21/11/2019

Συστήματα Παράλληλης Επεξεργασίας

Σειρά 1η - Τελική Αναφορά

Ομάδα: parlab30
Τσαγκαράκης Στυλιανός - 03115180
Τσάκας Νικόλαος - 03115433

1. Εξαρτήσεις Δεδομένων

Ξεκινώντας να προσεγγίσουμε την παραλληλοποίηση του αλγορίθμου FW στις διαφορετικές υλοποιήσεις που δίνονται, εντοπίσαμε ορισμένες ομοιότητες μεταξύ τους. Ο αλγόριθμος δρα και στις 3 υλοποιήσεις με μία παρόμοια σειρά εξαρτήσεων εκτέλεσης καθώς ενημερώνονται διαφορετικά σημεία του πίνακα, όπως παρουσιάζεται στις διαφάνειες για το εργαστήριο.

Αρχικά, στην κλασική εκδοχή είναι υποχρεωτική η ολοκλήρωση της k επανάληψης πριν την εκτέλεση της k+1, το οποίο υποννοεί μία 'διαγώνιο' κίνηση του αλγορίθμου μέσα στον πίνακα. Έπειτα στην αναδρομική εκδοχή η εκτέλεση των αναδρομικών κλήσεων σε κάθε τεταρτημόριο ακολουθεί τη σειρά εξαρτήσεων A11 → {A12, A21} → A22 → A22 → {A12, A21} → A11 που παρουσιάζει μία όμοια εικόνα. Ακόμη, η τρίτη εκδοχή παρουσιάζει εξάρτηση τόσο ανάμεσα στα k (που αντιπροσωπεύουν πλέον tiles μεγέθους B) όσο και κατά τη διάρκεια της εκτέλεσης για κάθε k: Προτεραιότητα έχει η αρχική κλήση του αλγορίθμου FW, μετά την εκτέλεση της οποίας έπονται τα 4 for loops για την ενημέρωση των 'αξόνων' από tiles πάνω, κάτω και εκατέρωθεν του (k,k). Τέλος ενημερώνονται τα ορθογώνια τμήματα ανάμεσα στους 'άξονες' χρησιμοποιώντας πληροφορία από αυτούς.

2. Η παράμετρος B

Οι δύο νέες υλοποιήσεις χρησιμοποιούν και οι δύο μία παράμετρο B δεδομένη από το χρήστη, η οποία αντιπροσωπεύει το μέγεθος του υποπίνακα στο οποίο ξεκινά να εκτελείται τοπικά ο κλασικός FW. Παρατηρήσαμε μετά από δοκιμές σε διαφορετικά μηχανήματα και στο sandman ότι το μέγεθος της παραμέτρου B που οδηγεί σε βέλτιστα χρονικά αποτελέσματα παραμένει περίπου σταθερό σε κάθε μηχάνημα ανεξαρτήτως υλοποίησης παραλληλίας και μεγέθους πίνακα εισόδου. Το αποτέλεσμα αυτό είναι αναμενόμενο, αφού το βέλτιστο B εξαρτάται κατά κόρον από το μέγεθος της κρυφής μνήμης που διαθέτει κάθε thread.

Όταν το B είναι πολύ μικρό, ο υποπίνακας χωράει ολόκληρος στην κρυφή μνήμη που διατίθεται σε κάθε thread αλλά εκτελούνται υπερβολικά πολλές αναδρομικές κλήσεις / χωρίζεται ο πίνακας σε πολλά tiles. Αυτό έχει ως αποτέλεσμα να αναλώνονται οι πόροι του συστήματος σε δημιουργία threads στις περισσότερες παράλληλες εκδοχές μας. Αντίθετα όταν είναι υπερβολικά μεγάλο δεν χωράει στην κρυφή μνήμη κάθε thread, οπότε και αναλώνεται το σύστημα σε memory fetches.

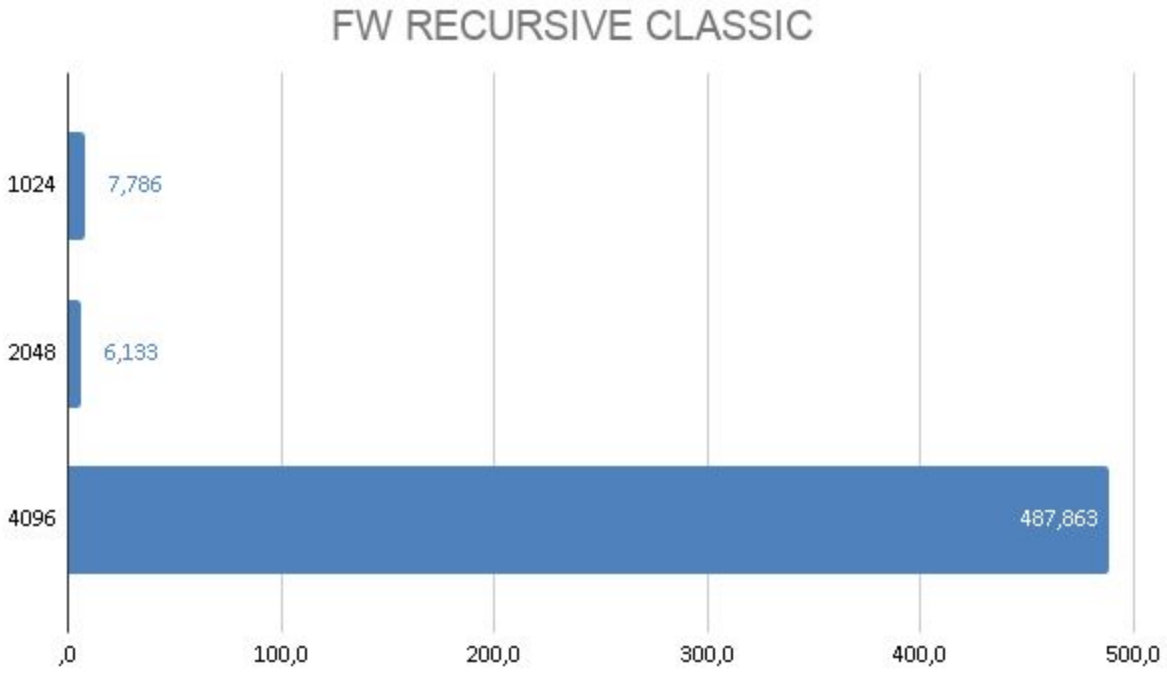
3. OpenMP

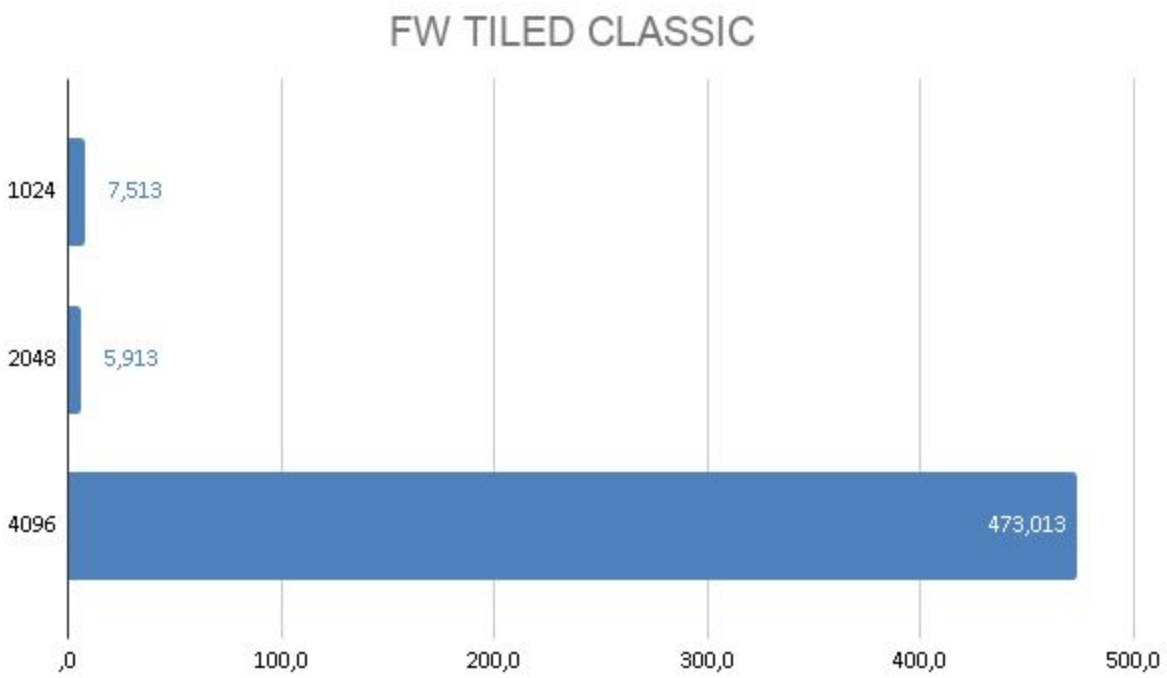
Για την παραλληλοποίηση των διαφορετικών εκδοχών του αλγορίθμου αποφασίσαμε να χρησιμοποιήσουμε OpenMP λόγω της απλότητας του, θυσιάζοντας κάποια ευελιξία έναντι της χρήσης TBB. Βάση των παραπάνω παρατηρήσεων κάναμε ορισμένες 'ενημερωμένες'

πειραματικές δοκιμές χρησιμοποιώντας τα constructs omp for, omp sections και omp task/taskwait.

4. Αποτελέσματα κλασσικών/σειριακών αλγόριθμων

Παρακάτω παρατίθενται οι μετρήσεις για τους σειριακούς αλγόριθμους.

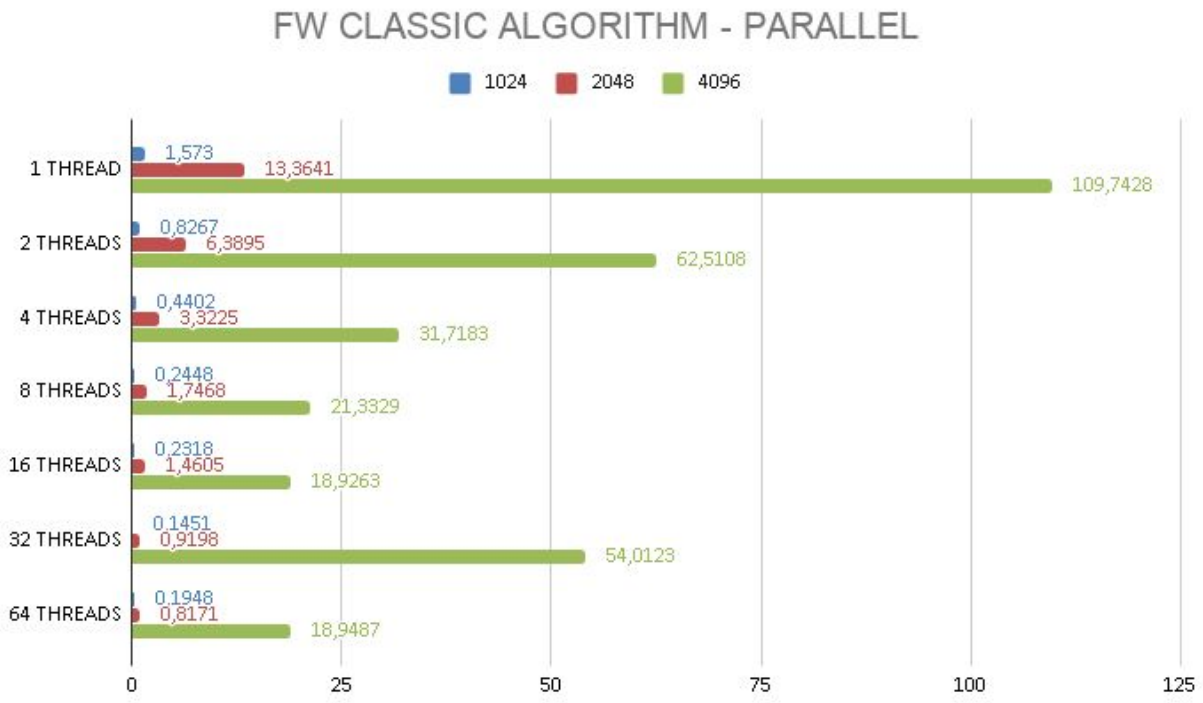

FW RECURSIVE CLASSIC
1024
7,786
2048
6,133
4096
487,863
,0
100,0
200,0
300,0
400,0
500,0


FW TILED CLASSIC
1024
7,513
2048
5,913
4096
473,013
,0
100,0
200,0
300,0
400,0
500,0

5. Παραλληλοποίηση του κλασικού Floyd-Warshall

Για τον κλασσικό αλγόριθμο ο κώδικας εγγενώς δεν προσφέρεται για σημαντική βελτιστοποίηση σε παράλληλα συστήματα, οπότε η καλύτερη προσέγγιση μας ήταν η παραλληλοποίηση των υπολογισμών σε κάθε θέση του πίνακα ανά βήμα του μετρητή k:

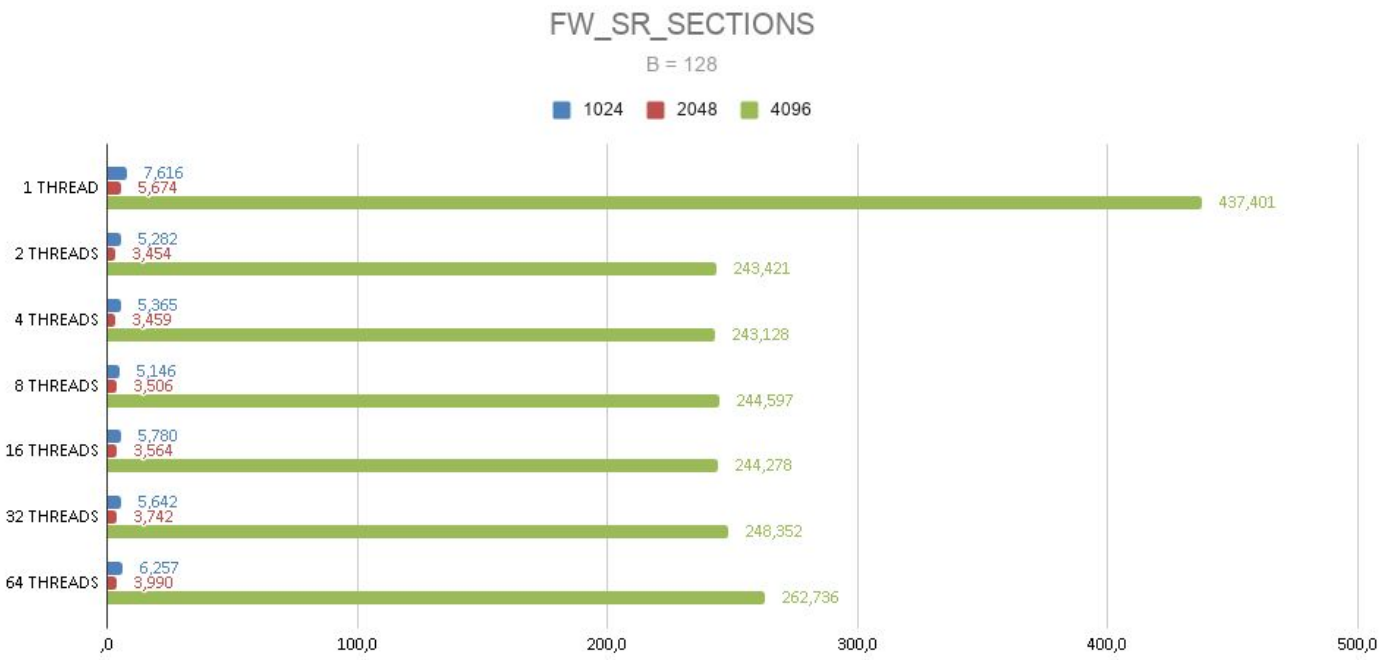
```
for(k=0;k<N;k++)
    #pragma omp parallel for shared(A) private(i,j) firstprivate(k,
N)
    for(i=0; i<N; i++)
        for(j=0; j<N; j++)
            A[i][j]=min(A[i][j], A[i][k] + A[k][j]);
```

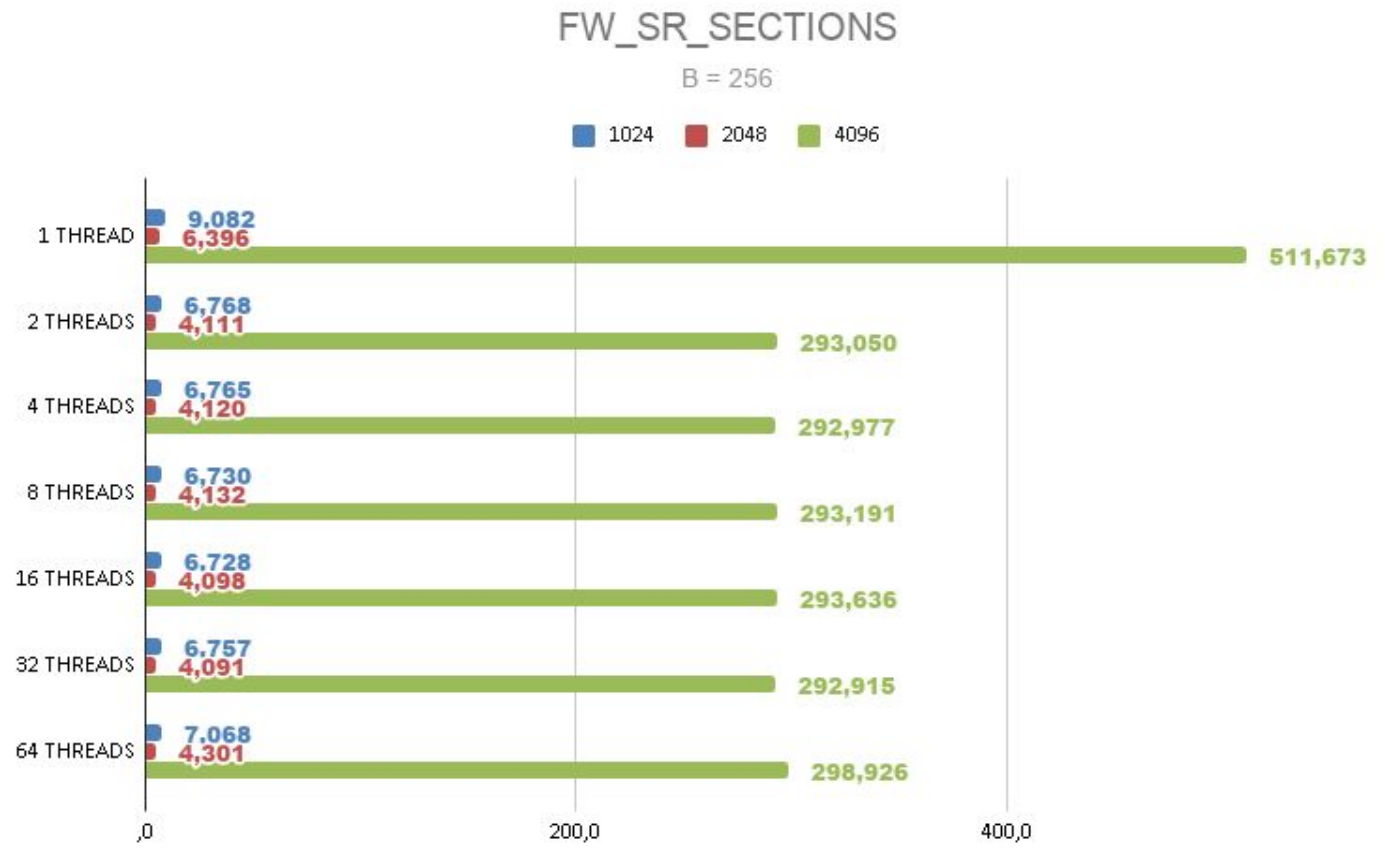


6.. Παραλληλοποίηση της Recursive Υλοποίησης

Για τη recursive υλοποίηση παρατηρούμε ότι η μόνη περιοχή που είναι παραλληλοποιήσιμη σε κάθε αναδρομική κλήση είναι η ενημέρωση των περιοχών A12, A21 καθώς δεν υπάρχει εξάρτηση μεταξύ τους. Αρχικά επιχειρήσαμε τη χρήση του omp sections construct για την παραλληλοποίηση αυτών των περιοχών:

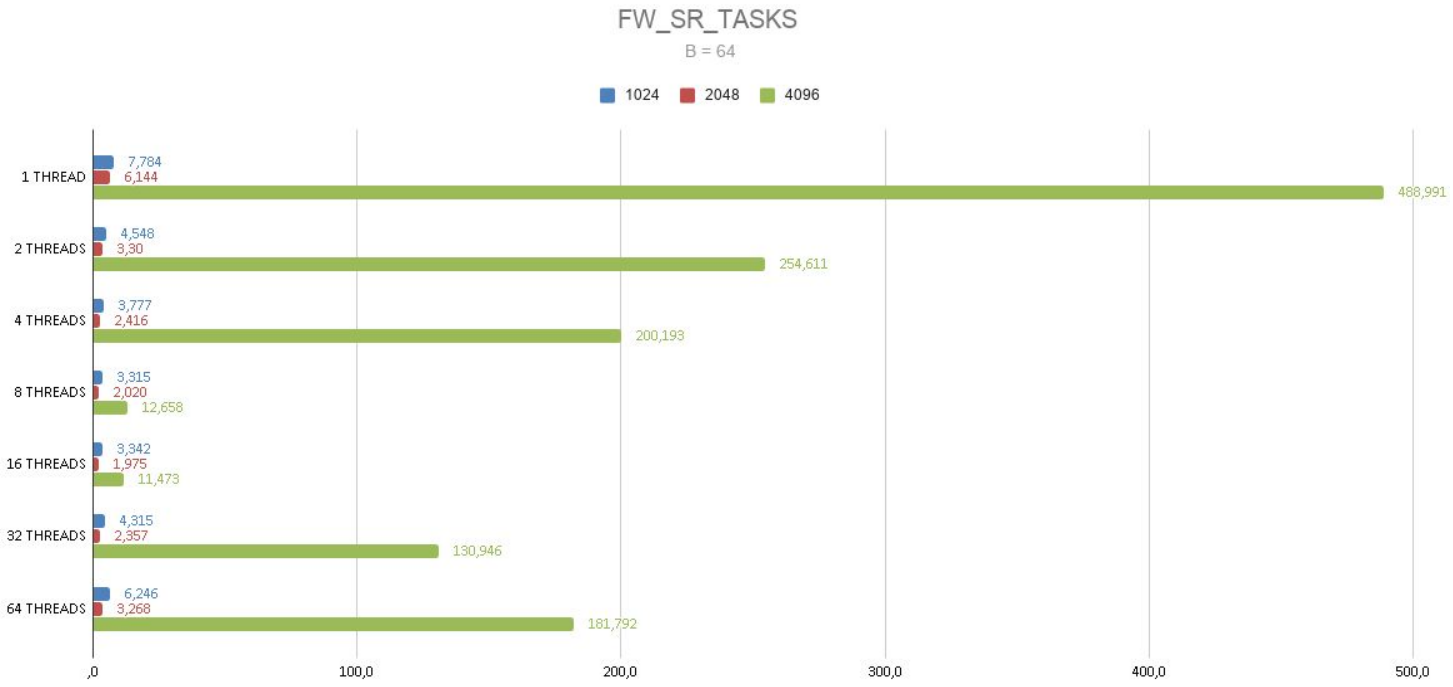
```
FWR (A11, B11, C11);
#pragma omp sections
{
      #pragma omp section
      FWR (A12, B11, C12);

      #pragma omp section
      FWR (A21, B21, C11);
}
FWR (A22, B21, C12);
FWR (A22, B21, C12);
#pragma omp sections
{
      #pragma omp section
      FWR (A21, B21, C11);

      #pragma omp section
      FWR (A12, B11, C12);
}
FWR (A11, B11, C11);
```

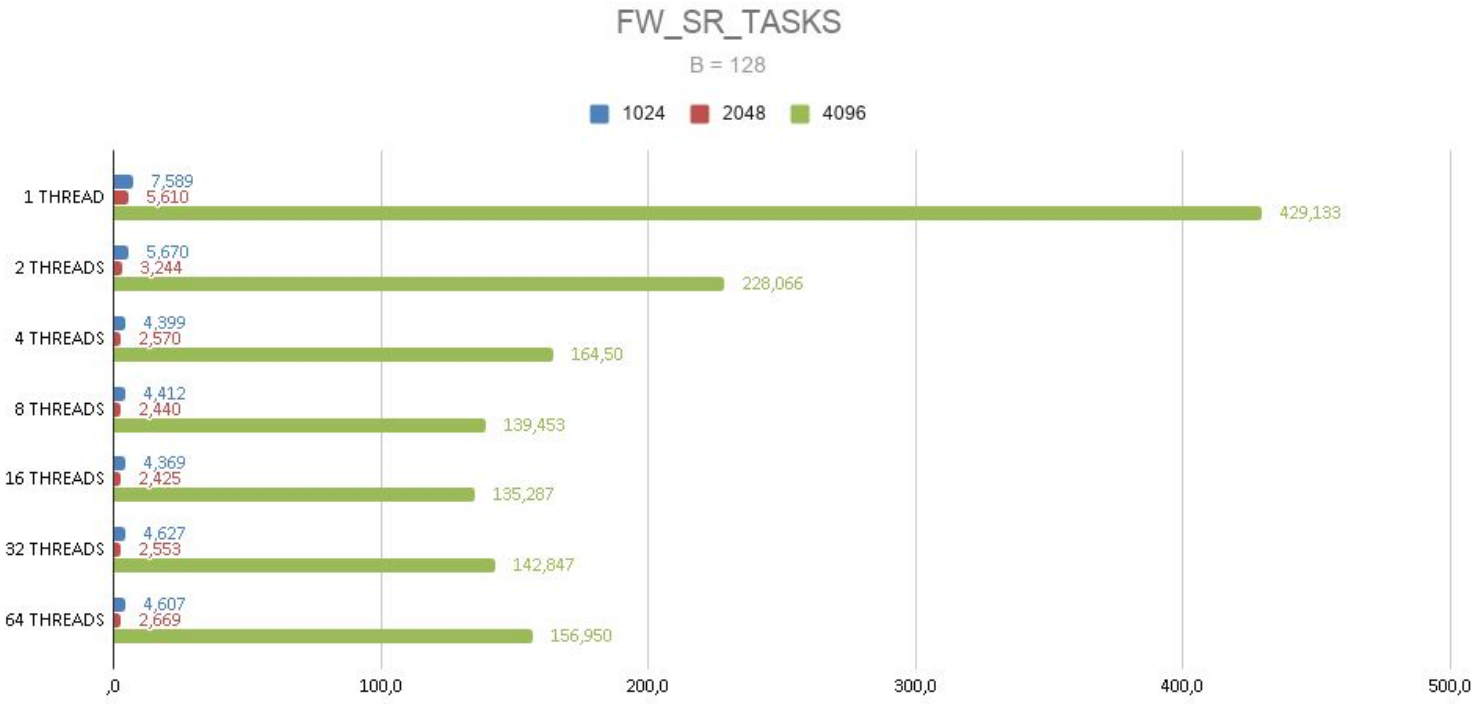

FW_SR_SECTIONS
B = 128
1024
2048
4096
1 THREAD
2 THREADS
4 THREADS
8 THREADS
16 THREADS
32 THREADS
64 THREADS
7,616
5,674
437,401
5,282
3,454
243,421
5,365
3,459
243,128
5,146
3,506
244,597
5,780
3,564
244,278
5,642
3,742
248,352
6,257
3,990
262,736
,0
100,0
200,0
300,0
400,0
500,0

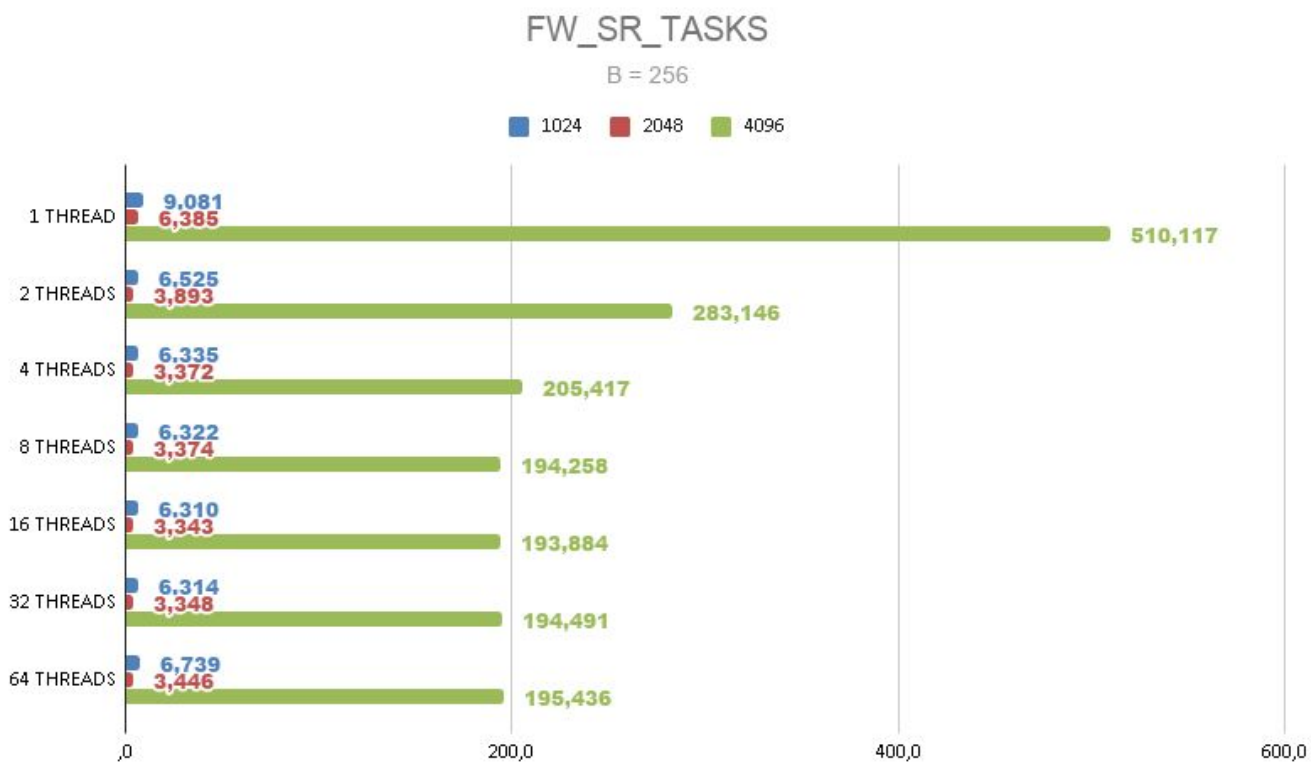

FW_SR_SECTIONS
B = 256
1024
2048
4096
1 THREAD
2 THREADS
4 THREADS
8 THREADS
16 THREADS
32 THREADS
64 THREADS
9,082
6,396
511,673
6,768
4,111
293,050
6,765
4,120
292,977
6,730
4,132
293,191
6,728
4,098
293,636
6,757
4,091
292,915
7,068
4,301
298,926
,0
200,0
400,0

Έπειτα χρησιμοποίησαμε omp tasks τα οποία όντας πιο lightweight δομές επιτρέπουν καλύτερη οργάνωση των πόρων του συστήματος κατά την δημιουργία threads:

```
FWR (A11, B11, C11);
#pragma omp taskwait
#pragma omp task
FWR (A12, B11, C12);
#pragma omp task
FWR (A21, B21, C11);
#pragma omp taskwait
FWR (A22, B21, C12);
FWR (A22, B21, C12);
#pragma omp taskwait
#pragma omp task
FWR (A21, B21, C11);
#pragma omp task
FWR (A12, B11, C12);
#pragma omp taskwait
FWR (A11, B11, C11);
```




FW_SR_TASKS
B = 128
1024
2048
4096
1 THREAD
7,589
5,610
429,133
2 THREADS
5,670
3,244
228,066
4 THREADS
4,399
2,570
164,50
8 THREADS
4,412
2,440
139,453
16 THREADS
4,369
2,425
135,287
32 THREADS
4,627
2,553
142,847
64 THREADS
4,607
2,669
156,950
,0
100,0
200,0
300,0
400,0
500,0


FW_SR_TASKS
B = 256
1024
2048
4096
1 THREAD
9,081
6,385
510,117
2 THREADS
6,525
3,893
283,146
4 THREADS
6,335
3,372
205,417
8 THREADS
6,322
3,374
194,258
16 THREADS
6,310
3,343
193,884
32 THREADS
6,314
3,348
194,491
64 THREADS
6,739
3,446
195,436
,0
200,0
400,0
600,0

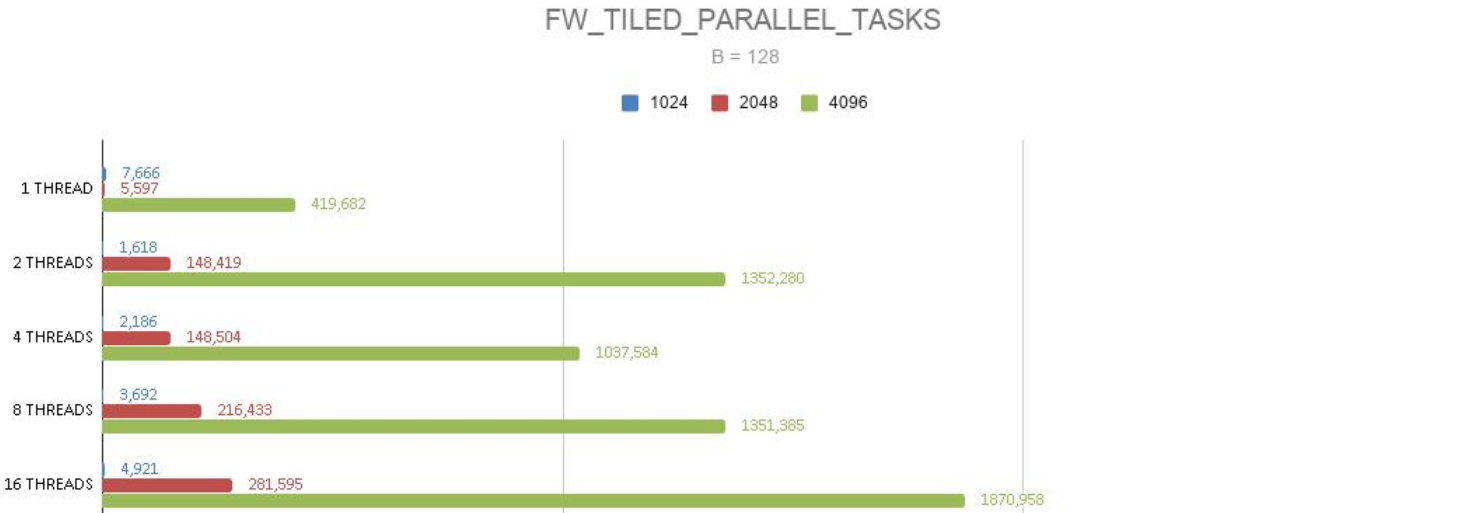
Σημ.: Πιο πολύπλοκες προσεγγίσεις, συμπεριλαμβανομένης και της παραλληλοποίησης του τμήματος που εφαρμόζεται ο κλασικός FW αποδείχθηκαν μη αποδοτικές.

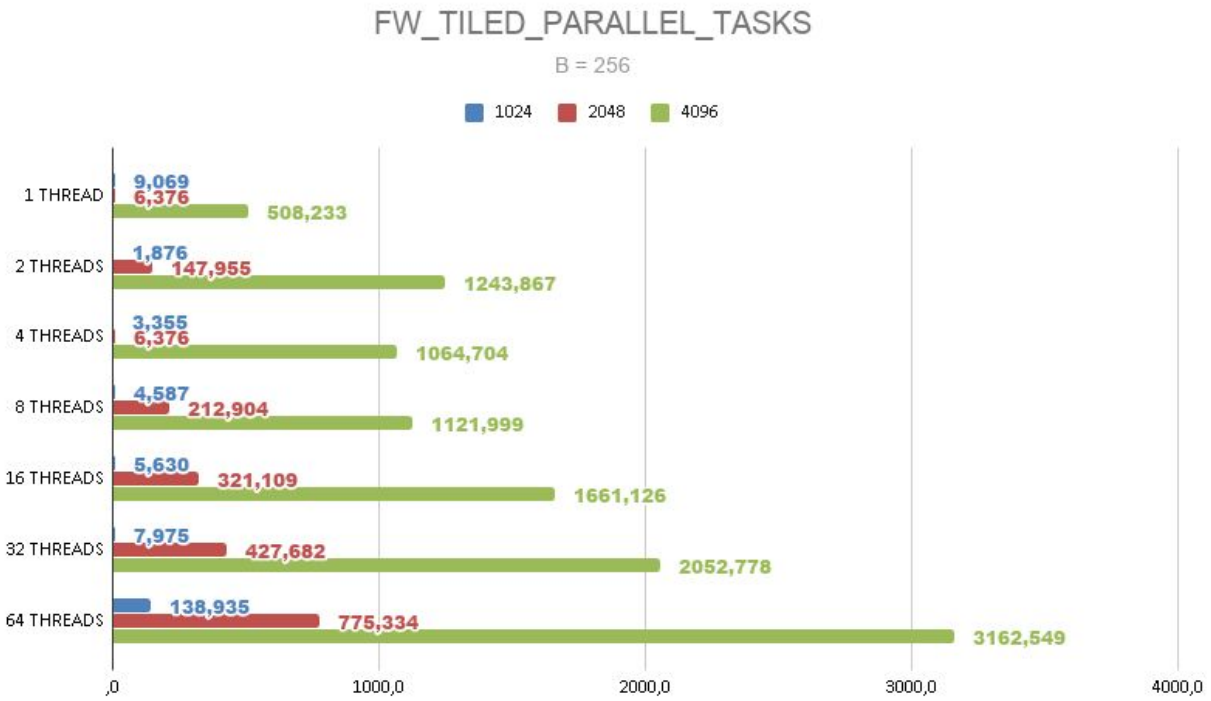
7. Παραλληλοποίηση της Tiled Υλοποίησης

Για την tiled εκδοχή του αλγορίθμου αρχικά μας φάνηκε προφανής μία προσέγγιση αντίστοιχη της recursive, δηλαδή task centric παραλληλοποίηση των independent τμημάτων του κώδικα:

```
for k = 1:N step B {
      FW(k,k)
      #pragma omp taskwait

      #pragma omp task
      for all tiles above
            apply FW
      #pragma omp task
      for all tiles below
            apply FW
      #pragma omp task
      for all tiles right
            apply FW
      #pragma omp task
      for all tiles left
            apply FW
      #pragma omp taskwait

      #pragma omp task
      for all tiles up right
            apply FW
      #pragma omp task
      for all tiles up left
            apply FW
      #pragma omp task
      for all tiles down right
            apply FW
      #pragma omp task
      for all tiles down left
            apply FW
      #pragma omp taskwait
}
```


FW_TILED_PARALLEL_TASKS
B = 64
1024
2048
4096
1 THREAD
2 THREADS
4 THREADS
8 THREADS
16 THREADS
32 THREADS
64 THREADS
7,518
5,938
474,830
2,396
270,768
2235,382
2,281
186,027
1573,859
3,698
298,016
2104,483
5,351
362,820
2422,795
6,351
417,310
2674,339
12,410
529,392
3037,291
,0
1000,0
2000,0
3000,0

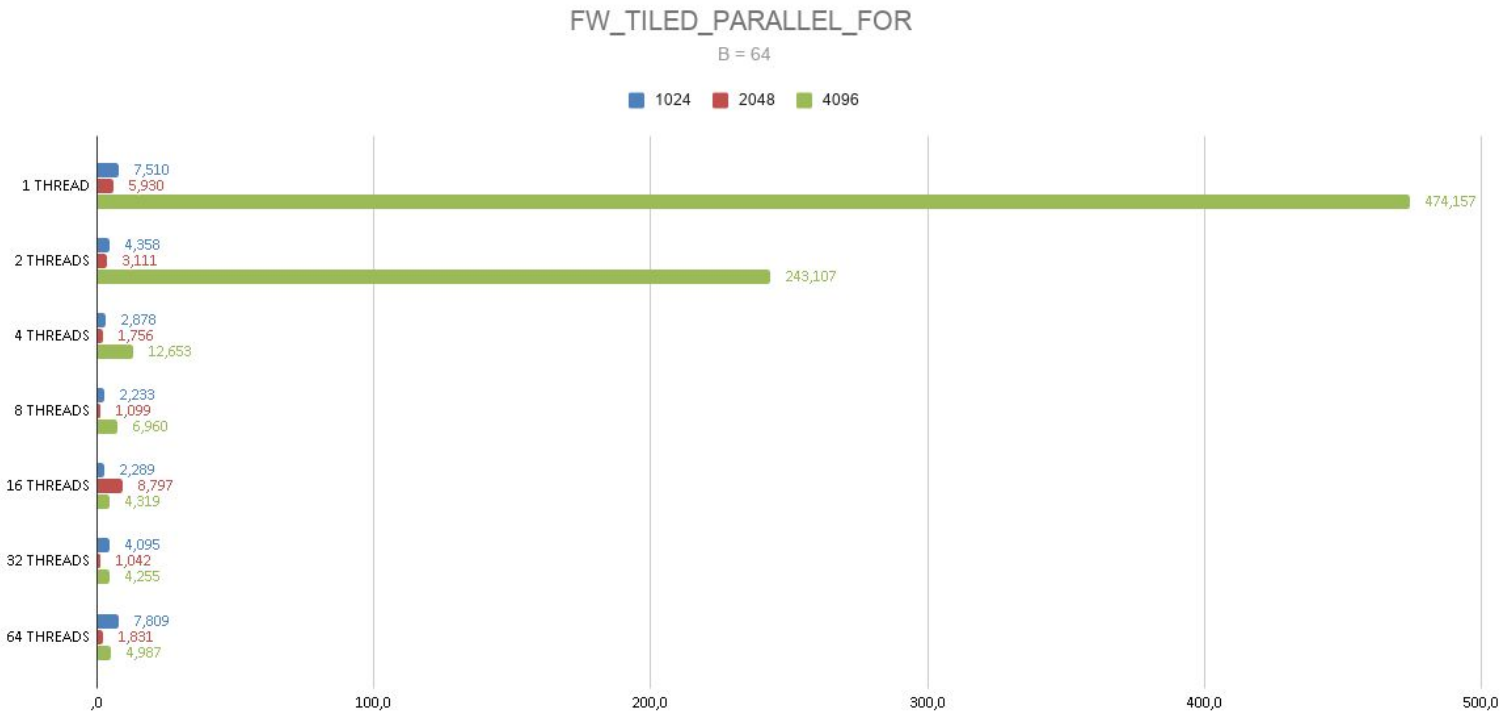

FW_TILED_PARALLEL_TASKS
B = 128
1024
2048
4096
1 THREAD
2 THREADS
4 THREADS
8 THREADS
16 THREADS
32 THREADS
64 THREADS
7,666
5,597
419,682
1,618
148,419
1352,280
2,186
148,504
1037,584
3,692
216,433
1351,385
4,921
281,595
1870,958
6,765
355,052
1985,225
12,682
607,128
2795,81
,0
1000,0
2000,0

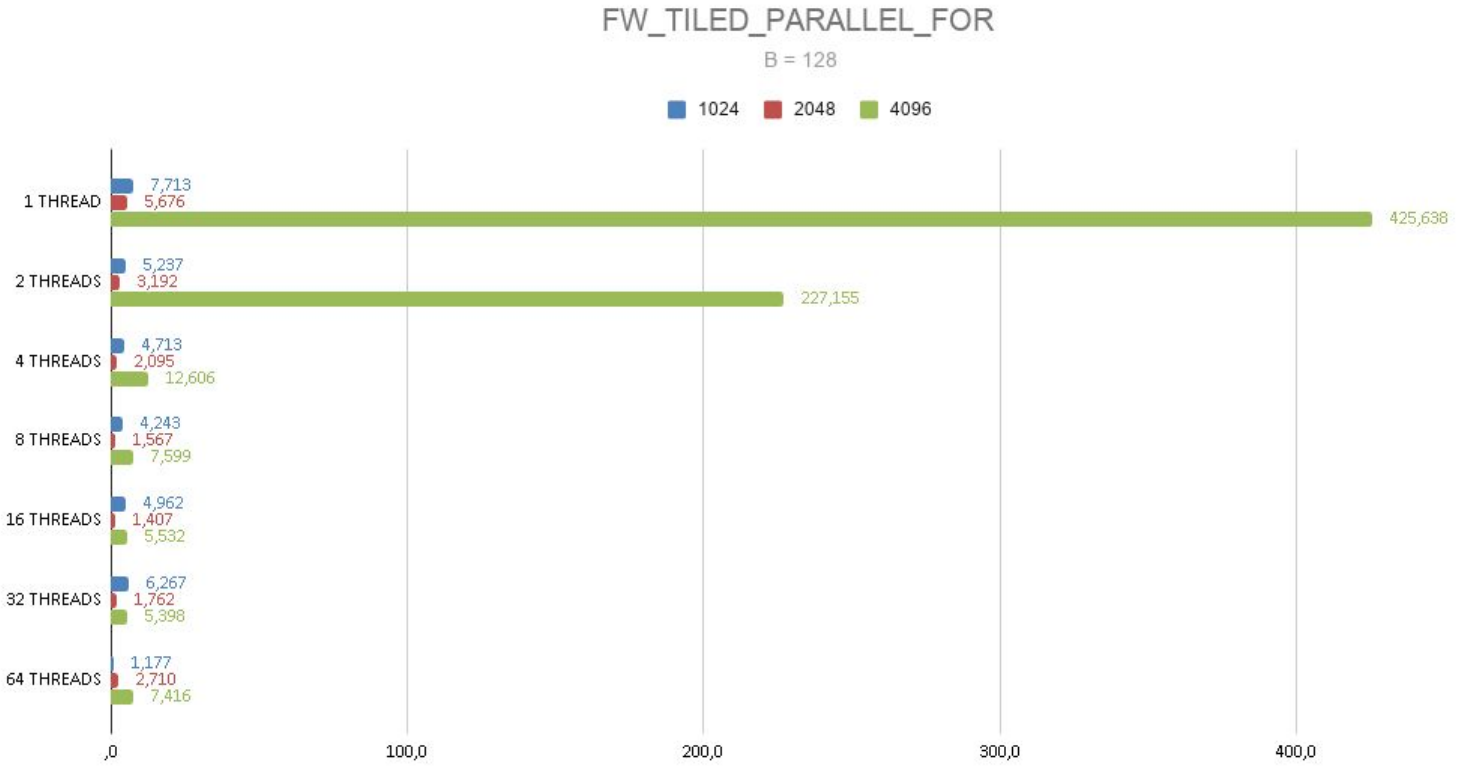


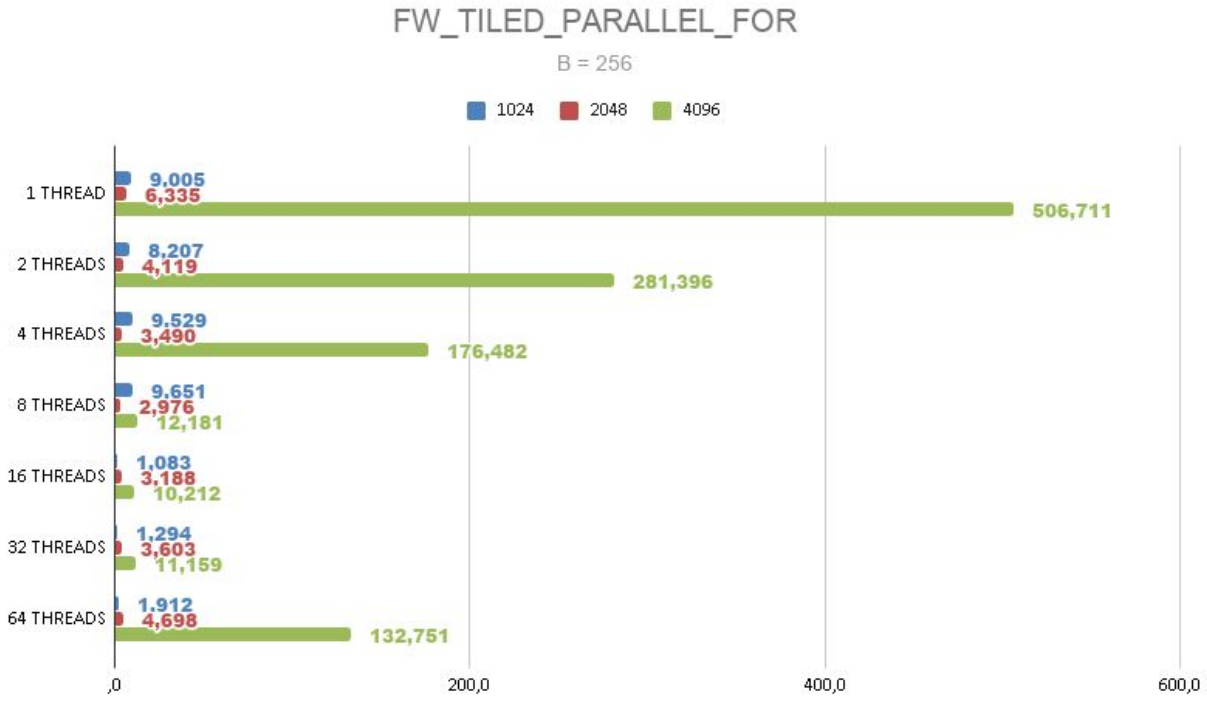
Ο κώδικας παρουσίασε σαφή βελτίωση με αυτή την προσέγγιση, παρ' αυτα η πιο αποδοτική στρατηγική αποδείχθηκε απλούστερη, συγκεκριμένα η εκτέλεση όλων των for loops παραλληλοποιόντας το καθένα με τη σειρα:

```
for k = 1:N step B {
      FW(k,k)
      #pragma omp for
      for all tiles above
            apply FW
      #pragma omp for
      for all tiles below
            apply FW
      #pragma omp for
      for all tiles right
            apply FW
      #pragma omp for
      for all tiles left
            apply FW

      #pragma omp for
      for all tiles up right
            apply FW
```

```
        #pragma omp for
        for all tiles up left
              apply FW
        #pragma omp for
        for all tiles down right
              apply FW
        #pragma omp for
        for all tiles down left
              apply FW
  }
```




FW_TILED_PARALLEL_FOR
B = 128
1024
2048
4096
1 THREAD
7,713
5,676
425,638
2 THREADS
5,237
3,192
227,155
4 THREADS
4,713
2,095
12,606
8 THREADS
4,243
1,567
7,599
16 THREADS
4,962
1,407
5,532
32 THREADS
6,267
1,762
5,398
64 THREADS
1,177
2,710
7,416
,0
100,0
200,0
300,0
400,0


FW_TILED_PARALLEL_FOR
B = 256
1024
2048
4096
1 THREAD
9,005
6,335
506,711
2 THREADS
8,207
4,119
281,396
4 THREADS
9,529
3,490
176,482
8 THREADS
9,651
2,976
12,181
16 THREADS
1,083
3,188
10,212
32 THREADS
1,294
3,603
11,159
64 THREADS
1,912
4,698
132,751
,0
200,0
400,0
600,0

Για άλλη μία φορά, πιο πολύπλοκες υλοποιήσεις, συμπεριλαμβανομένου και του υβριδικού μοντέλου (παράλληλα for και κατανομή σε tasks) δεν επέφεραν καλύτερη απόδοση.