4.1 Παραλληλοποίηση και βελτιστοποίηση αλγορίθμων σε αρχιτεκτονικές κατανεμημένης μνήμη

K-Means

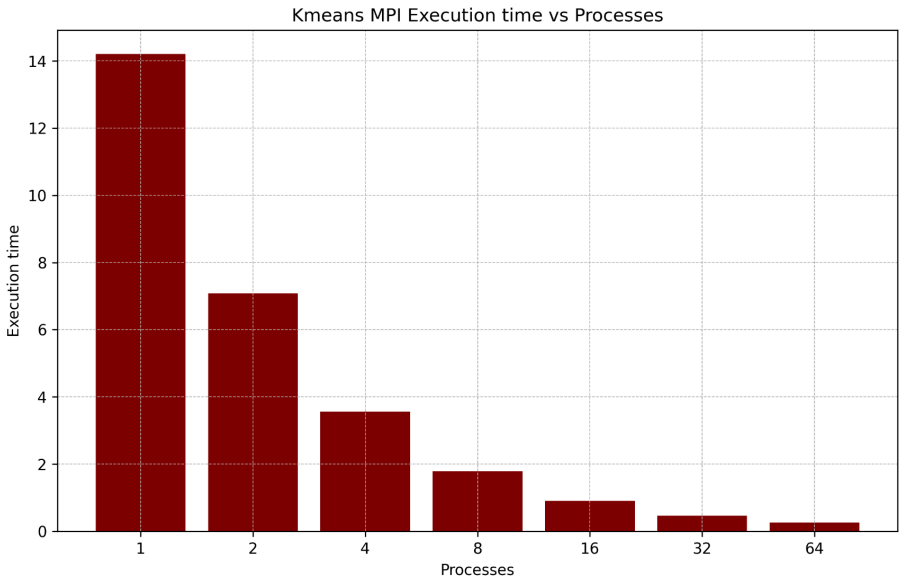
Συμπληρώνουμε το main.c στο κομμάτι των αναθέσεων των κομματιών του dataset στα επιμέρους ranks ως εξής:

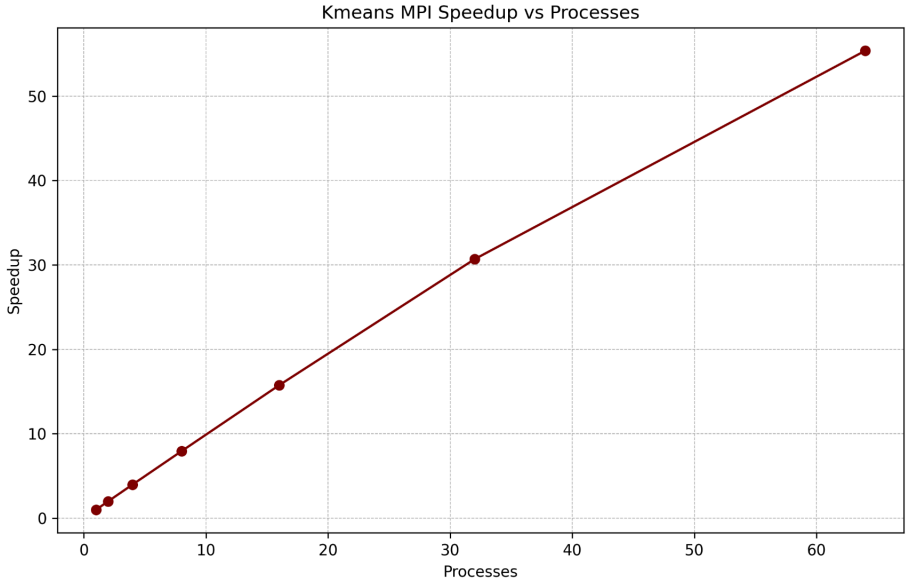
```
142     // Gather membership information from all ranks to tot_membership
143     int recvcounts[size], displs[size];
144     if (rank == 0) {
145         /* TODO: Calculate recvcounts and displs, which will be used to gather data from each rank.
146          * Hint: recvcounts: number of elements received from each rank
147          *       displs: displacement of each rank's data
148          */
149          for (i=0; i<size; i++) {
150            recvcounts[i] =  rank_numObjs  ;
151            displs[i] = i * rank_numObjs ;
152
153
154     }
155     }
156
157     /*
158      * TODO: Broadcast the recvcounts and displs arrays to other ranks.
159      */
160     MPI_Bcast(recvcounts, size, MPI_INT, 0, MPI_COMM_WORLD);
161     MPI_Bcast(displs, size, MPI_INT, 0, MPI_COMM_WORLD);
162
163
164     /*
165      * TODO: Gather membership information from every rank. (hint: each rank may send different number of objects)
166      */
167     MPI_Gatherv(membership, rank_numObjs, MPI_INT, tot_membership, recvcounts, displs, MPI_INT, 0, MPI_COMM_WORLD);
168
169
170     if (_debug && rank == 0)
171         for (i = 0; i < numObjs; ++i)
```

Στο αρχείο kmeans.c το κάθε rank αναλαμβάνει να κάνει τους υπολογισμούς του για τα δικά του objects:

```
        // update new cluster centers : sum of objects located within
        rank_newClusterSize[index]++;
        for (j=0; j<numCoords; j++)
            rank_newClusters[index*numCoords + j] += objects[i*numCoords + j];
    }

    /*
     * TODO: Perform reduction of cluster data (rank_newClusters, rank_newClusterSize) from local arrays to shared.
     */
    MPI_Allreduce(rank_newClusters, newClusters, numClusters * numCoords, MPI_DOUBLE, MPI_SUM, MPI_COMM_WORLD);
    MPI_Allreduce(rank_newClusterSize, newClusterSize, numClusters, MPI_INT, MPI_SUM, MPI_COMM_WORLD);

    // MPI_Reduce(rank_newClusters, newClusters, numClusters * numCoords, MPI_DOUBLE, MPI_SUM,0, MPI_COMM_WORLD);
    // MPI_Reduce(rank_newClusterSize, newClusterSize, numClusters, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);


    // average the sum and replace old cluster centers with newClusters
    for (i=0; i<numClusters; i++) {
        if (newClusterSize[i] > 0) {
            for (j=0; j<numCoords; j++) {
                clusters[i*numCoords + j] = newClusters[i*numCoords + j] / newClusterSize[i];
            }
        }
    }

    /*
     * TODO: Perform reduction from rank_delta variable to delta variable, that will be used for convergence check.
     */
     MPI_Allreduce(&rank_delta,&delta,1,MPI_DOUBLE, MPI_SUM,MPI_COMM_WORLD);


    // Get fraction of objects whose membership changed during this loop. This is used as a convergence criterion.
    delta /= numObjs;

```

{Size, Coords, Clusters, Loops} = {256, 16, 16,10}

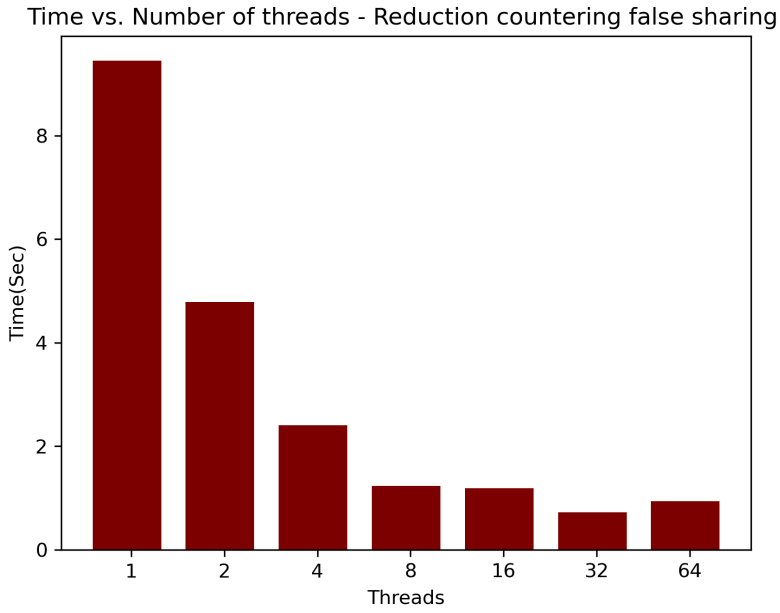


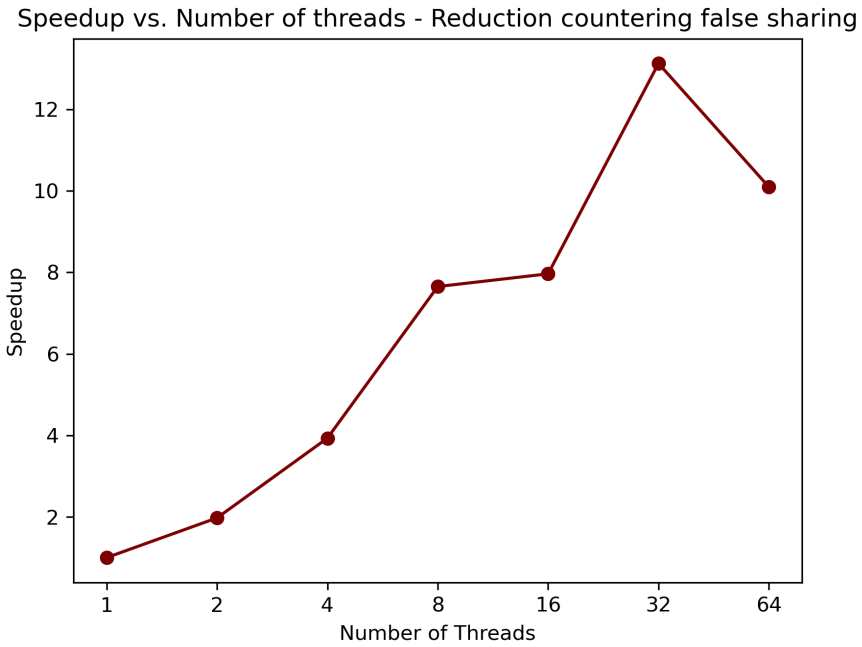


Παρατηρούμε απο τα διαγραμματα χρόνου και speedup σχεδόν τέλεια γραμμικη κλιμάκωση καθως διπλασιάζουμε τα MPI processes.

Bonus ερώτημα:

Η καλύτερη επίδοση που πετύχαμε σε αυτό το config με OpenMP είναι:





Είχαμε υλοποιήσει σχήμα reduction πηγαίνοντας να προσομοιώσουμε κάπως αυτο που γίνεται και στην MPI με distributed processing. Αυτό έγινε λόγω της διαίρει και βασίλευε λογικής του reduction.

Παρατηρούμε πως πραγματικά γραμμικό speedup φαίνεται να εμφανίζει η MPI ενώ η OpenMP λόγω thread communication και resource conflicts στη CPU και ιδιαίτερα στο πεδίο του hyperthreading (64 threads) χάνει κλιμάκωση. Στη πράξη βλέπουμε ότι πραγματικό reduction πραγματοποιείται στο MPI που τα processes γίνονται σε ξεχωριστά nodes του parlab.

Heat transfer

Μετρήσεις με έλεγχο σύγκλισης T=256 σε πίνακα 1024x1024

Μέθοδος	TotalTime
Jacobi	221.430868
Gauss-Seidel SOR	2.201772
RedBlack SOR	1.395950

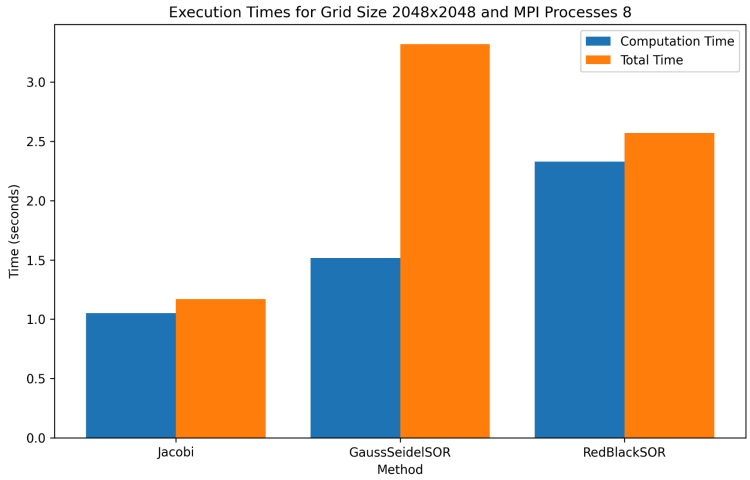
Παρατηρούμε τεράστια διαφορα στους χρόνους σύγκλισης για τις βελτιωμένες μορφές του αλγορίθμου (δηλαδή GaussSeidelSOR και RedBlackSOR). Εξηγείται η επιτάχυνση αυτή και στο πίνακα αριθμού iterations:

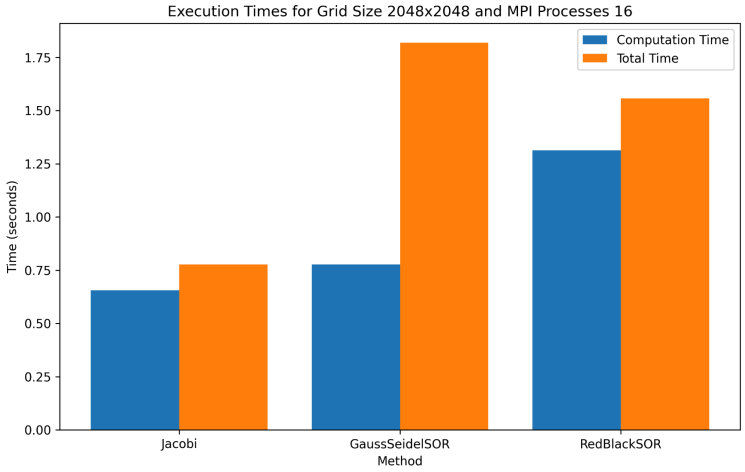
Μέθοδος	Iterations
Jacobi	798201
Gauss-Seidel SOR	3201
RedBlack SOR	2501

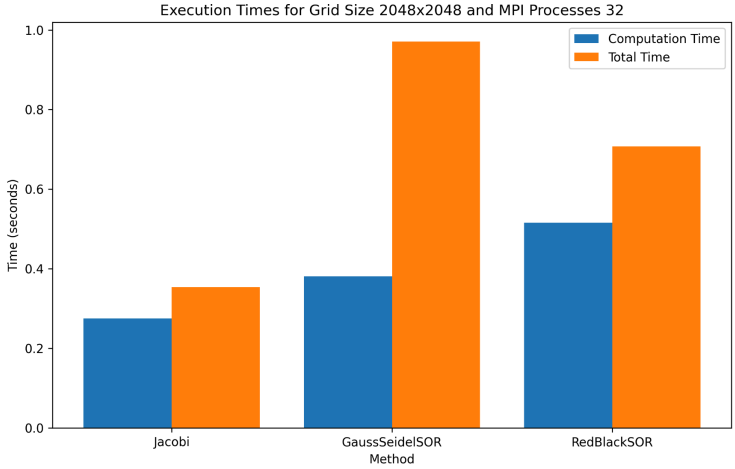
Παρολο που υπολογιστικά οι τροποποιήσεις δε φαίνεται να έχουν ουσιαστική διαφορά, μελετώντας το κώδικα, επιτυγχάνουν σύγκλιση σε λιγότερα iterations και αυτό αποτυπώνεται στο συνολικό χρόνο.

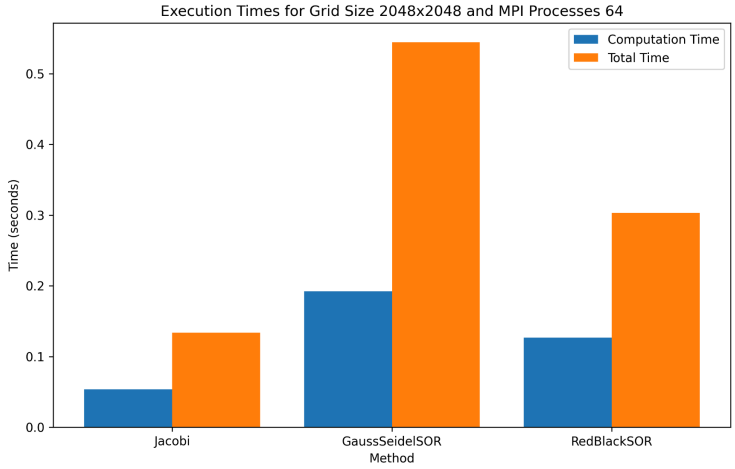
Μετρήσεις χωρίς έλεγχο σύγκλισης

Πίνακας 2048x2048

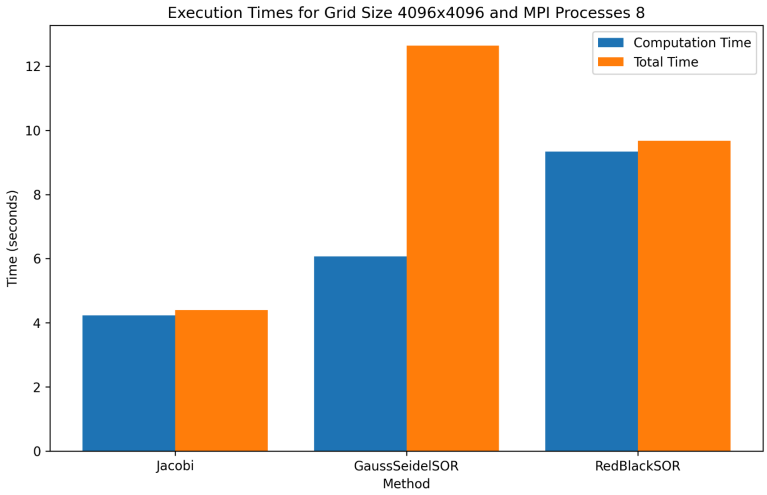


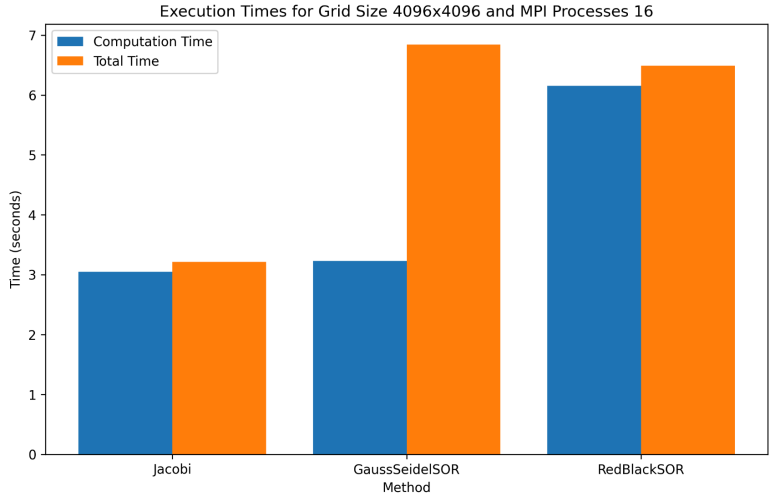

Execution Times for Grid Size 2048x2048 and MPI Processes 16
Computation Time
Total Time
Time (seconds)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
Jacobi
GaussSeidelSOR
RedBlackSOR
Method

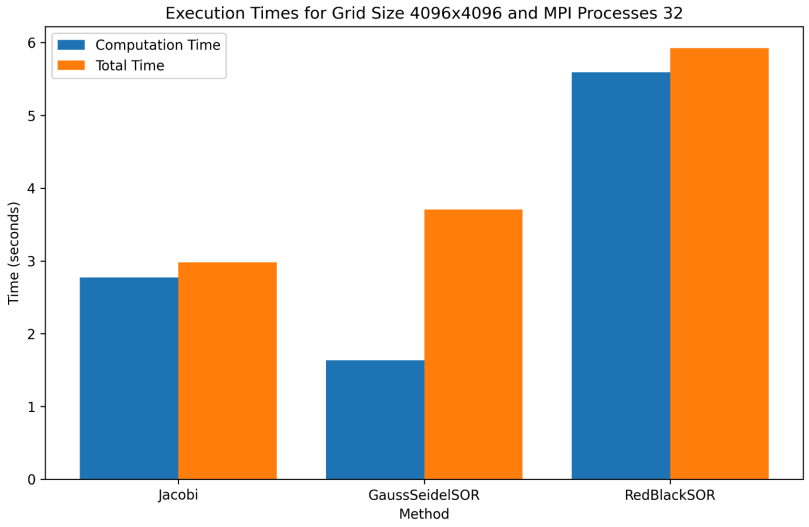

Execution Times for Grid Size 2048x2048 and MPI Processes 32
Computation Time
Total Time
Time (seconds)
0.0
0.2
0.4
0.6
0.8
1.0
Jacobi
GaussSeidelSOR
RedBlackSOR
Method

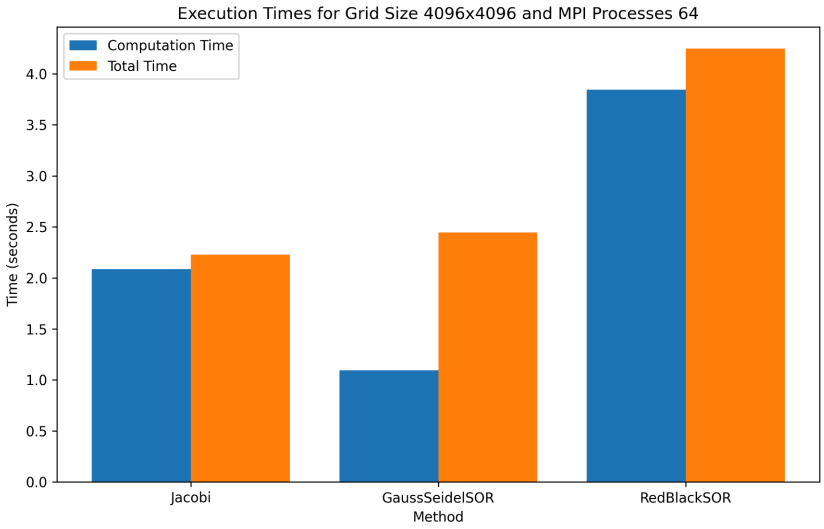

Execution Times for Grid Size 2048x2048 and MPI Processes 64
Computation Time
Total Time
Time (seconds)
0.0
0.1
0.2
0.3
0.4
0.5
Jacobi
GaussSeidelSOR
RedBlackSOR
Method

Πίνακας 4096x4096

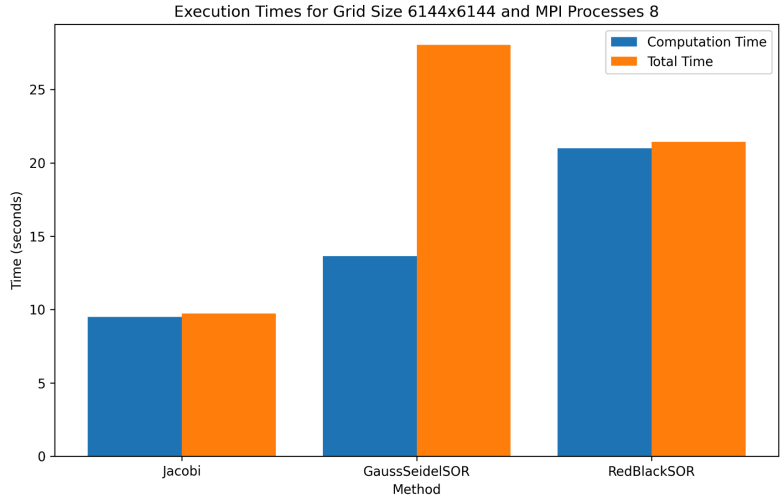


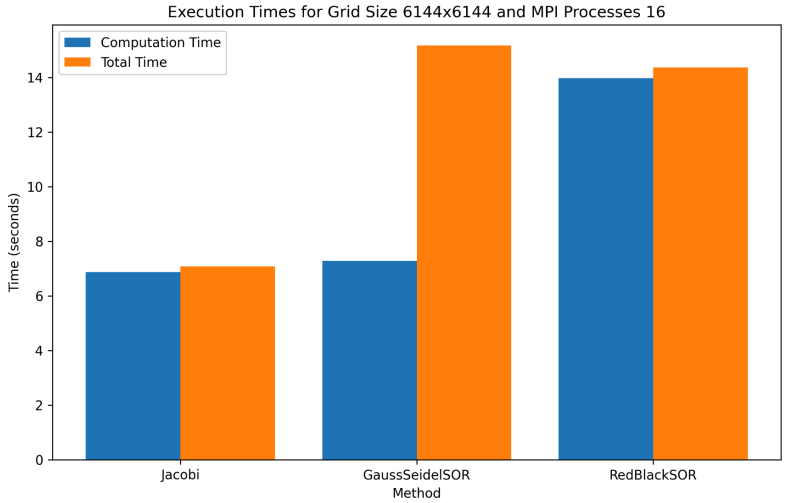


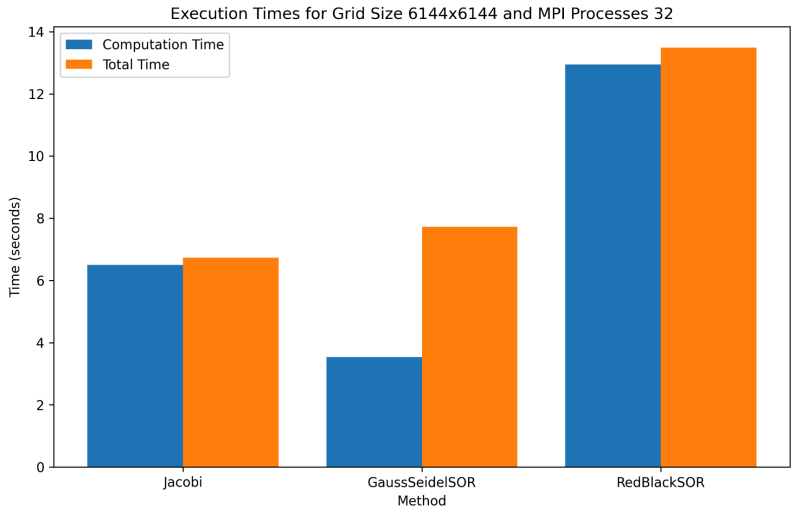

Execution Times for Grid Size 4096x4096 and MPI Processes 32
Computation Time
Total Time
Time (seconds)
0
1
2
3
4
5
6
Jacobi
GaussSeidelSOR
RedBlackSOR
Method

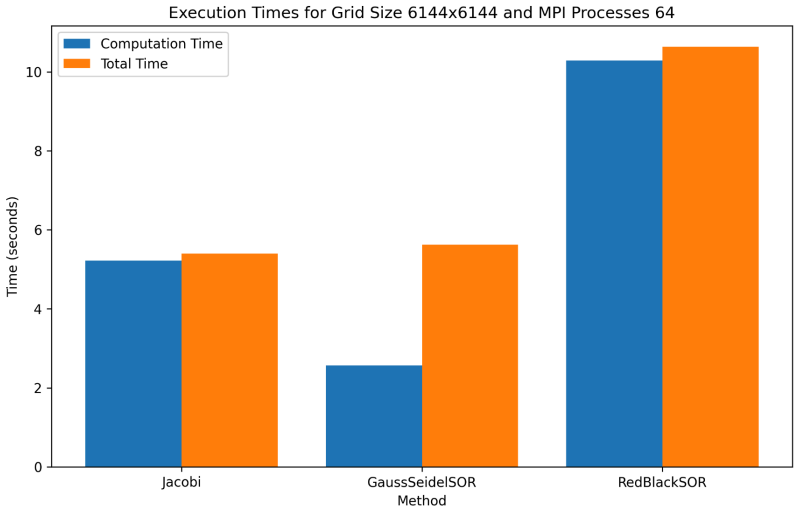

Execution Times for Grid Size 4096x4096 and MPI Processes 64
Computation Time
Total Time
Time (seconds)
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
Jacobi
GaussSeidelSOR
RedBlackSOR
Method

Πίνακας 6144x6144









Ποιοτική ανάλυση και ερμηνεία διαγραμμάτων

Jacobi: Σε κάθε επανάληψη όλες οι διεργασίες συλλέγουν αλλα και αποστέλλουν δεδομένα με γειτονικές διεργασίες ύστερα πραγματοποιούν τους τοπικούς τους υπολογισμούς. Όλες οι διεργασίες χρονικά συμμετρικές χωρίς εξαρτήσεις δεδομένων ίδιας χρονικής στιγμής κατα των υπολογισμο των νεων τιμων των σημείων.

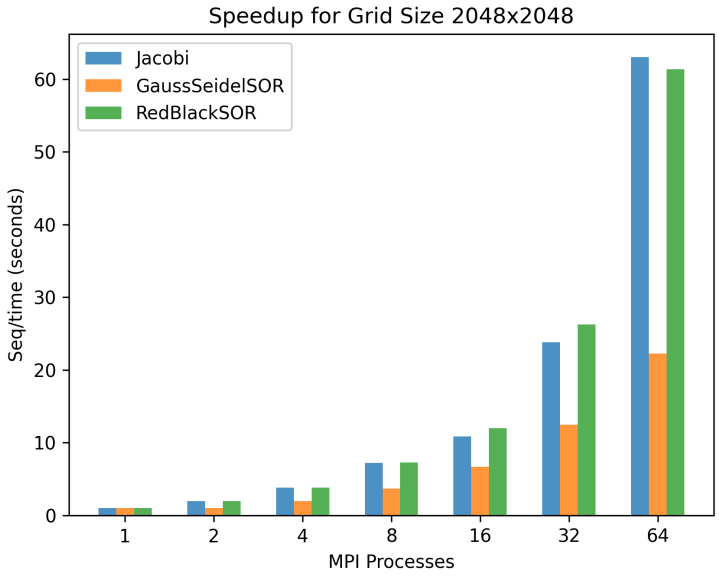
Gauss Seidel: Δεδομένης της εξάρτησης κάθε στοιχείου του πίνακα από το πιο αριστερό και το από πάνω του, δεν έχουμε συμμετρική επικοινωνία κατά τη διάρκεια κάθε επανάληψης. Ειδικότερα, απαιτείται η πρώτη διεργασία στην επάνω αριστερή γωνία να ολοκληρώσει πρώτα τους υπολογισμούς της και να μεταδώσει τα στοιχεία της προς τα δεξιά και κάτω. Το ίδιο μοτίβο

πρέπει να ακολουθήσει και κάθε επόμενη διεργασία, μεταδίδοντας τα αποτελέσματά της δεξιά και κάτω, αφού ολοκληρώσει τους υπολογισμούς της. Ωστόσο, μια διεργασία δεν μπορεί να αρχίσει πριν λάβει τα δεδομένα του τρέχοντος χρονικού βήματος από την αριστερή και την από πάνω της διεργασία. Αυτή η αλυσιδωτή εξάρτηση μεταξύ των διεργασιών οδηγεί την κάτω δεξιά διεργασία να περιμένει σημαντικά για τα δεδομένα, κάνοντας το κάθε επανάληψη να διαρκεί περισσότερο σε σύγκριση με άλλες μεθόδους.

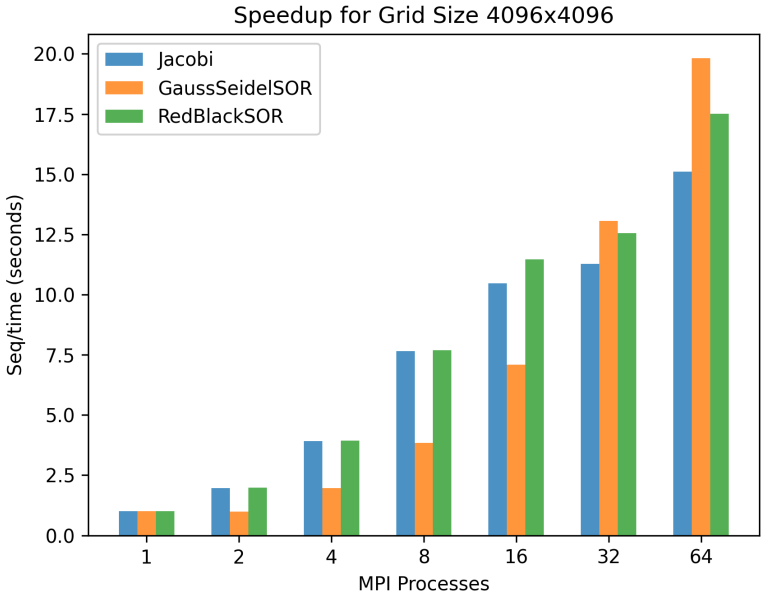
RedBlack SOR: Στη κόκκινη φαση θέλουμε δεδομένα από τη προηγουμενη επανάληψη οπότε πρώτα τα στέλνουμε απο τις υπόλοιπες διεργασίες και μετα υπολογίζουμε, ύστερα στη μαύρη φάση υπολογίζουμε βάση των προυπολογισμένων μαύρων κελιών, δεν παρατηρούμε κάποια ασυμμετρία.

Speedup για 2048,4096,6144 χωρίς έλεγχο σύγκλισης

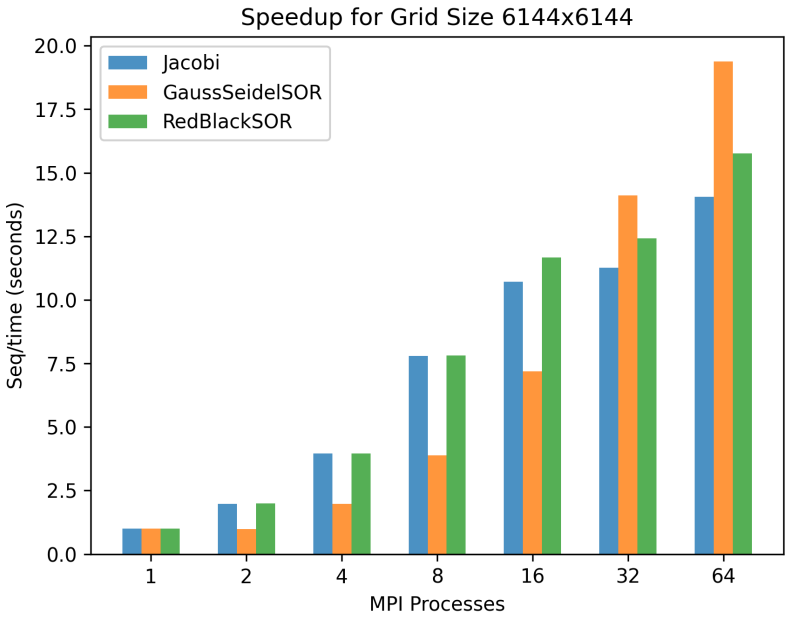
Η έννοια "time" αφορά το totaltime στα παρακάτω διαγράμματα



Εξετάζοντας το παραπάνω διάγραμμα παρατηρούμε οτι έχουμε κανονικό scaling καθώς αυξάνουμε τα threads, η μέθοδος Gauss-Seidel δυσκολεύεται να αποκτήσει μεγάλο speedup λόγω των εξαρτήσεων που αναλύσαμε και πιο πάνω. Jacobi και RedBlack φαίνεται να έχουν παρόμοια συμπεριφορά.



Για 4096x4096 παρατηρούμε ένα ήπιο σταμάτημα στο scaling για Jacobi και RedBlackSOR για πάνω από 16 threads. Αυτό οφείλεται στο cache contention των CPU. Στο parlab έχουμε συνολικά 8*2 = 16 cores για υπολογισμό. Άν πάμε να μεγαλώσουμε τα threads μπαίνουμε στα νερά του hyperthreading κάτι που αυξάνει τα conflicts στη cache. Παρατηρούμε οτι στον Gauss-Seidel δεν υπάρχει τέτοια συμπεριφορά και οφείλεται πιθανότατα οτι έχει μειωμένο cache contention λόγω οι υπολογισμοί των κελιών δε πραγματοποιούνται ακριβώς συγχρονισμένα, λόγω αλυσιδωτής σχέσης.



Ίδια συμπεριφορά παρατηρούμε και για 6144x6144. Μπορούμε να το ερμηνεύσουμε υπο το ίδιο πρίσμα όπως και για τη περίπτωση 4096x4096.