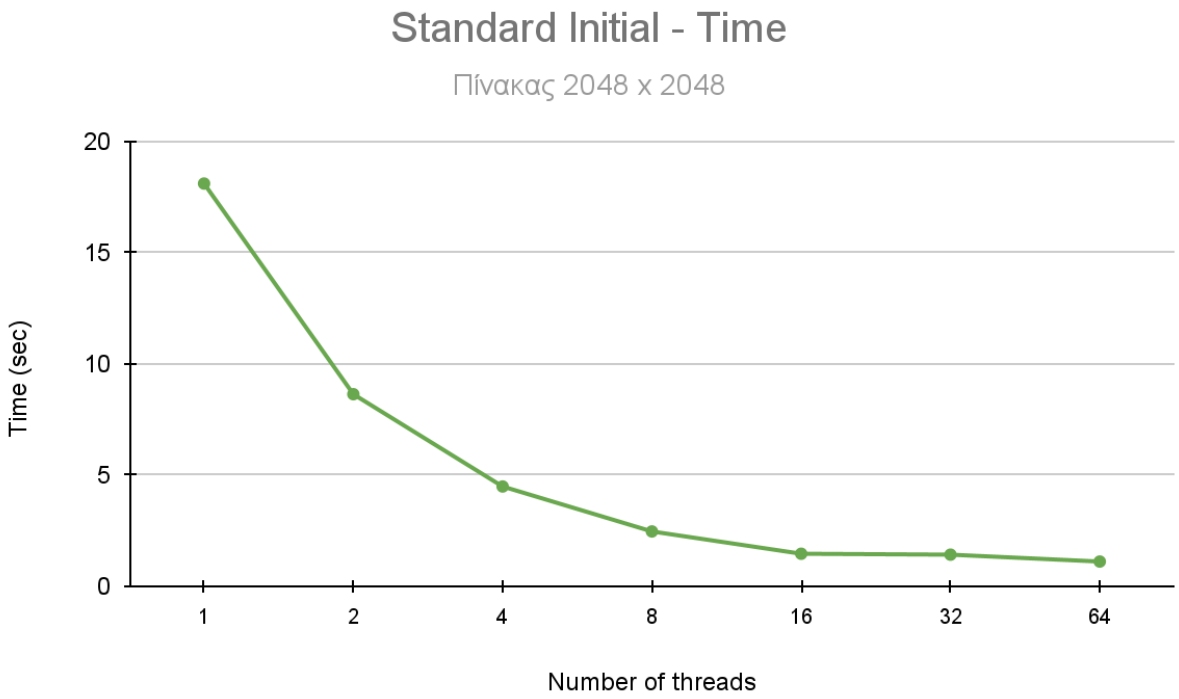
```
#!/bin/bash

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

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

##How long should the job run for?
#PBS -l walltime=00:30:00

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

module load openmp
cd /home/parallel/parlab20/a2/FW-parallel
for thr in 1 2 4 8 16 32 64
do
 export OMP_NUM_THREADS=$thr
 ./fw_p 1024
 ./fw_p 2048
 ./fw_p 4096
done
```

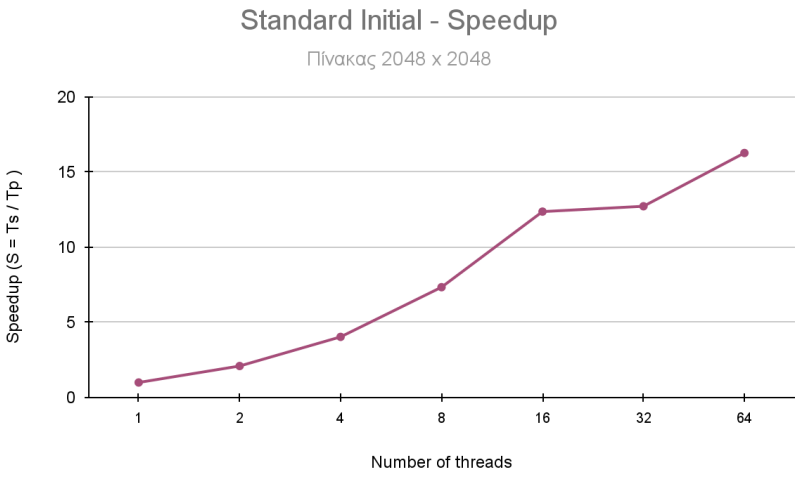
Οι χρόνοι που λαμβάνονται από την εκτέλεση του παραπάνω παραλληλοποιημένου κώδικα, για τα 3 ζητούμενα μεγέθη πινάκων (1024 X 1024, 2048 X 2048 ,4096 X 4096), καθώς και οι πληροφορίες που αφορούν τα speedup και efficiency που προκύπτουν από τους χρόνους αυτούς παρουσιάζονται στα ακόλουθα διαγράμματα.

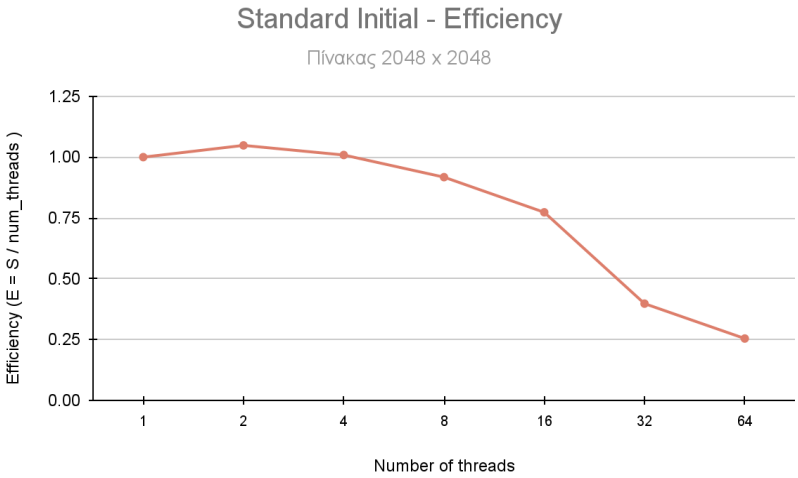

Standard Initial - Time
Πίνακας 1024 X 1024
Time (sec)
2.5
2.0
1.5
1.0
0.5
0.0
1
2
4
8
16
32
64
Number of threads

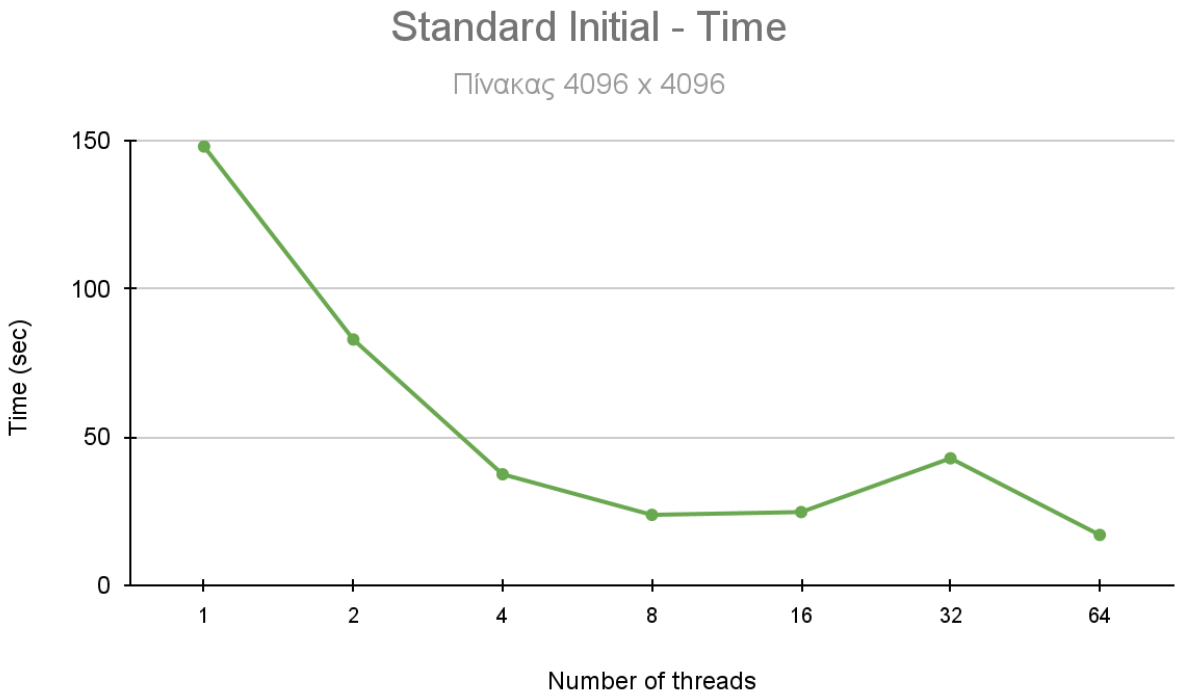

Standard Initial - Speedup
Πίνακας 1024 X 1024
Speedup (S = Ts / Tp)
10
8
6
4
2
0
1
2
4
8
16
32
64
Number of threads

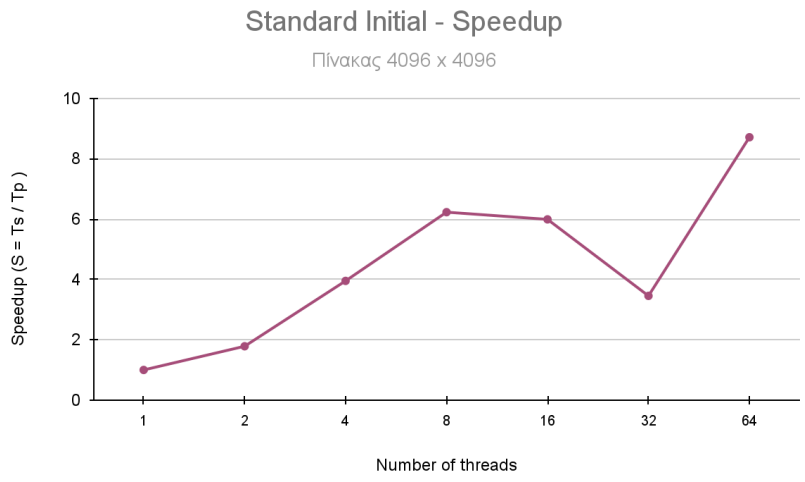

Standard Initial - Efficiency
Πίνακας 1024 X 1024
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads

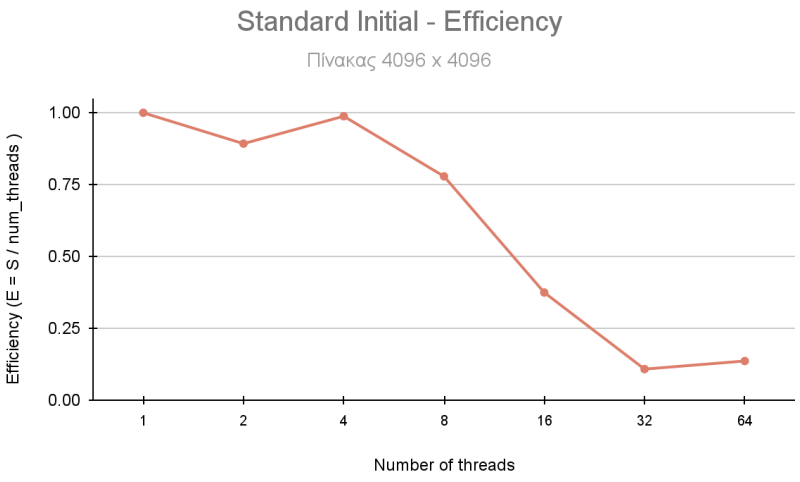

Standard Initial - Time
Πίνακας 2048 x 2048
Time (sec)
20
15
10
5
0
1
2
4
8
16
32
64
Number of threads


Standard Initial - Speedup
Πίνακας 2048 x 2048
Speedup (S = Ts / Tp)
20
15
10
5
0
1
2
4
8
16
32
64
Number of threads


Standard Initial - Efficiency
Πίνακας 2048 x 2048
Efficiency (E = S / num_threads)
1.25
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads


Standard Initial - Time
Πίνακας 4096 x 4096
150
100
50
0
Time (sec)
1
2
4
8
16
32
64
Number of threads


Standard Initial - Speedup
Πίνακας 4096 x 4096
10
8
6
4
2
0
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Standard Initial - Efficiency
Πίνακας 4096 x 4096
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Παρατηρούμε ότι, όπως ήταν αναμενόμενο, ο χρόνος εκτέλεσης μειώνεται με την αύξηση των threads. Συγκεκριμένα αυτό παρατηρείται αδιάκοπα στις περιπτώσεις των μεγάλων πινάκων 2048, 4096, ενώ στην περίπτωση του 1024 η πτωτική τάση ανακόπτεται στα 32 threads. Το γεγονός αυτό είναι φυσιολογικό, αφού οι μεγαλύτεροι πίνακες έχουν μεγαλύτερο όγκο υπολογισμών, οι οποίοι μπορούν να επωφεληθούν από την ύπαρξη περισσότερων threads. Από την άλλη στην περίπτωση του μικρότερου πίνακα, στα 64 threads εμφανίζεται κάποιο bottleneck, αφού ο υπολογιστικός όγκος προφανώς δεν επαρκεί για όλα τα threads και δημιουργούνται καθυστερήσεις, που μάλλον έχουν να κάνουν με το overhead της δημιουργίας και επικοινωνίας των threads που δεν είναι χρήσιμα.

Γενικότερα, αν και όπως αναφέραμε παρατηρουνται μειώσεις, οι χρόνοι είναι παρόμοιοι από τα 8 threads και μετά, γεγονός που δικαιολογεί και την ανακοπή της αυξητικής τάσης του speedup. Ακόμα βλέπουμε μια πτωτική τάση στο efficiency και πολύ πιο έντονα μετα τα 8 threads, το οποίο έχει να κάνει με την μείωση του χρήσιμου όγκου εργασίας ανά thread. Επομένως, αν και βλέπουμε μικρή μείωση του χρόνου και μετά τα 8 threads, η μεγάλη πτώση του efficiency οδηγεί στο συμπέρασμα ότι για παραπάνω από 8 threads δεν έχουμε το ίδιο καλή αξιοποίηση του υλικού, γεγονός το οποίο θα θέλαμε να βελτιωθεί.

Recursive αλγόριθμος Floyd-Warshall

Για τον recursive αλγόριθμο Floyd-Warshall σημειώνεται ότι στην ενδιάμεση αναφορά είχαμε υποθέσει ότι θα μπορούσε να παραλληλοποιηθεί και το τριπλό loop όπως και στον standard αλγόριθμο. Δοκιμάζοντας να παραλληλοποιήσουμε το τριπλό αυτό loop τα αποτελέσματα που λάβαμε ήταν πολύ μεγάλα, μεγαλύτερα ακόμα και από την εκτέλεση του σειριακού recursive FW αλγορίθμου. Για τον λόγο αυτό θεωρήθηκε ότι εκείνη η προσπάθεια παραλληλοποίησης δεν άξιζε καν να παρατεθεί ως ενδιάμεση - πιθανή λύση, αφού δεν υπήρξε βελτίωση της απόδοσης, που είναι και το ζητούμενό μας. Παρακάτω παρουσιάζεται το κομμάτι του κώδικα *fw_recursive_parallel.c* που τελικά υφίσταται παραλληλοποίηση.

```
else {
        #pragma omp parallel
        {
                #pragma omp single
                {
                        FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
                        #pragma omp task shared(A, B, C)
                                FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
                        #pragma omp task shared(A, B, C)
                               FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
                        #pragma omp taskwait
                        FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2, myN/2, bsize);

                        FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol+myN/2, myN/2, bsize);
```

```
                        #pragma omp task shared(A, B, C)
                                FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
                        #pragma omp task shared(A, B, C)
                                 FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2, myN/2, bsize);
                        #pragma omp taskwait
                        FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
                }
        }
}
```

Για την παραλληλοποίηση του κώδικα χρησιμοποιήθηκε η δομή των tasks. Κάθε task εμπεριέχει μία από τις δύο αναδρομές που μπορούν να εκτελεστούν παράλληλα, αφού οι δύο αναδρομές αυτές θέλουμε να ανατεθούν σε διαφορετικά threads. Σημειώνεται ότι έχει προστεθεί το directive shared, ώστε και τα δύο tasks να χρησιμοποιούν κοινούς πίνακες A, B και C. Επιπλέον, επειδή, όπως είδαμε και στην ενδιάμεση αναφορα, η ακριβώς επόμενη αναδρομή εξαρτάται από αυτές τις δύο κάνουμε ένα taskwait, έτσι ώστε η εκτέλεση να μην συνεχιστεί σε περίπτωση που δεν έχει ολοκληρωθεί ο πλήρης υπολογισμός και των δύο tasks. Η ίδια δομή και το ίδιο σκεπτικό χρησιμοποιείται δύο φορές, αφού όπως έχει εξηγηθεί, έχουμε δύο δυάδες αναδρομών που μπορουν να εκτελεστούν παράλληλα. Επίσης αξίζει να αναφερθεί ότι μετα τον ορισμό της παράλληλης περιοχής χρησιμοποιείται η δομή omp single, ώστε να εξασφαλίσουμε τόσο ότι όλα τα υπόλοιπα κομμάτια του κώδικα, πέραν των tasks, θα εκτελεστούν σειριακά από ένα thread, όπως επιβάλλεται λόγω των εξαρτήσεων, όσο και ότι τα tasks θα έχουν τον ίδιο γονέα και θα δουλέψει όπως περιμένουμε το takswait.

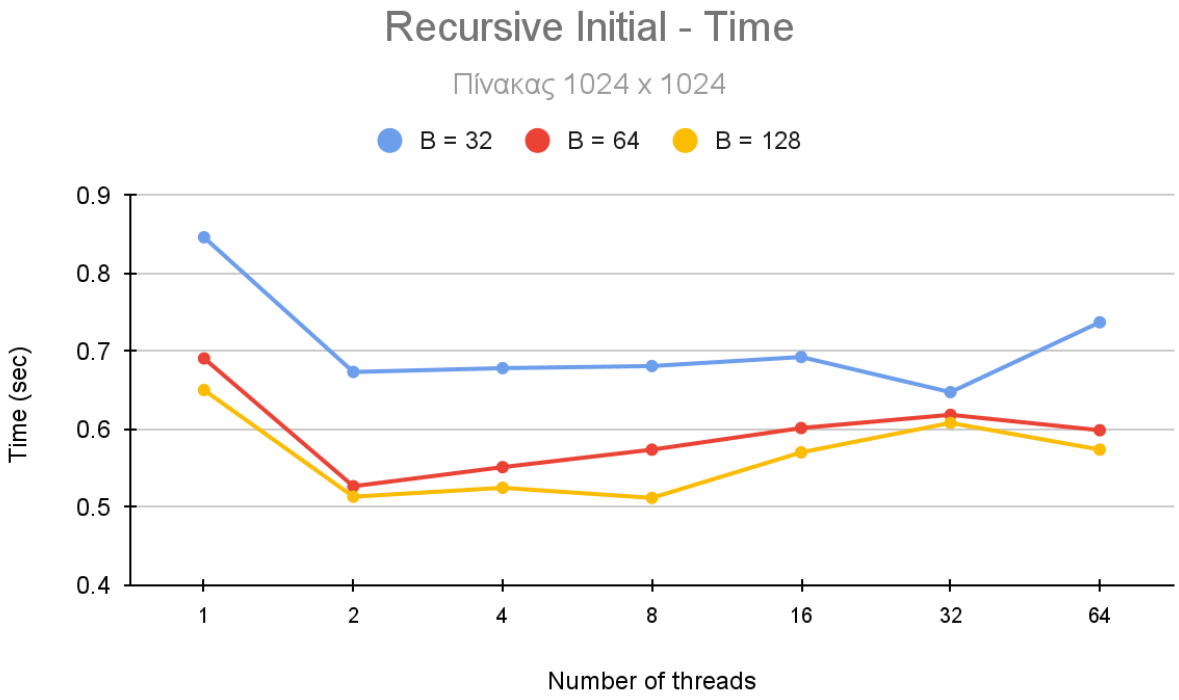
Για να μεταγλωτιστεί και να εκτελεστεί ο κώδικας αυτός δημιουργούμε τα αρχεία *make_on_queue_recursive_p.sh* και *run_on_queue_recursive_p.sh* , τα οποία παρουσιάζονται με την σειρά παρακάτω. Σημειώνεται ότι εδώ, πέραν των διαφορετικών μεγεθών πίνακα, ορίζουμε και διαφορετικά μεγέθη υποπινάκων (32, 64, 128) για τα οποία θα εκτελείται το base case, δηλαδή το μέγεθος υποπίνακα για το οποίο σταματούν κάθε φορά οι αναδρομές και εκτελείται το if clause (στα διαγράμματα αναφέρεται ως B).

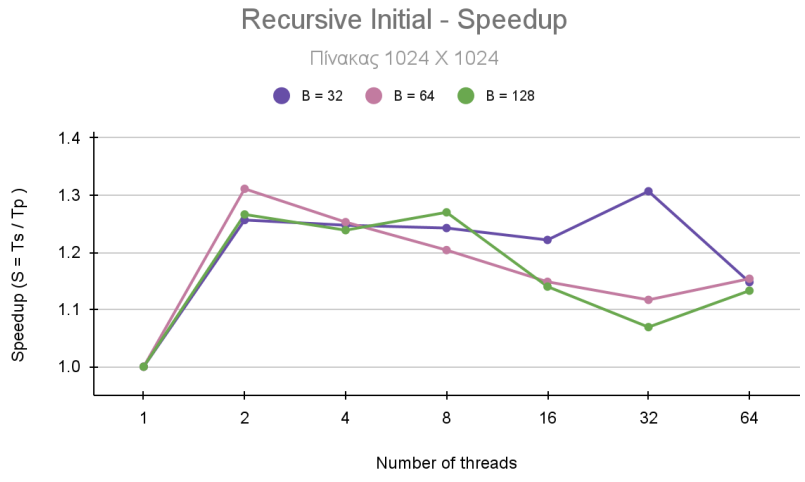
```
#!/bin/bash

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

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

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

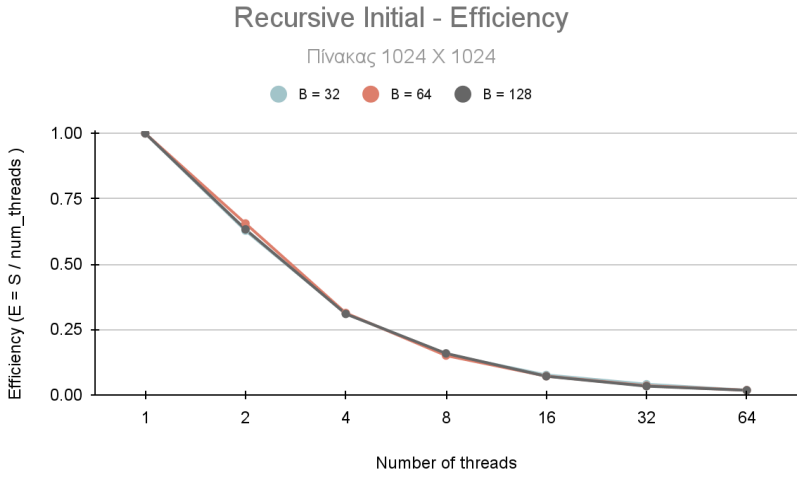
```
## Start
## Run make in the src folder (modify properly)

module load openmp
cd /home/parallel/parlab20/a2/FW-parallel
make
```

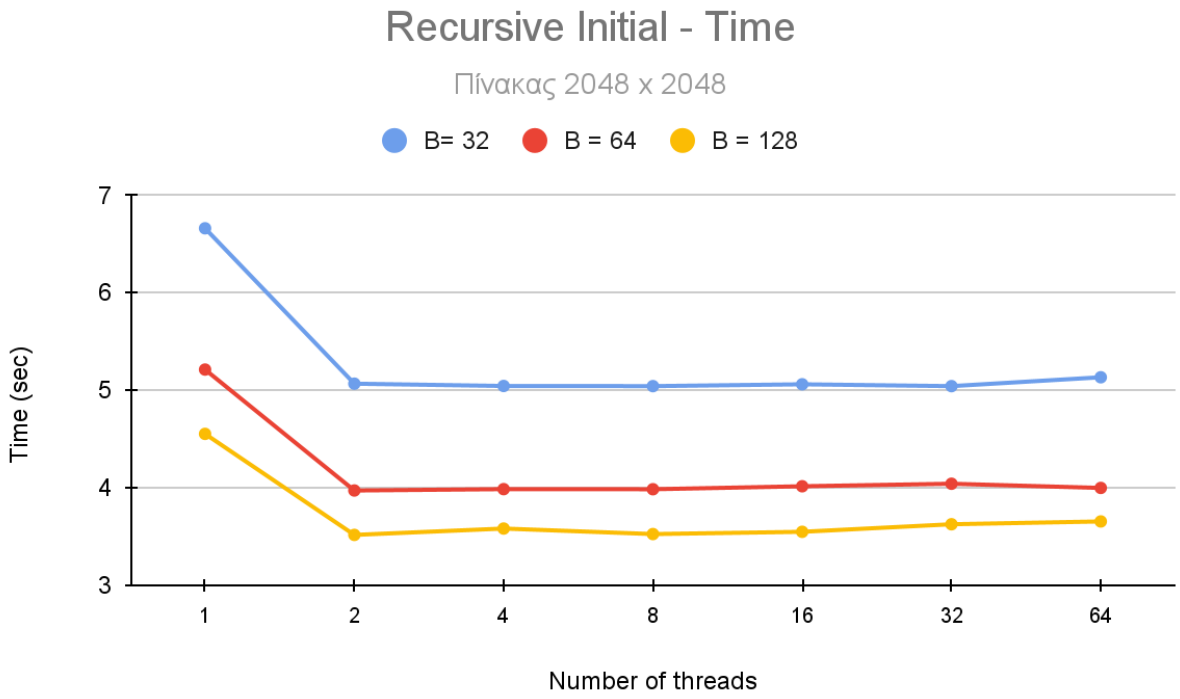
```
#!/bin/bash

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

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

##How long should the job run for?
#PBS -l walltime=00:30:00

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

module load openmp
cd /home/parallel/parlab20/a2/FW-parallel

for thr in 1 2 4 8 16 32 64
do
  export OMP_NUM_THREADS=$thr
  ./fw_recursive_parallel 1024 32
  ./fw_recursive_parallel 2048 32
  ./fw_recursive_parallel 4096 32
  ./fw_recursive_parallel 1024 64
  ./fw_recursive_parallel 2048 64
  ./fw_recursive_parallel 4096 64
  ./fw_recursive_parallel 1024 128
  ./fw_recursive_parallel 2048 128
  ./fw_recursive_parallel 4096 128

done
```

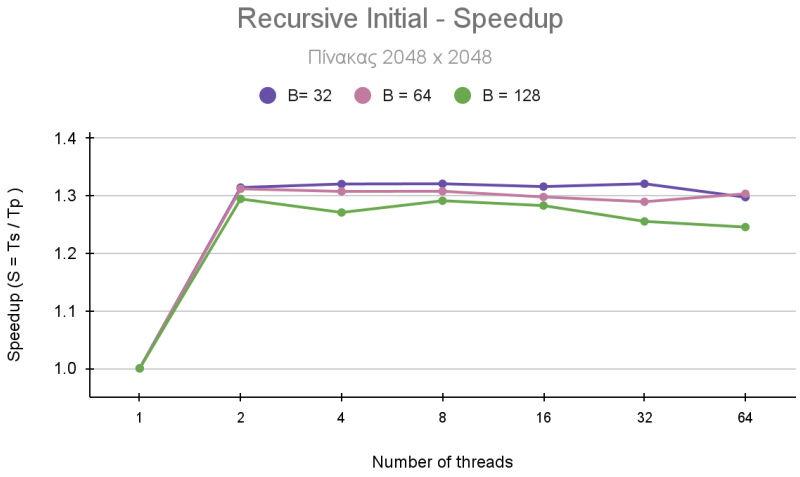
Οι χρόνοι που λαμβάνονται από την εκτέλεση του παραπάνω παραλληλοποιημένου κώδικα, καθώς και οι πληροφορίες που αφορούν τα speedup και efficiency που προκύπτουν από τους χρόνους αυτούς παρουσιάζονται στα ακόλουθα διαγράμματα.

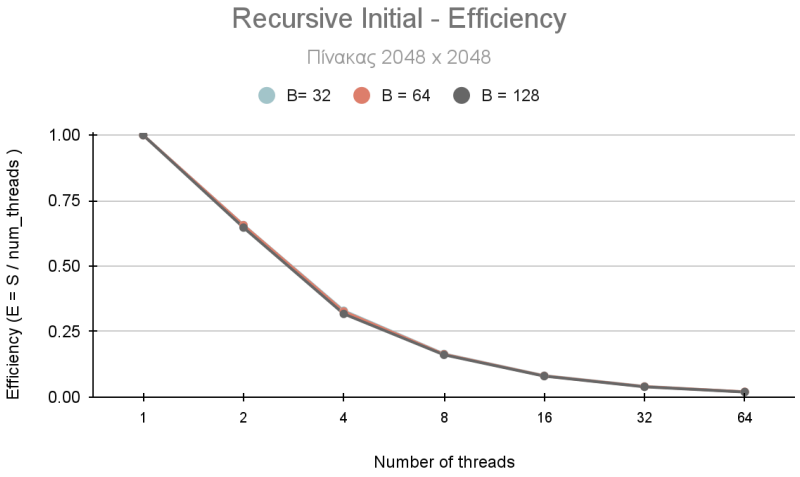


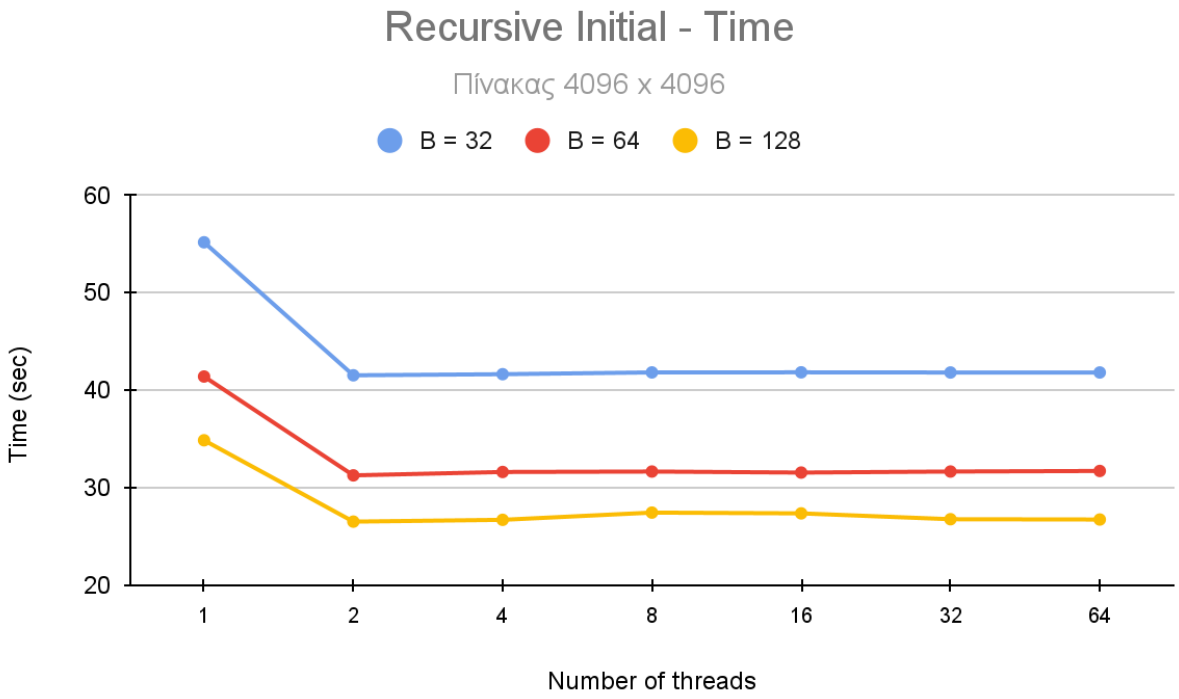


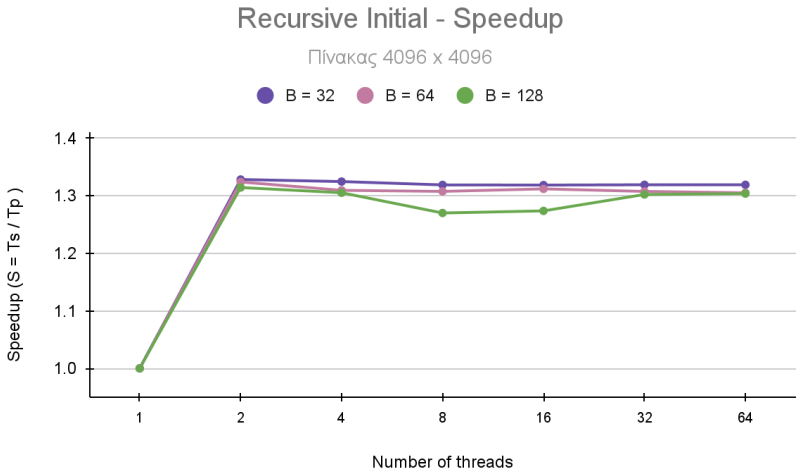


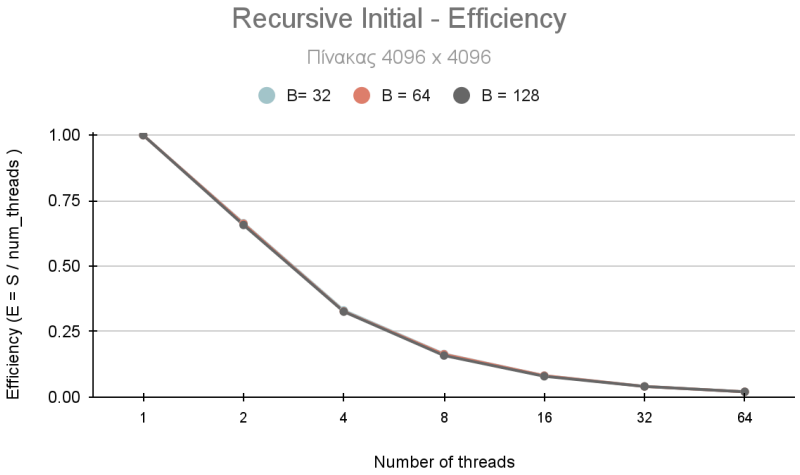

Recursive Initial - Time
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Time (sec)
7
6
5
4
3
1
2
4
8
16
32
64
Number of threads


Recursive Initial - Speedup
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Speedup (S = Ts / Tp)
1.4
1.3
1.2
1.1
1.0
1
2
4
8
16
32
64
Number of threads


Recursive Initial - Efficiency
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads


Recursive Initial - Time
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
60
50
40
30
20
Time (sec)
1
2
4
8
16
32
64
Number of threads


Recursive Initial - Speedup
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
1.4
1.3
1.2
1.1
1.0
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Recursive Initial - Efficiency
Πίνακας 4096 x 4096
B= 32
B = 64
B = 128
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Παρατηρείται ότι μείωση του χρόνου εκτέλεσης έχουμε μόνο για τα 2 threads, σε όλα τα μεγέθη πινάκων 1024, 2048 και 4096. Για περισσότερα από 2 threads, όσο και να αυξάνονται αυτά, ο χρόνος εκτέλεσης παραμένει σχετικά σταθερός, ενώ σε κάποιες περιπτώσεις παρατηρούνται ακόμα και μικρές αυξήσεις. Το ότι δεν υπάρχει βελτίωση του χρόνου μετά τα 2 threads αντικατοπτρίζεται και στο γεγονός ότι το speedup αυξάνεται μόνο στην μετάβαση από τα 1 στα 2 threads και μετά παρατηρούμε στασιμότητα ή και μείωσή του. Αυτό πιστεύουμε πως δικαιολογείται από την φύση του αλγόριθμου, αφού το μόνο που μπορεί να εκτελεστεί ταυτόχρονα είναι 2 αναδρομές, οι οποίες θα πρέπει να υπολογιστούν πλήρως για να προχωρήσουμε στην επόμενη αναδρομή λόγω των εξαρτήσεων που έχουν αναλυθεί στην ενδιάμεση αναφορά (για αυτόν ακριβώς το λόγο χρησιμοποιήθηκε και το taskwait).

Επομένως είναι προφανές ότι για βελτίωση του χρόνου πρέπει να παραλληλοποιήσουμε με 2 threads, ενω για μεγαλύτερο αριθμό threads έχουμε απλά σπατάλη των διαθέσιμων πόρων χωρίς να βλέπουμε καμία βελτίωση απόδοσης, κάτι το οποίο επιβεβαιώνεται από την συνεχή και ταχεία μείωση του efficiency, που καταλήγει να παίρνει εξαιρετικά χαμηλές τιμές για μεγάλο αριθμό threads.

Tiled αλγόριθμος Floyd-Warshall

Ο παραλληλοποιημένος κώδικας που δοκιμάσαμε αρχικά για τον αλγόριθμο tiled Floyd-Warshall παρατίθεται παρακάτω.

```
    for(k=0;k<N;k+=B){

       FW(A,k,k,k,B);

       #pragma omp parallel shared(A,B,k) private(i,j)
       {

       #pragma omp for nowait
       for(i=0; i<k; i+=B)
          FW(A,k,i,k,B);

       #pragma omp for nowait
       for(i=k+B; i<N; i+=B)
          FW(A,k,i,k,B);

       #pragma omp for nowait
       for(j=0; j<k; j+=B)
          FW(A,k,k,j,B);
```

```
        #pragma omp for nowait
        for(j=k+B; j<N; j+=B)
           FW(A,k,k,j,B);
        }

        #pragma omp parallel shared(A,B,k) private(i,j)
        {

        #pragma omp for collapse(2) nowait
        for(i=0; i<k; i+=B)
           for(j=0; j<k; j+=B)
              FW(A,k,i,j,B);

        #pragma omp for collapse(2) nowait
        for(i=0; i<k; i+=B)
           for(j=k+B; j<N; j+=B)
              FW(A,k,i,j,B);

        #pragma omp for collapse(2) nowait
        for(i=k+B; i<N; i+=B)
           for(j=0; j<k; j+=B)
              FW(A,k,i,j,B);

        #pragma omp for collapse(2) nowait
        for(i=k+B; i<N; i+=B)
           for(j=k+B; j<N; j+=B)
              FW(A,k,i,j,B);
        }

        }
    }
```

Για την παραλληλοποίηση του κώδικα χρησιμοποιήθηκαν δομές parallel for και στην ουσία ο κώδικας χωρίστηκε σε 2 μεγάλα διαδοχικά κομμάτια παραλληλοποιήσιμου κώδικα, ένα που αφορούσε τον υπολογισμό των tiles που βρίσκονται στην ίδια γραμμή και την ίδια στήλη με το αρχικό tile και ένα που αφορούσε τον υπολογισμό των υπόλοιπων tiles.

Στο πρώτο κομμάτι, το οποίο αποτελείται από 4 μονά for loops, ορίστηκαν κοινές μεταβλητές ο πίνακας A και τα k, B, ενώ τα i, j είναι ιδιωτικά για κάθε thread. Μέσα σε αυτήν την παράλληλη περιοχή χρησιμοποιήθηκε το directive για τα for loops με την επισήμανση nowait ώστε να μην χρειάζεται κάποιο thread να περιμένει για να συνεχίσει την εκτέλεση.

Στο δεύτερο κομμάτι, το οποίο αποτελείται από 4 διπλά for loops, ορίστηκαν ξανά κοινές μεταβλητές ο πίνακας A και τα k, B, ενώ τα i, j είναι ιδιωτικά για κάθε thread. Μέσα σε αυτήν την παράλληλη περιοχή χρησιμοποιήθηκε το directive για τα for loops με την επισήμανση nowait, καθώς και το collapse(2) ώστε να δημιουργηθεί κοινό iteration space για τους εμφωλευμένους βρόγχους και να βοηθηθεί και άλλο η παραλληλοποίηση του for.

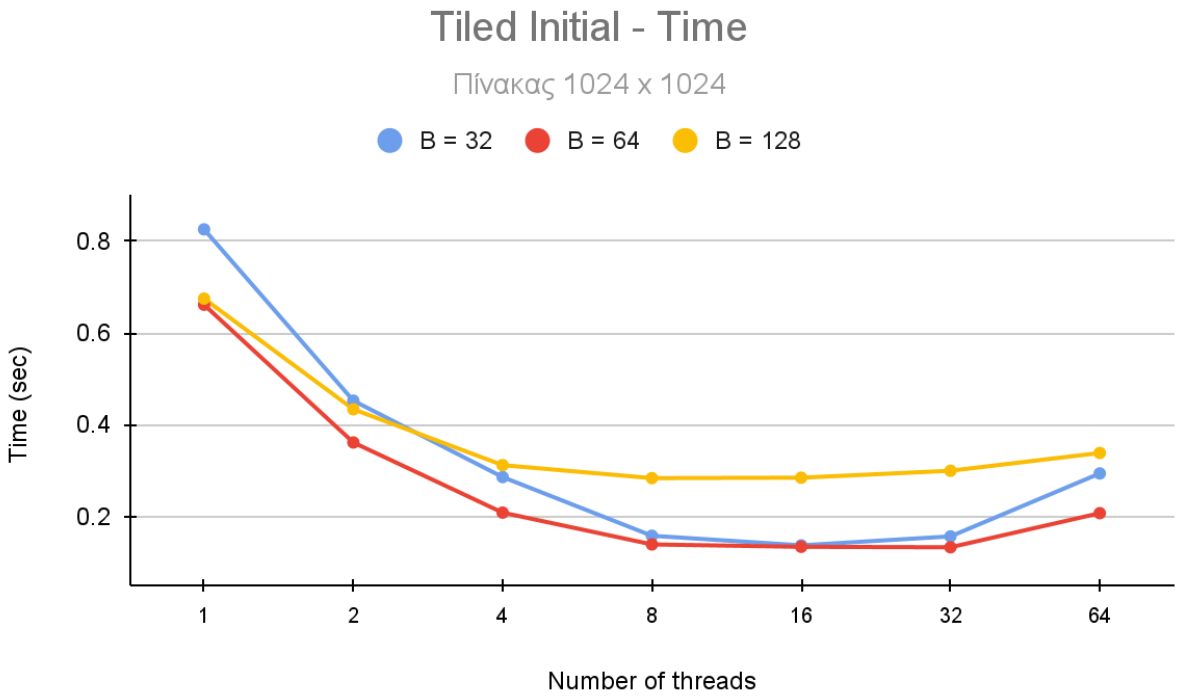
Οι δύο παράλληλες περιοχές ορίστηκαν ξεχωριστά η μία από την άλλη καθώς, όπως είδαμε στην ενδιάμεση αναφορά, θα πρέπει πρώτα να έχουν υπολογιστεί τα tiles που βρίσκονται στην ίδια στήλη και την ίδια γραμμή με το αρχικό tile, και ύστερα να περάσουμε στον υπολογισμό των υπολοίπων, αφού υπάρχει εξάρτηση ανάμεσα σε αυτούς τους υπολογισμούς.

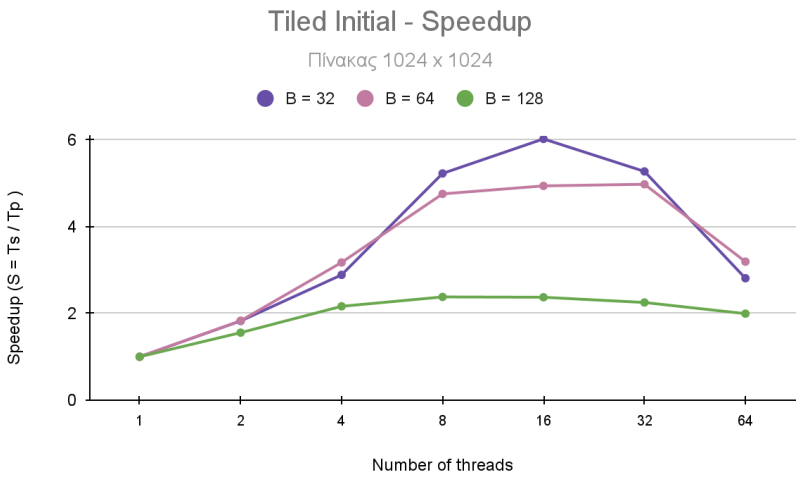
Τα αντίστοιχα αρχεία *make_on_queue_tiled_p.sh* και *run_on_queue_tiled_p.sh* παρουσιάζονται παρακάτω. Σημειώνεται ότι και σε αυτή την περίπτωση τρέχουμε τα προγράμματα με ίδιο μέγεθος πίνακα για διαφορετικα μεγέθη B (32, 64, 128), τα οποία σε αυτή την περίπτωση είναι το μέγεθος του tile που χρησιμοποιείται.

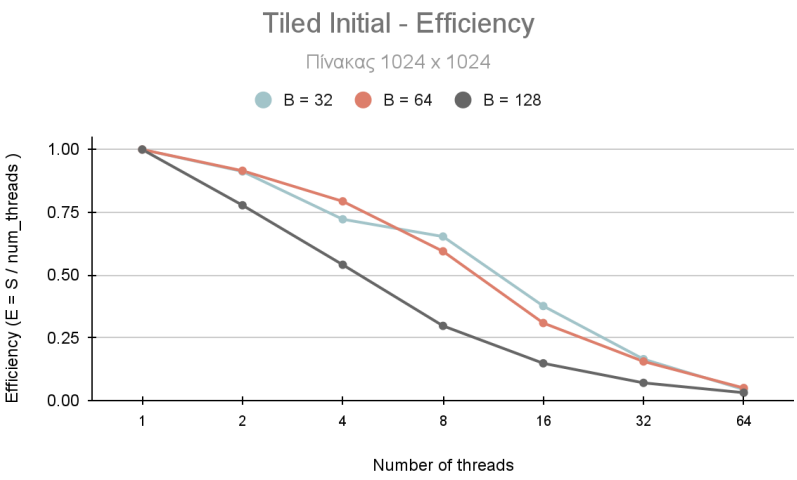
```
#!/bin/bash

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

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

##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/parlab20/a2/FW-parallel
make
```

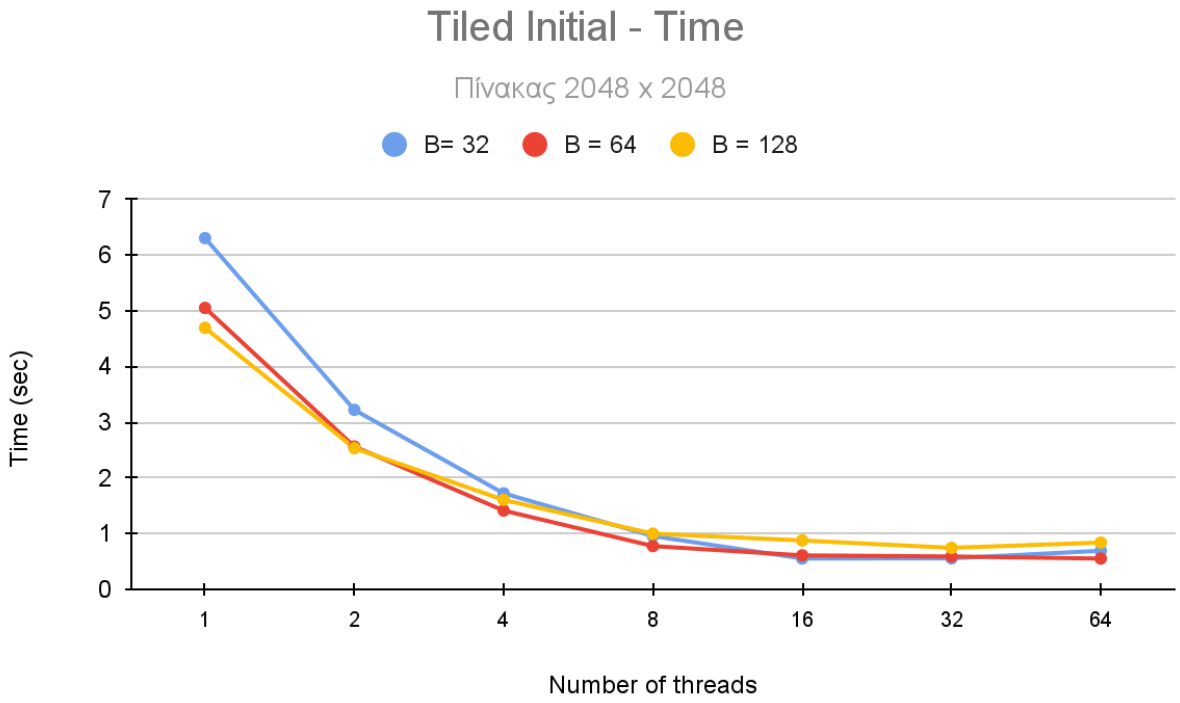
```
#!/bin/bash

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

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

##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/parlab20/a2/FW-parallel

for thr in 1 2 4 8 16 32 64
  do
   export OMP_NUM_THREADS=$thr
   ./fw_tiled_parallel 1024 32
   ./fw_tiled_parallel 2048 32
   ./fw_tiled_parallel 4096 32
   ./fw_tiled_parallel 1024 64
   ./fw_tiled_parallel 2048 64
   ./fw_tiled_parallel 4096 64
   ./fw_tiled_parallel 1024 128
   ./fw_tiled_parallel 2048 128
   ./fw_tiled_parallel 4096 128
  done
```

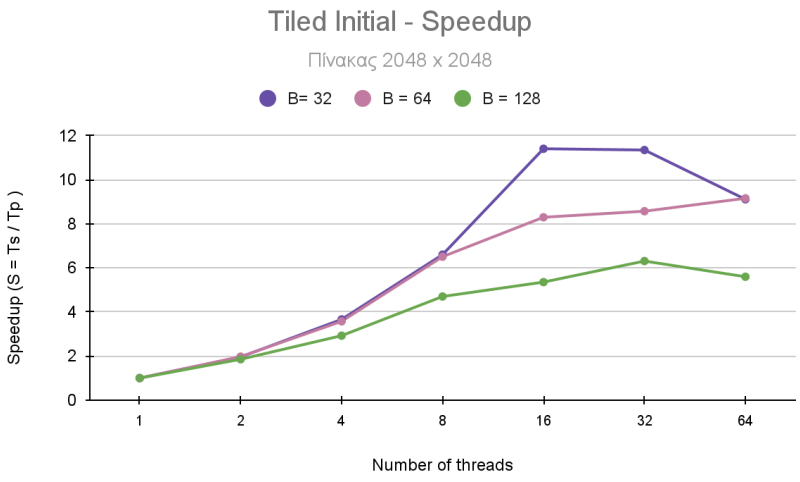
Οι χρόνοι που λαμβάνονται από την εκτέλεση του tiled FW παραλληλοποιημένου κώδικα, καθώς και οι πληροφορίες που αφορούν τα speedup και efficiency που προκύπτουν από τους χρόνους αυτούς παρουσιάζονται στα ακόλουθα διαγράμματα.

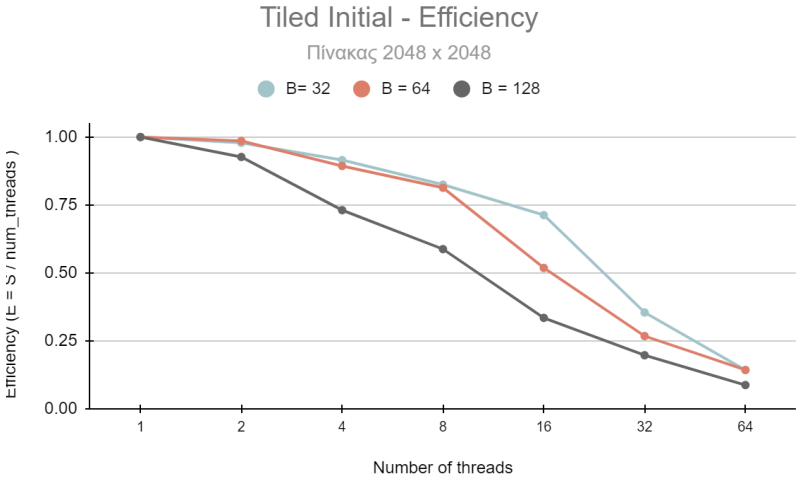

Tiled Initial - Time
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Time (sec)
0.8
0.6
0.4
0.2
1
2
4
8
16
32
64
Number of threads

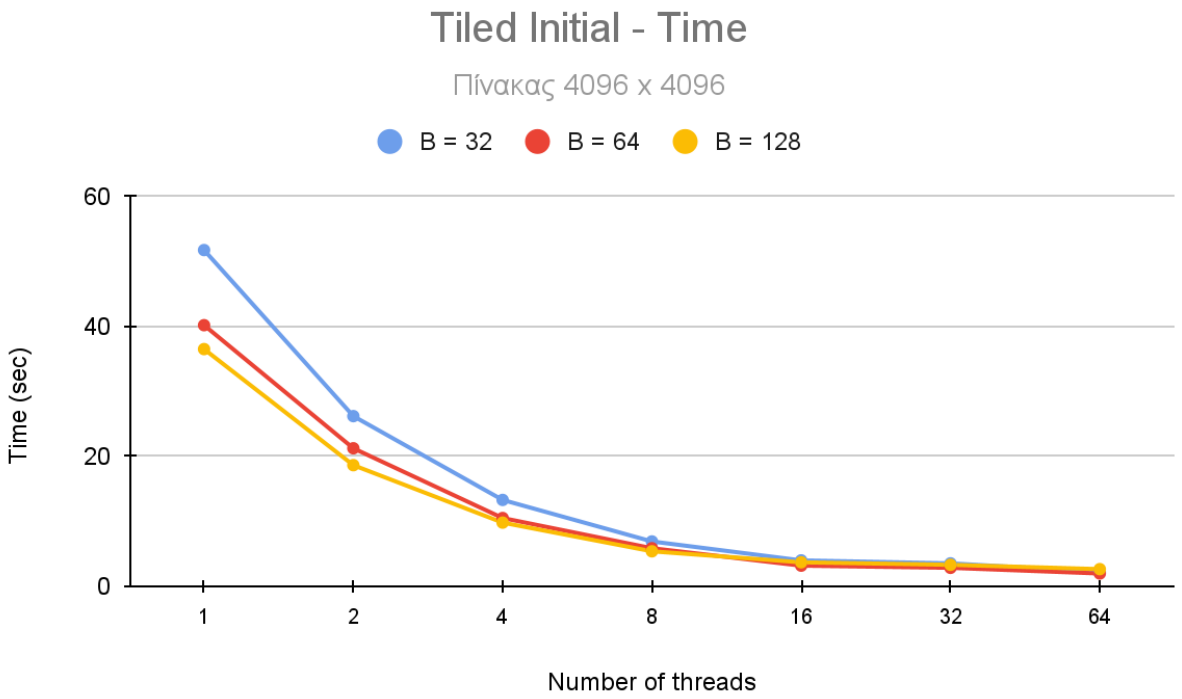

Tiled Initial - Speedup
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Speedup (S = Ts / Tp)
6
4
2
0
1
2
4
8
16
32
64
Number of threads

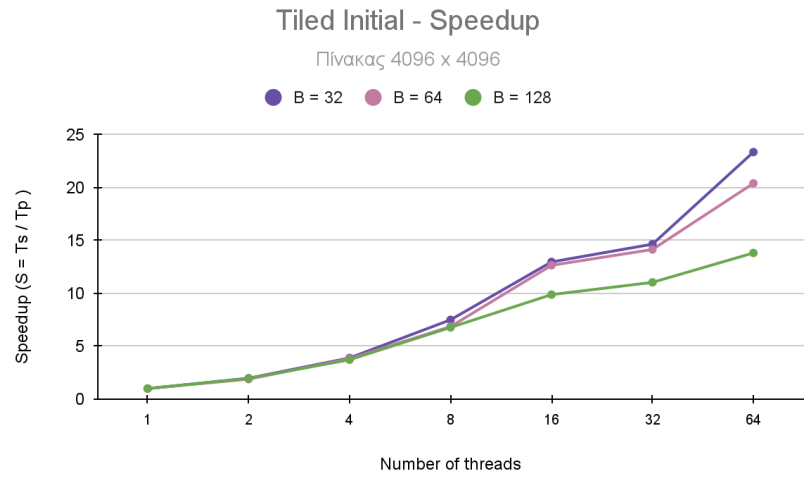

Tiled Initial - Efficiency
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads

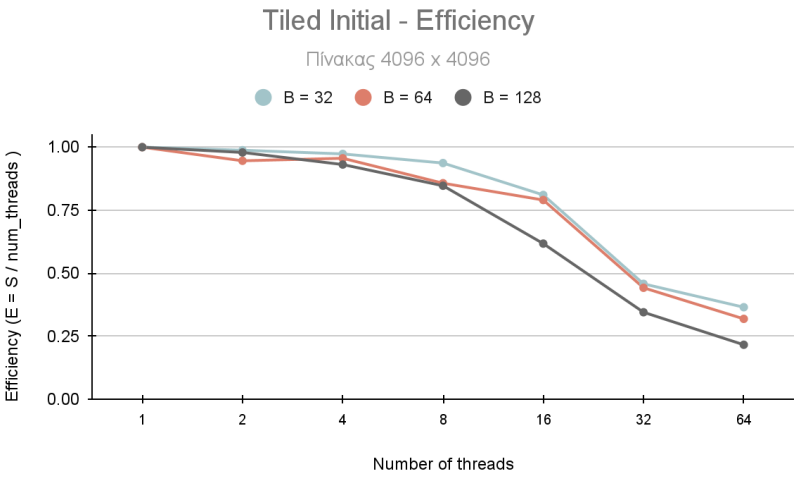

Tiled Initial - Time
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Time (sec)
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64
Number of threads


Tiled Initial - Speedup
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Speedup (S = Ts / Tp)
0
2
4
6
8
10
12
1
2
4
8
16
32
64
Number of threads


Tiled Initial - Efficiency
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Efficiency (E = S / num_threads)
0.00
0.25
0.50
0.75
1.00
1
2
4
8
16
32
64
Number of threads


Tiled Initial - Time
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
60
40
20
0
Time (sec)
1
2
4
8
16
32
64
Number of threads


Tiled Initial - Speedup
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
25
20
15
10
5
0
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Tiled Initial - Efficiency
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Παρατηρείται μείωση του χρόνου εκτέλεσης μέχρι και τα 64 Threads για τις περιπτώσεις των πινάκων με μέγεθος 2048 και 4096, ενώ για τον πίνακα 1024 μετά τα 8 threads υπάρχει μια στασιμότητα στους χρόνους εκτέλεσης με την εμφάνιση αύξησης στα 64. Όπως εξηγήθηκε και στον standard αλγόριθμο αυτό είναι λογικό, αφού για μικρότερα μεγέθη πινάκων έχουμε μικρότερο υπολογιστικό φόρτο και άρα λιγότερα threads που εκτελούν χρήσιμη δουλειά. Λόγω της μείωσης του χρόνου τα speedup σε γενικές γραμμές σημειώνουν αύξηση, με εξαίρεση την περίπτωση του 1024, που μετα τα 8 threads έχουμε σχετική στασιμότητα ή καθοδική τάση.
Όσον αφορά το efficiency παρατηρούμε το πως επηρεάζεται από το μέγεθος του tile. Συγκεκριμένα παρατηρούμε ότι όσο μικρότερο είναι το tile, τόσο μεγαλύτερο είναι το efficiency, καθώς μικρότερο tile συνεπάγεται λιγότεροι υπολογισμοί εντός αυτού, με το τριπλό for loop που δεν παραλληλοποιείται, και μεγαλύτερος φόρτος υπολογισμών στην παραλληλοποιήσιμη περιοχή. Αφού, λοιπόν, μεγαλύτερος όγκος υπολογισμών γίνεται στην παραλληλοποιημένη περιοχή έχουμε περισσότερα threads που κάνουν χρήσιμη δουλειά, άρα και μεγαλύτερο efficiency.

Βελτιωμένες παραλληλοποιήσεις

Standard αλγόριθμος Floyd-Warshall

Καταφέραμε να βελτιώσουμε τον χρόνο στον οποίο τρέχει ο standard αλγόριθμος Floyd-Warshall, παραλληλοποιώντας τη διαδικασία αρχικοποίησης του πίνακα μέσα στη συνάρτηση graph_init_random η οποία βρίσκεται στο αρχείο util.c, όπως φαίνεται παρακάτω:

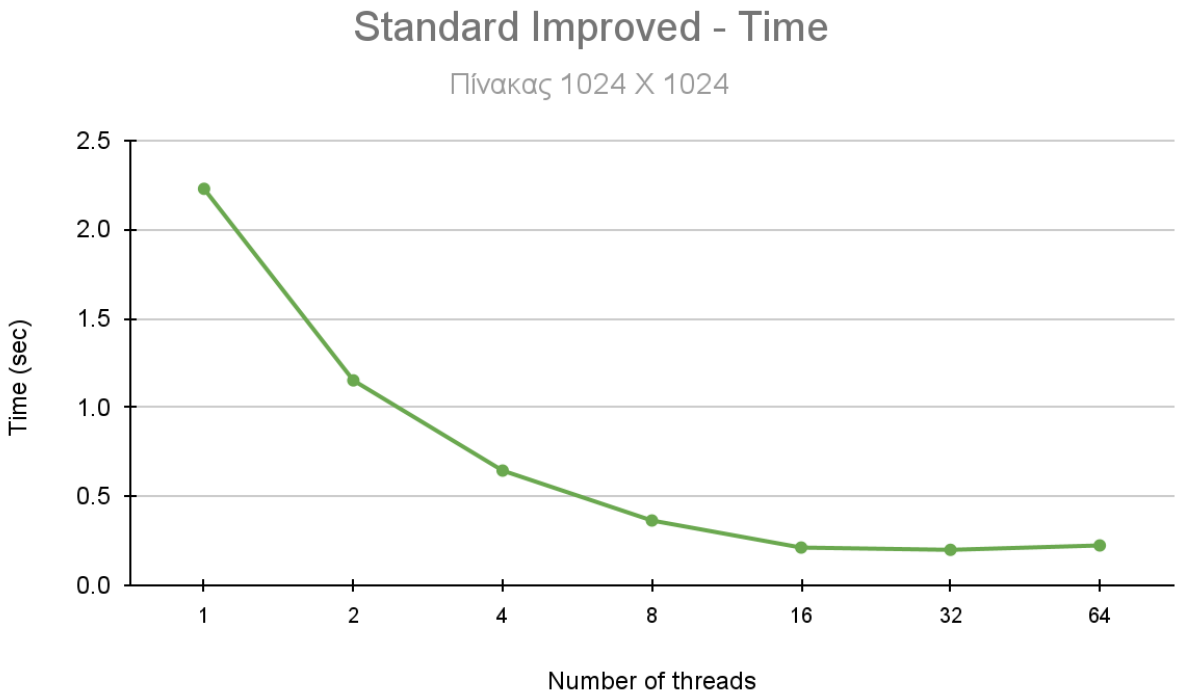
```
#pragma omp parallel for shared(adjm,n) private(i,j)
for(i=0; i<n; i++)
   for(j=0; j<n; j++)
      adjm[i][j] = abs((( int)lrand48()) % 1048576);
```

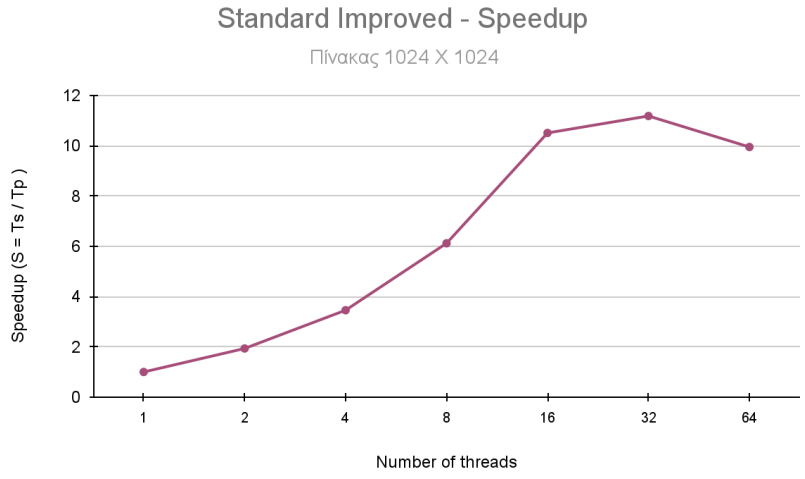
Για την παραλληλοποίηση του κώδικα χρησιμοποιήθηκε μια δομή parallel for. Για την ένταξη της αλλαγής στο εκτελέσιμο του standard αλγόριθμου, δημιουργήθηκε το νέο object file util.o και έγινε ξανά compile το πρόγραμμα. Τα βήματα αυτά γίνονται στο αρχείο *make_on_queue_fw_p.sh,* το οποίο τροποποιήθηκε κατάλληλα, ενώ το αρχείο *run_on_queue_fw_p.sh* χρησιμοποιήθηκε όπως παρουσιάστηκε παραπάνω.

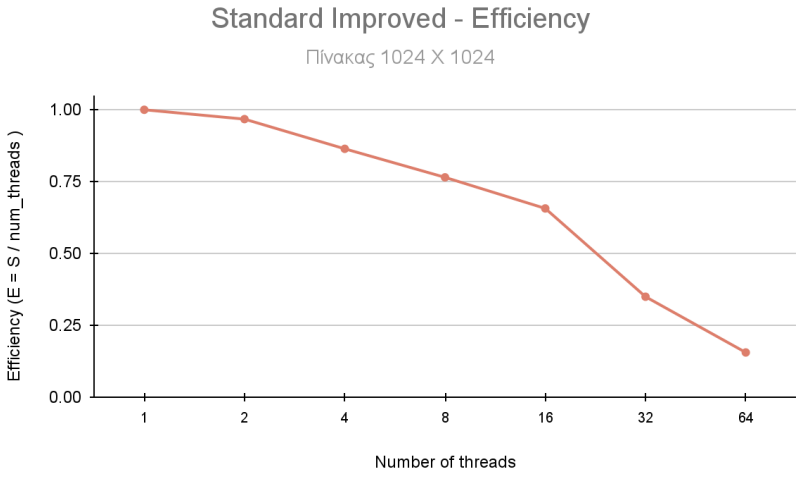
Το τροποποιημένο *make_on_queue_fw_p.sh* παρατίθεται παρακάτω.

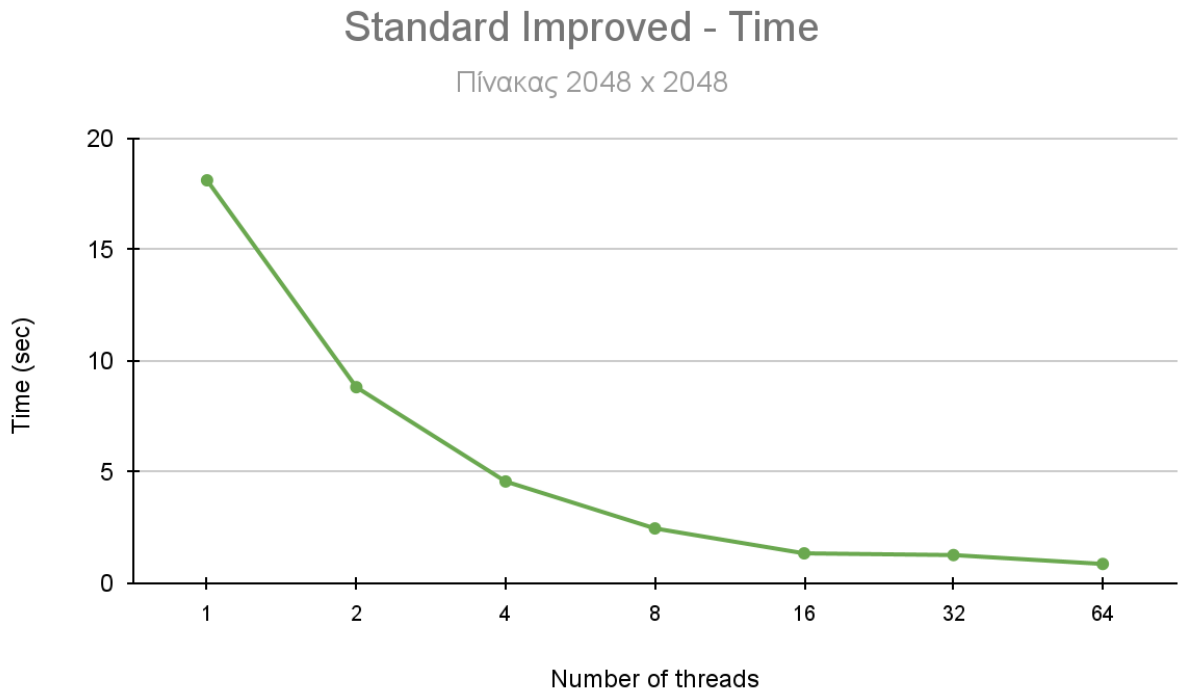
```
#!/bin/bash

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

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

##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/parlab20/a2/FW-parallel
make clean
gcc -Wall -fopenmp -c util.c
gcc util.o fw_parallel_v1.c -o fw_p -Wall -O3 -fopenmp
-Wno-unused-variable
```

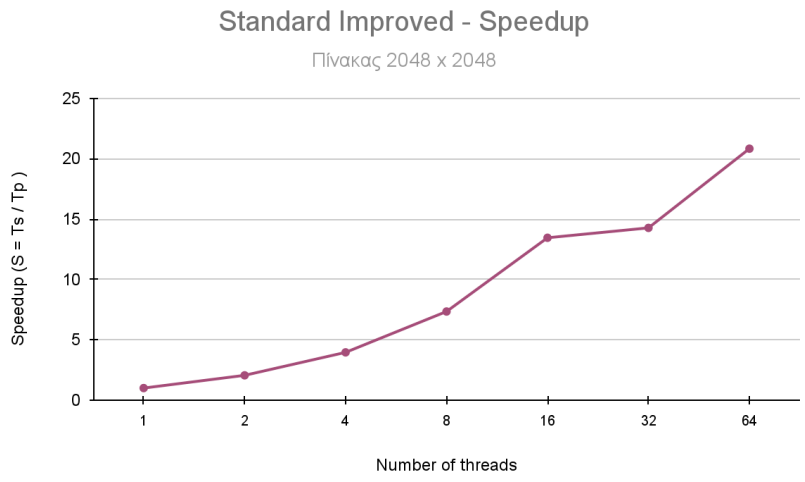
Οι χρόνοι που λαμβάνονται από την εκτέλεση της παραπάνω βελτιωμένης έκδοσης του παραλληλοποιημένου κώδικα και οι πληροφορίες που αφορούν τα speedup και efficiency που προκύπτουν από τους χρόνους αυτούς παρουσιάζονται στα ακόλουθα διαγράμματα.

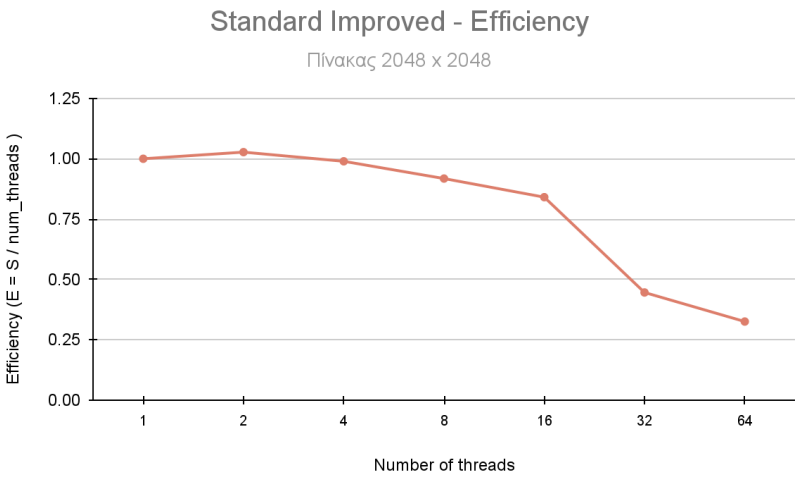

Standard Improved - Time
Πίνακας 1024 X 1024
Time (sec)
2.5
2.0
1.5
1.0
0.5
0.0
1
2
4
8
16
32
64
Number of threads

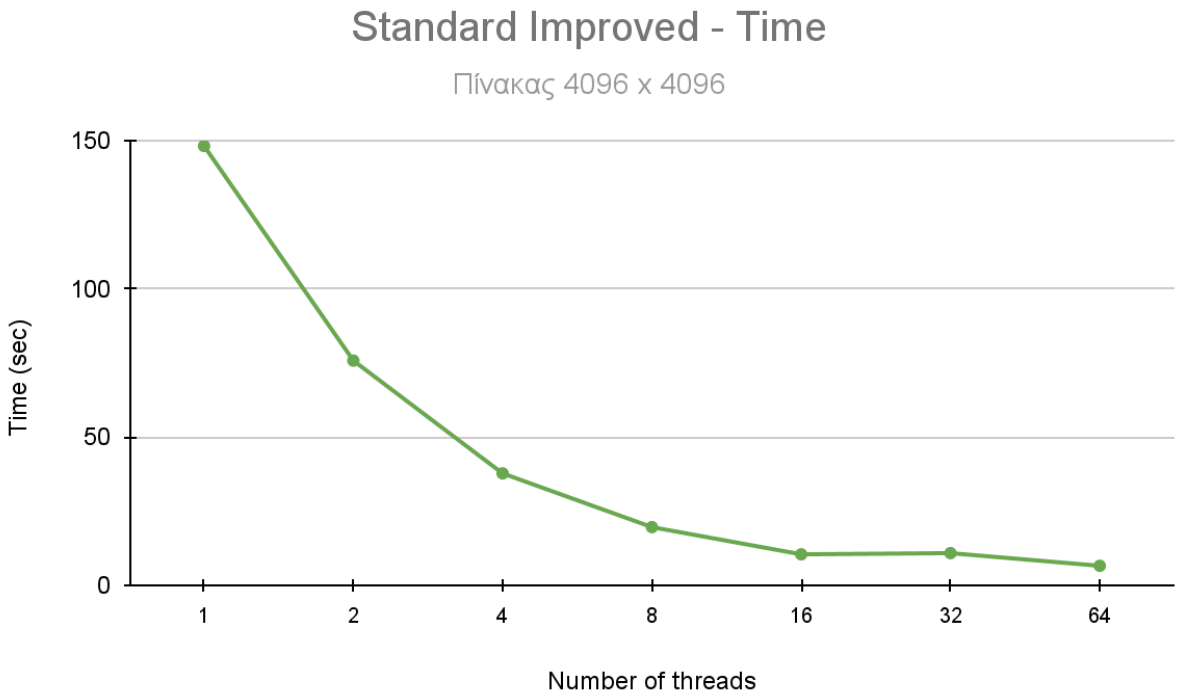

Standard Improved - Speedup
Πίνακας 1024 X 1024
Speedup (S = Ts / Tp)
12
10
8
6
4
2
0
1
2
4
8
16
32
64
Number of threads

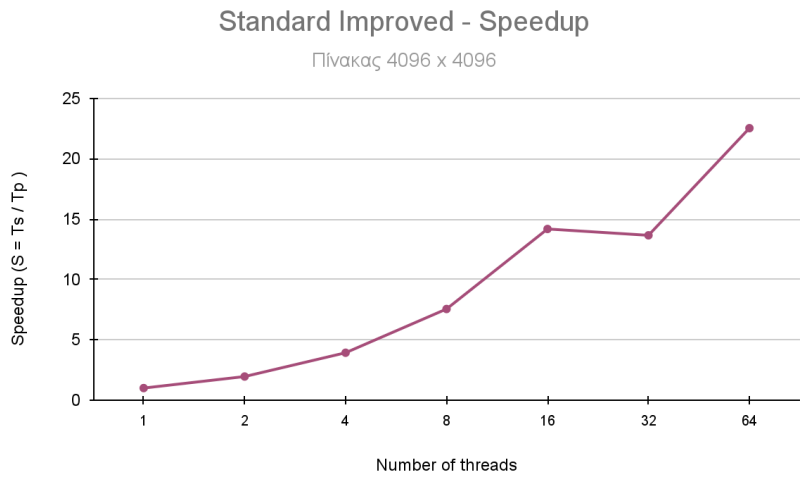

Standard Improved - Efficiency
Πίνακας 1024 X 1024
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads

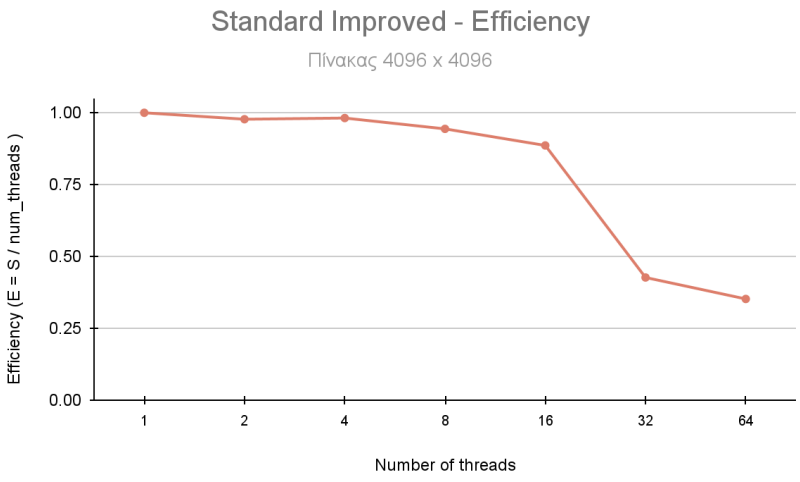

Standard Improved - Time
Πίνακας 2048 x 2048
Time (sec)
20
15
10
5
0
1
2
4
8
16
32
64
Number of threads


Standard Improved - Speedup
Πίνακας 2048 x 2048
Speedup (S = Ts / Tp)
25
20
15
10
5
0
1
2
4
8
16
32
64
Number of threads


Standard Improved - Efficiency
Πίνακας 2048 x 2048
Efficiency (E = S / num_threads)
1.25
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads


Standard Improved - Time
Πίνακας 4096 x 4096
150
100
50
0
Time (sec)
1
2
4
8
16
32
64
Number of threads


Standard Improved - Speedup
Πίνακας 4096 x 4096
25
20
15
10
5
0
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Standard Improved - Efficiency
Πίνακας 4096 x 4096
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Παρατηρούμε περαιτέρω βελτίωση των χρόνων εκτέλεσης, το οποίο συνεπάγεται επίσης καλύτερα speedup. Η παραλληλοποίηση της αρχικοποίησης του πίνακα παρατηρούμε ακόμα ότι βελτιώνει σημαντικά το efficiency, το οποίο διατηρείται σε υψηλά επίπεδα για περισσότερα threads σε σχέση με την αρχική έκδοση, ιδιαίτερα για τα μεγαλύτερα μεγέθη πίνακα.

Οι βελτιώσεις αυτές οφείλονται στην παραλληλοποίηση της αρχικοποίησης του πίνακα, με την οποία αφενός μειώνεται το αρχικό overhead της αρχικοποίησης, αφού αυτή μπορεί να μοιραστεί σε περισσότερους πυρήνες, και αφετέρου βοηθάει στην ισοκατανομή του φόρτου στους επεξεργαστές αυξάνοντας τον ωφέλιμο χρόνο του καθενός, γεγονός το οποίο αντανακλάται στην βελτίωση του efficiency.

Ελάχιστοι χρόνοι που επιτύχαμε:

Μέγεθος Πίνακα N	**Αριθμός Threads**	**Χρόνος (sec)**
1024 X 1024	32	0.1993
2048 X 2048	64	0.8697
4096 X 4096	64	6.5746

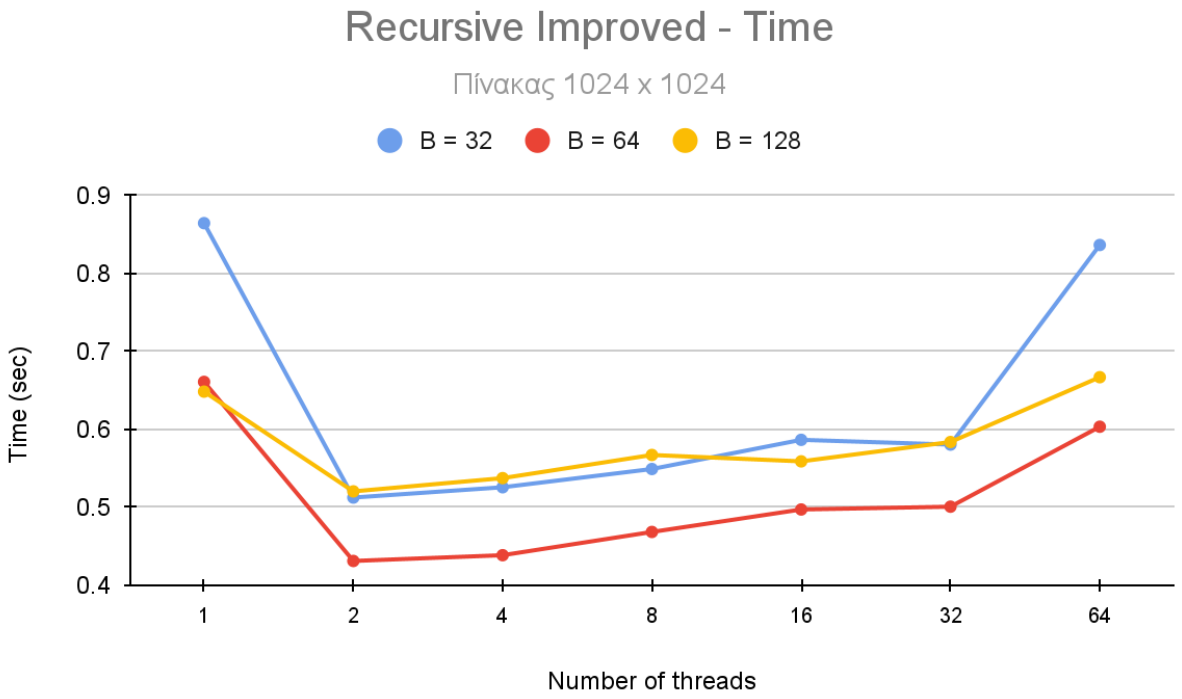
Recursive αλγόριθμος Floyd-Warshall

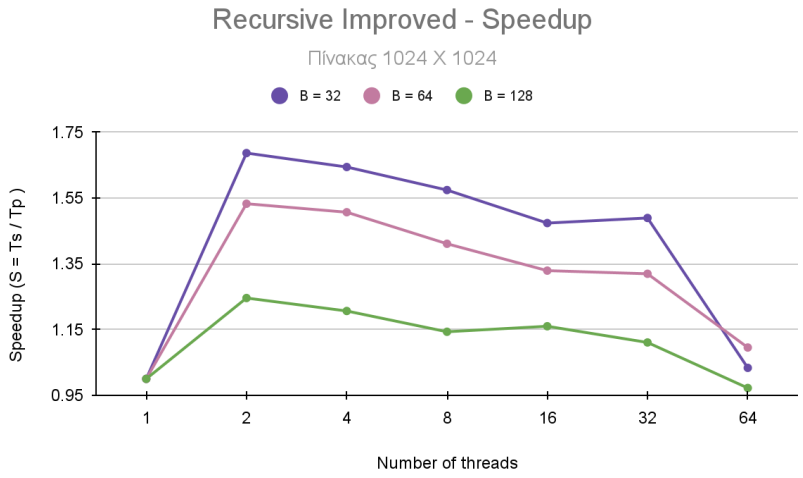
Για τον recursive αλγόριθμο Floyd-Warshall ο χρόνος εκτέλεσης βελτιώθηκε ορίζοντας 2 μικρότερες παράλληλες περιοχές, καθεμία από τις οποίες περιέχει μόνο μία από τις δύο δυάδες αναδρομών που μπορουν να εκτελεστούν παράλληλα. Στις παράλληλες αυτές περιοχές ορίζεται 1 αντί για 2 ξεχωριστά tasks, αφού η δεύτερη αναδρομή θα εκτελεστεί από το single thread (parent του task) χωρίς κίνδυνο το thread αυτό να προχωρήσει σε επόμενες αναδρομές. Ο κίνδυνος αυτός εδώ δεν υπάρχει γιατί στο τέλος της παράλληλης περιοχής θα γίνει συγχρονισμός των threads, άρα θα πρέπει να εκτελεστούν όλες οι διεργασίες και των 2 threads για να γίνει ο συγχρονισμός και να βγουν από την παράλληλη περιοχή. Επίσης για να εξασφαλιστεί ότι όντως κανένα thread δεν θα προχωρήσει πριν τελειώσουν και τα δύο τους υπολογισμούς τους προστίθεται και ένα taskwait στο τέλος της παράλληλης περιοχής. Επομένως ο βελτιωμένος κώδικας έχει την ακόλουθη μορφή που φαίνεται στην επόμενη σελίδα.

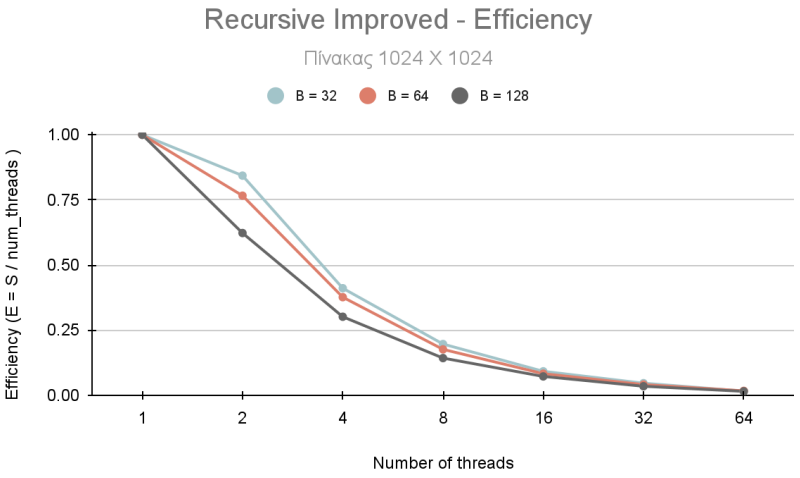
```
else {
         FW_SR(A,arow, acol,B,brow, bcol,C,crow, ccol, myN/2, bsize);
         #pragma omp parallel
         {
            #pragma omp single
            {
                #pragma omp task shared(A, B, C)
                     FW_SR(A,arow, acol+myN/2,B,brow, bcol,C,crow, ccol+myN/2, myN/2, bsize);
                FW_SR(A,arow+myN/2, acol,B,brow+myN/2, bcol,C,crow, ccol, myN/2, bsize);
                #pragma omp taskwait
            }
        }
        FW_SR(A,arow+myN/2, acol+myN/2,B,brow+myN/2, bcol,C,crow, ccol+myN/2, myN/2, bsize);

             FW_SR(A,arow+myN/2,  acol+myN/2,B,brow+myN/2,  bcol+myN/2,C,crow+myN/2,  ccol+myN/2,  myN/2,
bsize);
        #pragma omp parallel
        {
            #pragma omp single
            {
                #pragma omp task shared(A, B, C)
                          FW_SR(A,arow+myN/2,  acol,B,brow+myN/2,  bcol+myN/2,C,crow+myN/2,  ccol,  myN/2,
bsize);
                FW_SR(A,arow, acol+myN/2,B,brow, bcol+myN/2,C,crow+myN/2, ccol+myN/2, myN/2, bsize);
                #pragma omp taskwait
            }
        }
        FW_SR(A,arow, acol,B,brow, bcol+myN/2,C,crow+myN/2, ccol, myN/2, bsize);
}
```

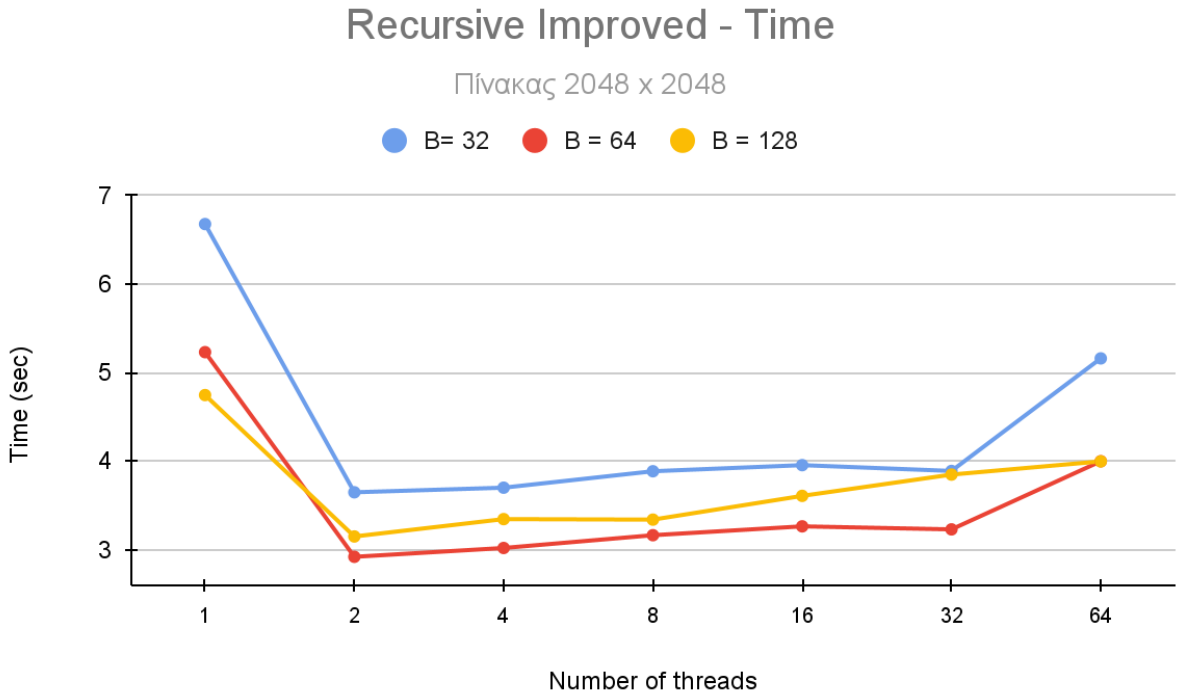
Σημειώνεται ότι χρησιμοποιούνται τα ίδια αρχεία *make_on_queue_recursive_p.sh* και *run_on_queue_recursive_p.sh* που παρουσιάστηκαν στην αρχική έκδοση.

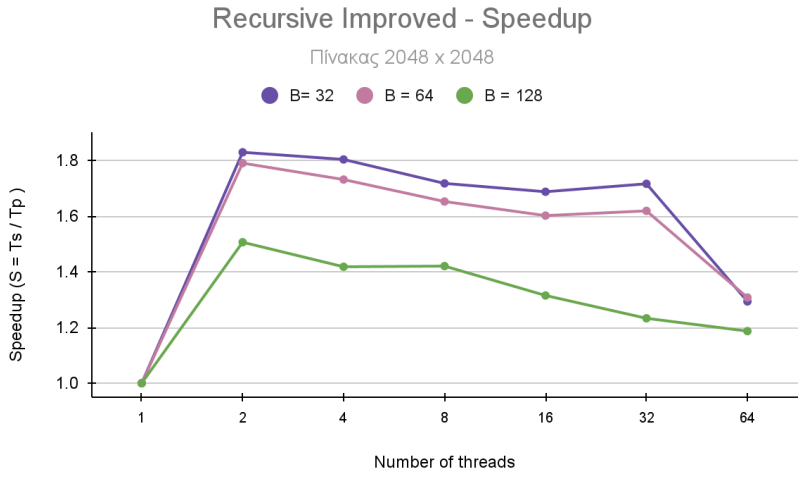
Οι χρόνοι που λαμβάνονται από την εκτέλεση της παραπάνω βελτιωμένης έκδοσης του παραλληλοποιημένου κώδικα του recursive FW αλγορίθμου και οι πληροφορίες που αφορούν τα speedup και efficiency που προκύπτουν από τους χρόνους αυτούς παρουσιάζονται στα ακόλουθα διαγράμματα.

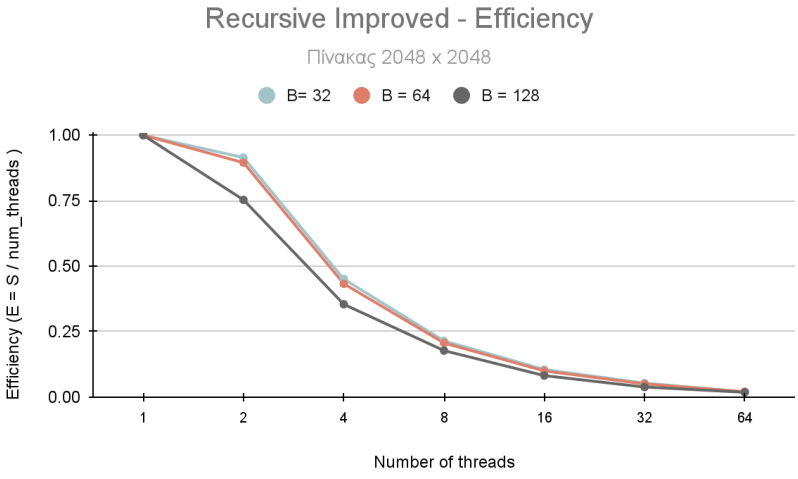

Recursive Improved - Time
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Time (sec)
0.9
0.8
0.7
0.6
0.5
0.4
1
2
4
8
16
32
64
Number of threads

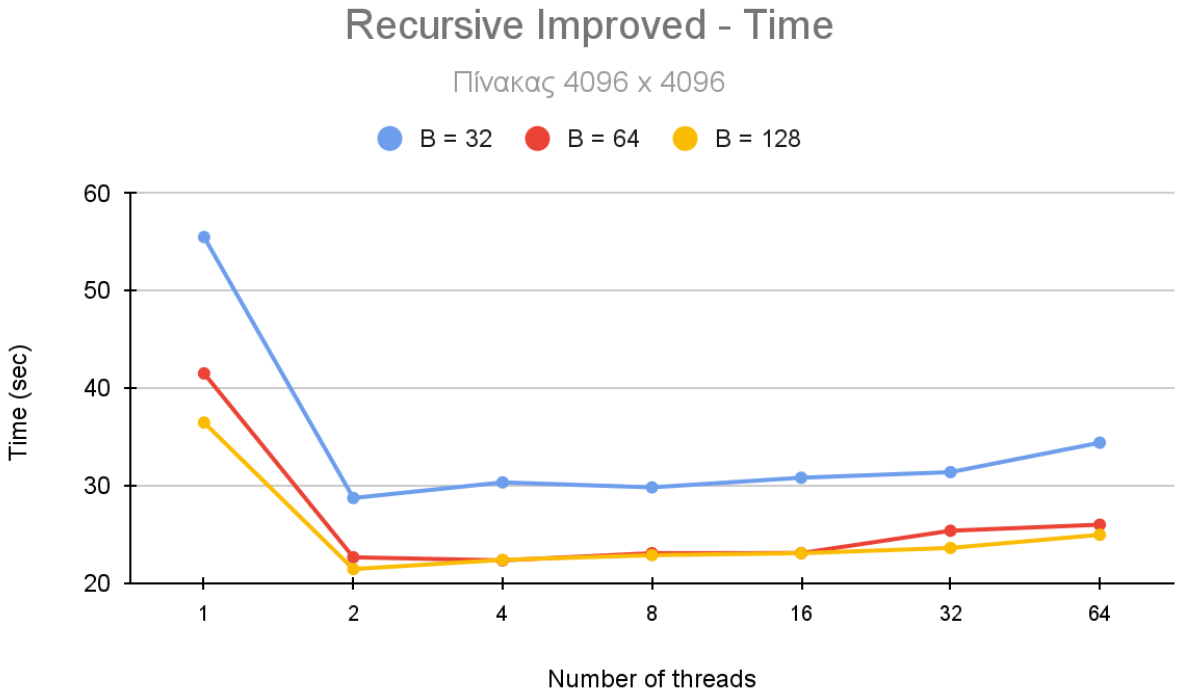

Recursive Improved - Speedup
Πίνακας 1024 X 1024
B = 32
B = 64
B = 128
Speedup (S = Ts / Tp)
1.75
1.55
1.35
1.15
0.95
1
2
4
8
16
32
64
Number of threads

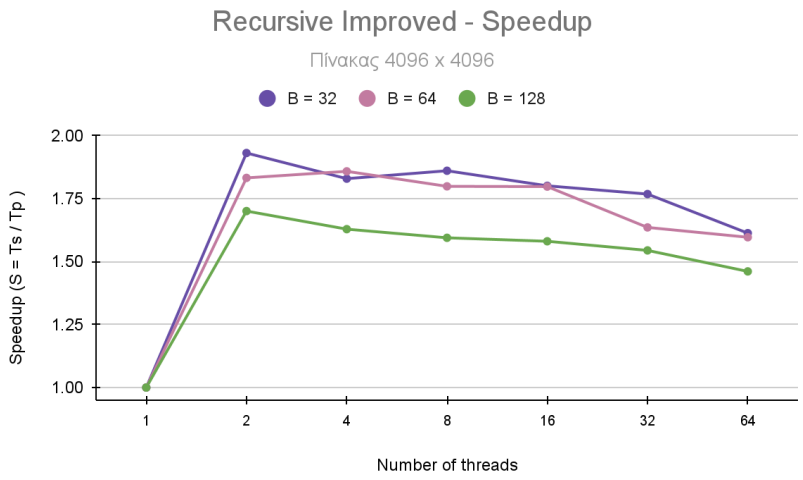

Recursive Improved - Efficiency
Πίνακας 1024 X 1024
B = 32
B = 64
B = 128
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads

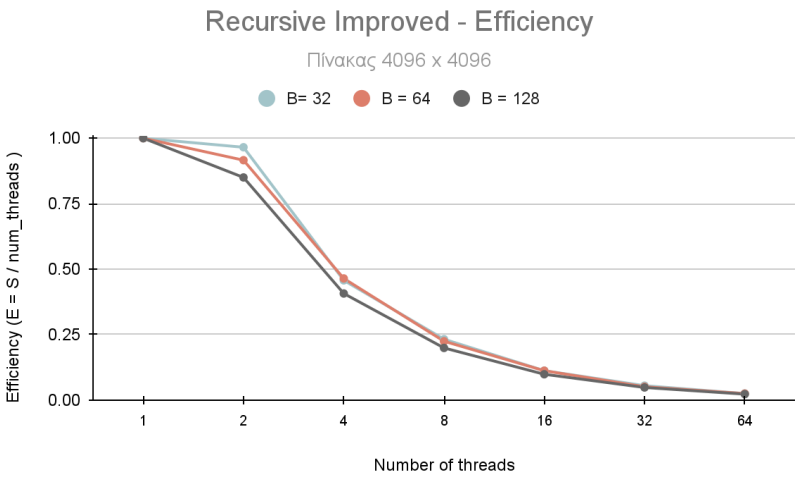

Recursive Improved - Time
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Time (sec)
7
6
5
4
3
1
2
4
8
16
32
64
Number of threads


Recursive Improved - Speedup
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Speedup (S = Ts / Tp)
1.8
1.6
1.4
1.2
1.0
1
2
4
8
16
32
64
Number of threads


Recursive Improved - Efficiency
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads


Recursive Improved - Time
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
60
50
40
30
20
Time (sec)
1
2
4
8
16
32
64
Number of threads


Recursive Improved - Speedup
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
2.00
1.75
1.50
1.25
1.00
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Recursive Improved - Efficiency
Πίνακας 4096 x 4096
B= 32
B = 64
B = 128
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Παρατηρείται ότι, για όλα τα μεγέθη πίνακα, σε σχέση με την αρχική έκδοση ο χρόνος μειώνεται πολύ περισσότερο για την μετάβαση από το 1 στα 2 threads, με αποτέλεσμα να έχουμε επίσης καλύτερη τιμή speedup και efficiency για τιμή thread ίση με 2. Επίσης σε όλα τα μεγέθη πίνακα, σε αυτή την βελτιωμένη έκδοση πολύ πιο αισθητά και στον πίνακα 1024, για threads μεταξύ των 4 και 32 ο χρόνος εκτέλεσης παραμένει σχετικά σταθερός με μια μικρή τάση αύξησης, με αποτέλεσμα το speedup να παραμένει επίσης σχετικά σταθερό με μια μικρη τάση μείωσης, άρα το efficiency να μειώνεται σημαντικά και σε αυτή την έκδοση. Γενικά, όπως εξηγήθηκε αναλυτικότερα στην αρχική έκδοση, η συμπεριφορά αυτή είναι αναμενόμενη, μιας και λόγω της φύσης του αλγορίθμου εν τέλει χρησιμοποιούνται μόνο 2 threads.

Η βελτίωση στην απόδοση προφανώς οφείλεται στο γεγονός ότι χρησιμοποιείται 1 task αντί για 2, επομένως θα αρχικοποιηθεί ένα μόνο extra thread πέραν του parent. Για αυτό γλιτώνουμε το overhead που δημιουργείται με την αρχικοποίηση, την ανάθεση task, την επικοινωνία και τον συγχρονισμό του extra thread που δεν χρησιμοποιήθηκε στην βελτιωμένη έκδοση, για αυτό και παρατηρείται βελτίωση του χρόνου.

Ελάχιστοι χρόνοι που επιτύχαμε:

Μέγεθος Πίνακα N	Αριθμός Threads	Μέγεθος base case B	Χρόνος (sec)
1024 X 1024	2	64	0.4312
2048 X 2048	2	64	2.9215
4096 X 4096	2	128	21.4591

Tiled αλγόριθμος Floyd-Warshall

Για τον αλγόριθμο tiled Floyd-Warshall καταφέραμε να κάνουμε βελτίωση της παραλληλοποίησης και να λάβουμε τελικά καλύτερους χρόνους με τη χρήση binding το οποίο επιτεύχθηκε με την αλλαγή της μεταβλητής περιβάλλοντος OMP_PROC_BIND.

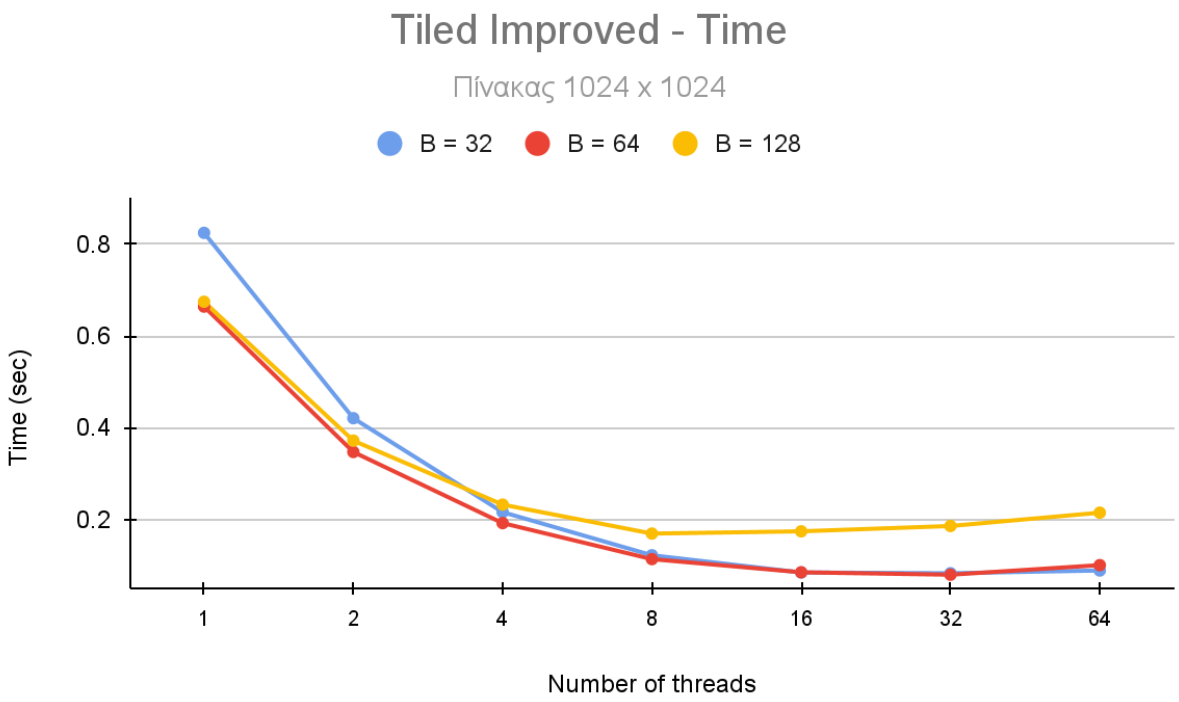
Με τη χρήση του binding, αναθέτουμε ένα συγκεκριμένο thread σε συγκεκριμένο επεξεργαστή και μπορούμε να πετύχουμε καλύτερο affinity και να εκμεταλλευτούμε καλύτερα την τοπικότητα των δεδομένων. Η αλλαγή της μεταβλητής περιβάλλοντος γίνεται με την παρακάτω εντολή, η οποία προστίθεται στον κώδικα του αρχείου *run_on_queue_tiled_p.sh,* το οποίο παρουσιάστηκε παραπάνω.

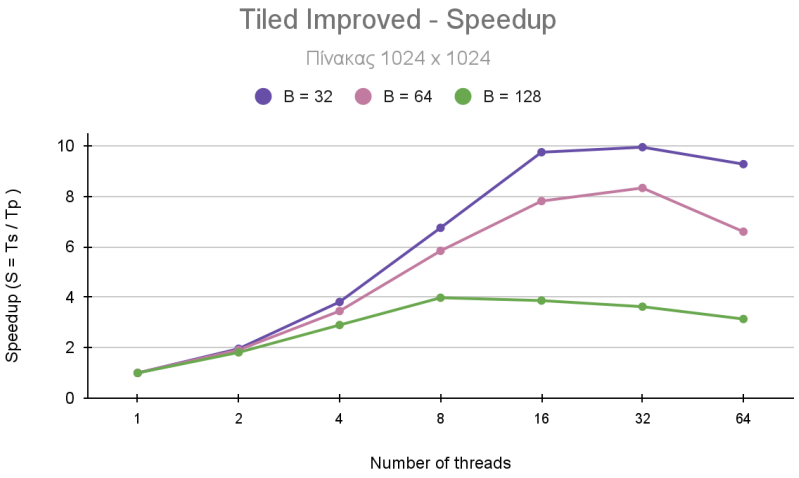
```
export OMP_PROC_BIND=true
```

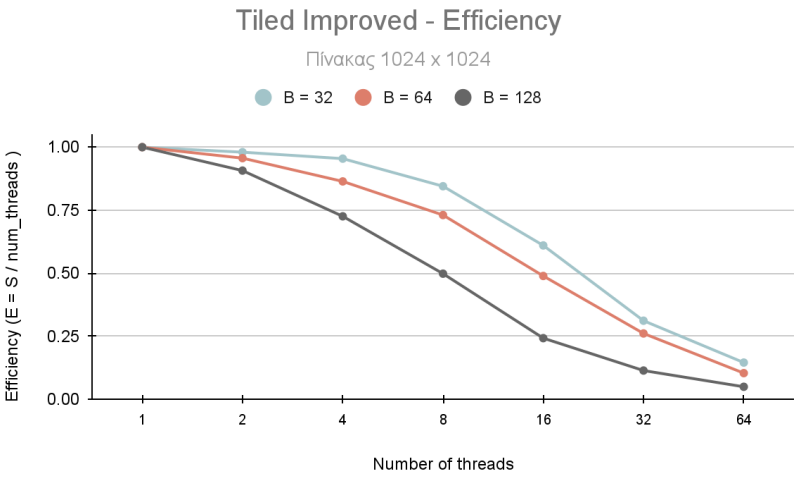
Σημειώνεται ότι χρησιμοποιήθηκε το ίδιο αρχείο *make_on_queue_tiled_p.sh* που παρουσιάστηκε στην αρχική έκδοση, ενώ το νέο αρχείο *run_on_queue_tiled_p.sh* που περιλαμβάνει την παραπάνω αλλαγή φαίνεται παρακάτω:

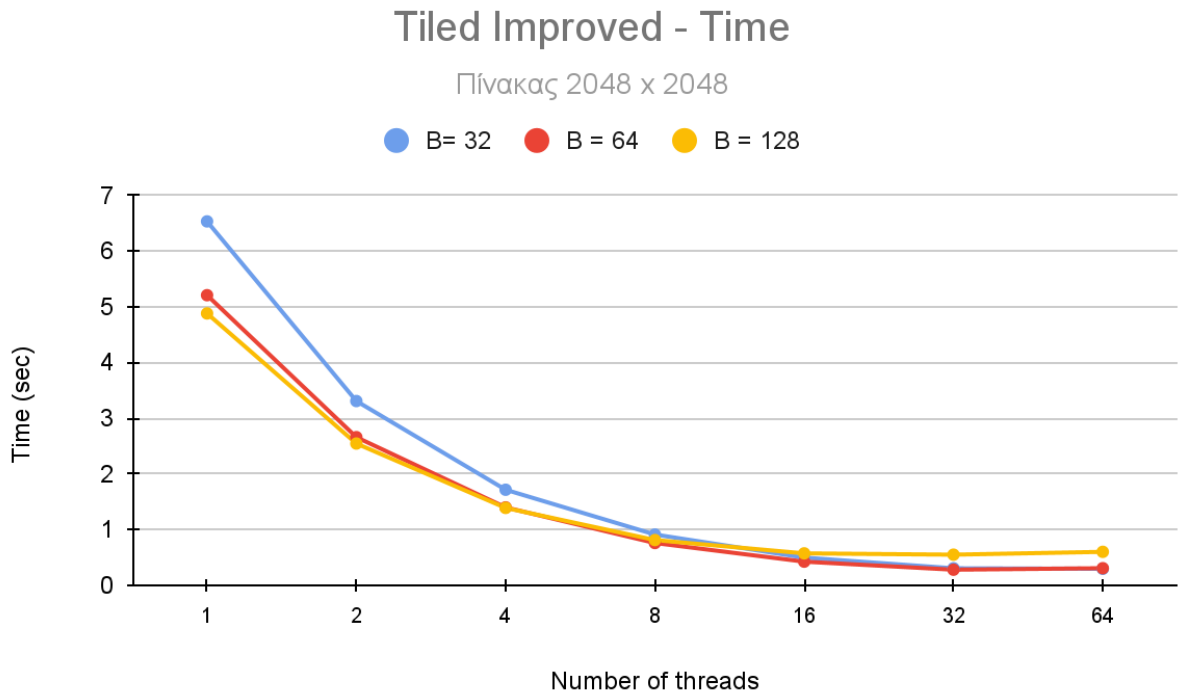
```
#!/bin/bash

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

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

##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/parlab20/a2/FW-parallel

for thr in 1 2 4 8 16 32 64
  do
   export OMP_NUM_THREADS=$thr
   export OMP_PROC_BIND=true
   ./fw_tiled_parallel 1024 32
   ./fw_tiled_parallel 2048 32
   ./fw_tiled_parallel 4096 32
   ./fw_tiled_parallel 1024 64
   ./fw_tiled_parallel 2048 64
   ./fw_tiled_parallel 4096 64
   ./fw_tiled_parallel 1024 128
   ./fw_tiled_parallel 2048 128
   ./fw_tiled_parallel 4096 128
  done
```

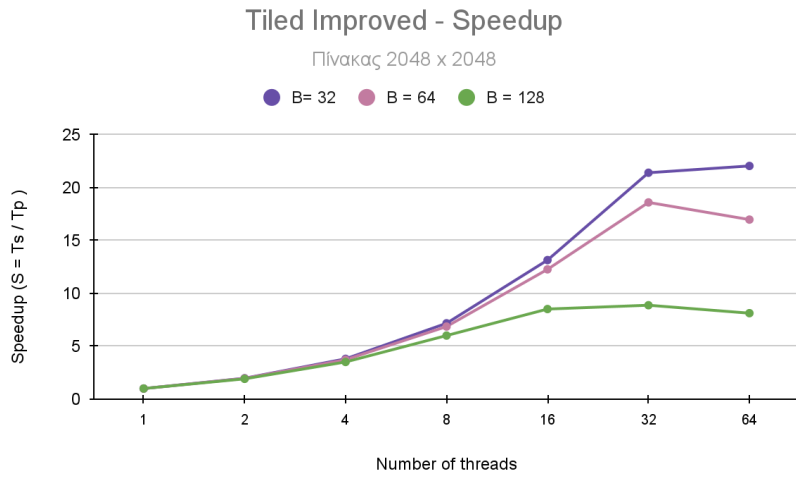
Σε αυτή την περίπτωση τα διαγράμματα χρόνου που προκύπτουν από την εκτέλεση του βελτιωμένου tiled αλγόριθμου και τα αντίστοιχα speedup και efficiency είναι τα ακόλουθα.

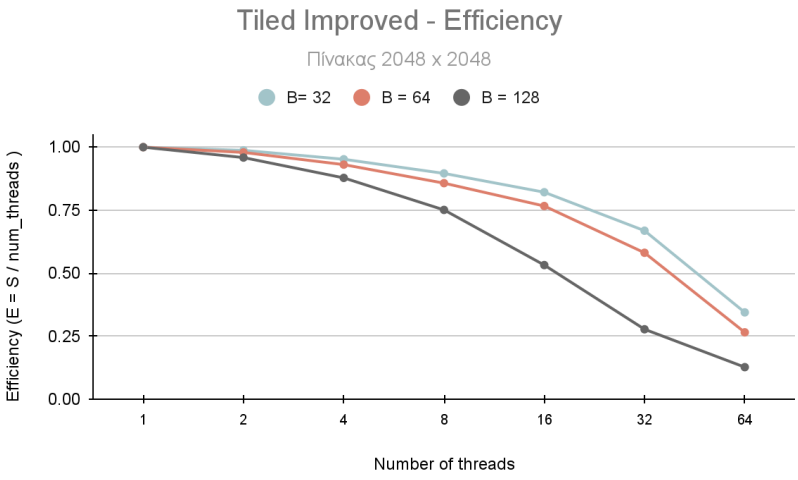

Tiled Improved - Time
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Time (sec)
0.8
0.6
0.4
0.2
1
2
4
8
16
32
64
Number of threads

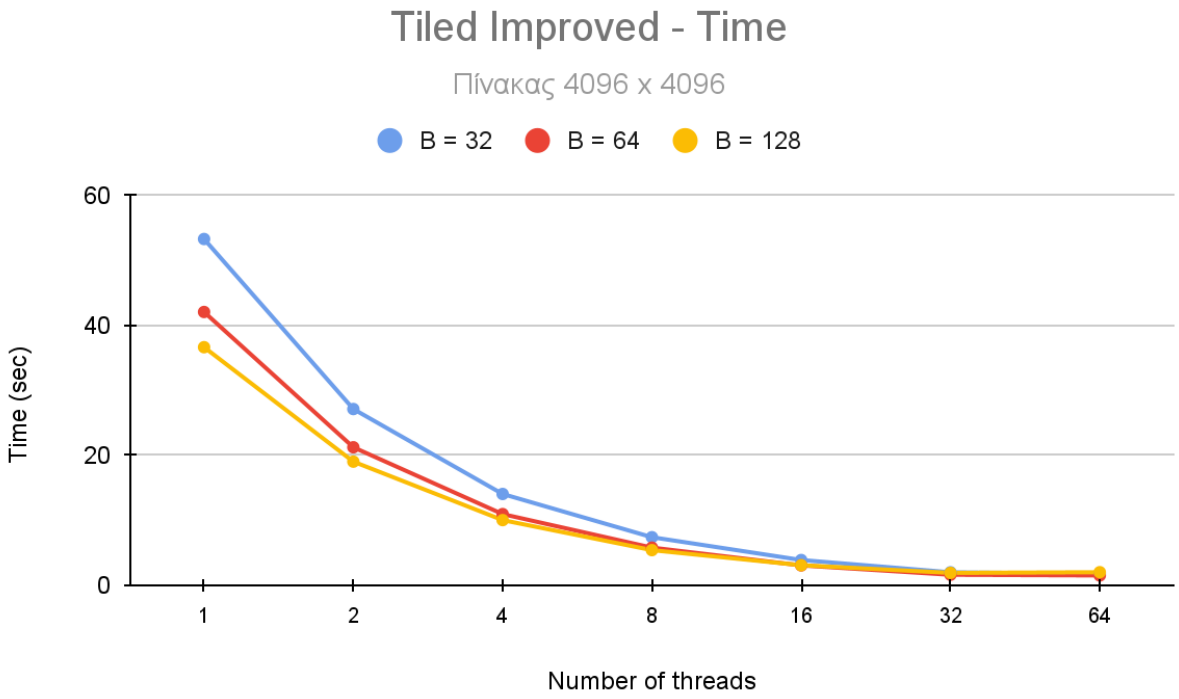

Tiled Improved - Speedup
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Speedup (S = Ts / Tp)
10
8
6
4
2
0
1
2
4
8
16
32
64
Number of threads

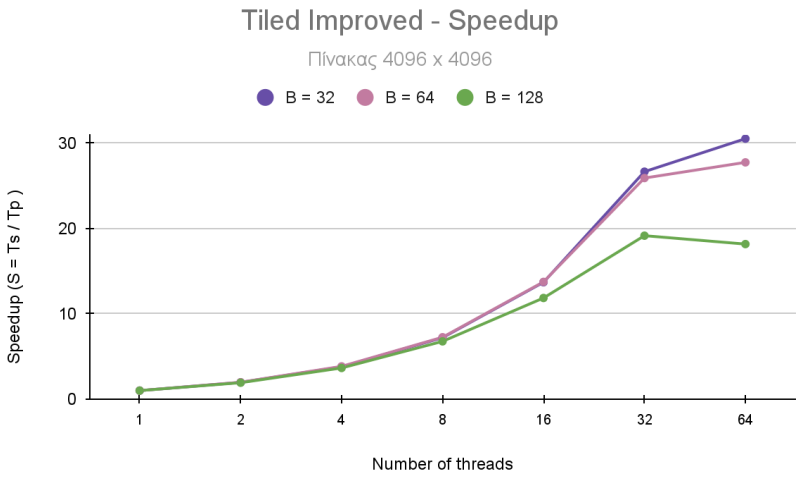

Tiled Improved - Efficiency
Πίνακας 1024 x 1024
B = 32
B = 64
B = 128
Efficiency (E = S / num_threads)
1.00
0.75
0.50
0.25
0.00
1
2
4
8
16
32
64
Number of threads

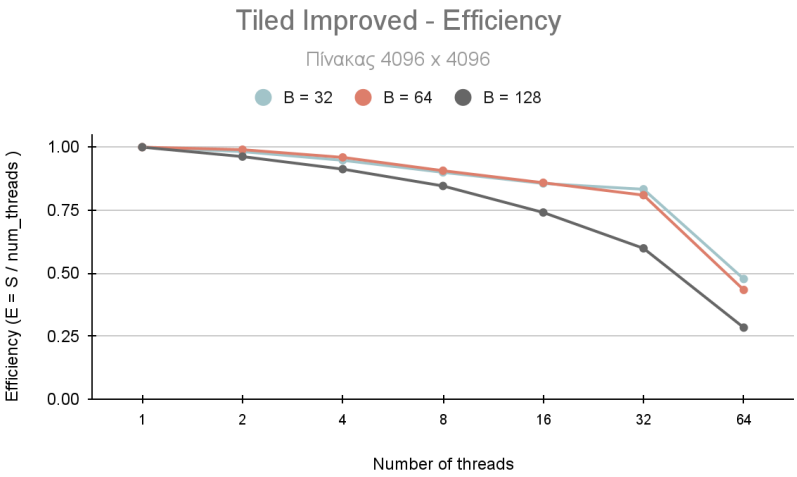

Tiled Improved - Time
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Time (sec)
0
1
2
3
4
5
6
7
1
2
4
8
16
32
64
Number of threads


Tiled Improved - Speedup
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Speedup (S = Ts / Tp)
0
5
10
15
20
25
1
2
4
8
16
32
64
Number of threads


Tiled Improved - Efficiency
Πίνακας 2048 x 2048
B= 32
B = 64
B = 128
Efficiency (E = S / num_threads)
0.00
0.25
0.50
0.75
1.00
1
2
4
8
16
32
64
Number of threads


Tiled Improved - Time
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
60
40
20
0
Time (sec)
1
2
4
8
16
32
64
Number of threads


Tiled Improved - Speedup
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
30
20
10
0
Speedup (S = Ts / Tp)
1
2
4
8
16
32
64
Number of threads


Tiled Improved - Efficiency
Πίνακας 4096 x 4096
B = 32
B = 64
B = 128
1.00
0.75
0.50
0.25
0.00
Efficiency (E = S / num_threads)
1
2
4
8
16
32
64
Number of threads

Σχολιασμός και Παρατηρήσεις:

Με την αλλαγή της τιμής της μεταβλητής περιβάλλοντος OMP_PROC_BIND σε true, επιτεύχθηκε περαιτέρω βελτίωση των χρόνων εκτέλεσης, η οποία γίνεται ιδιαίτερα αισθητή στις περιπτώσεις που χρησιμοποιούνται πολλά threads. Αυτό αντικατοπτρίζεται και στο speedup που ενώ στην αρχική έκδοση έπασχε και είχε έντονη πτωτική τάση από τα 32 threads (με εξαίρεση την περίπτωση του πίνακα 4096), πλέον έχει σταθερότερη ανοδική τάση και τείνει να παρουσιάσει πτώση στην περίπτωση των 64 threads.

Η σημαντικότερη αλλαγή όμως, έγκειται στη βελτίωση του efficiency, ειδικά στην περίπτωση του πίνακα 4096 όπου διατηρείται σε πολύ καλύτερα επίπεδα από ότι στην πρώτη έκδοση μέχρι και τα 32 threads. Η βελτίωση αυτή οφείλεται βεβαίως στο affinity το οποίο πετυχαίνουμε με το binding, αφού έχουμε ανάθεση του κάθε thread σε συγκεκριμένο επεξεργαστή και πολύ καλύτερη εκμετάλλευση της τοπικότητας, οπότε αυξάνεται ο ωφέλιμος χρόνος για κάθε πυρήνα.

Ελάχιστοι χρόνοι που επιτύχαμε:

Μέγεθος Πίνακα N	Αριθμός Threads	Μέγεθος Tile B	Χρόνος (sec)
1024 X 1024	32	64	0.0796
2048 X 2048	32	64	0.2801
4096 X 4096	64	64	1.5156