Συστήματα Παράλληλης Επεξεργασίας

Άσκηση 4
4.1 Διάδοση θερμότητας σε δύο διαστάσεις

Ομάδα	parlab04
Θεόδωρος Αράπης	el18028
Εμμανουήλ Βλάσσης	el18086
Παναγιώτης Παπαδέας	el18039

Ζητούμενα

Ως δεδομένα στην Άσκηση δόθηκαν οι σειριακές υλοποιήσεις των αλγορίθμων Jacobi, Gauss-Seidel-SOR και Red-Black-SOR. Ζητούνται:

1. Ο εντοπισμός του παραλληλισμού των αλγορίθμων και σχεδίαση της παράλληλης υλοποίησή τους σε αρχιτεκτονική κατανεμημένης μνήμης που υποστηρίζει ανταλλαγή μηνυμάτων.
2. Η παραλληλοποίησή τους με χρήση της βιβλιοθήκης MPI.
3. Μετρήσεις επίδοσης.
4. Σύγκριση αποτελεσμάτων και σχολιασμός.

Σχεδιασμός Παραλληλοποίησης

Πίνακας U

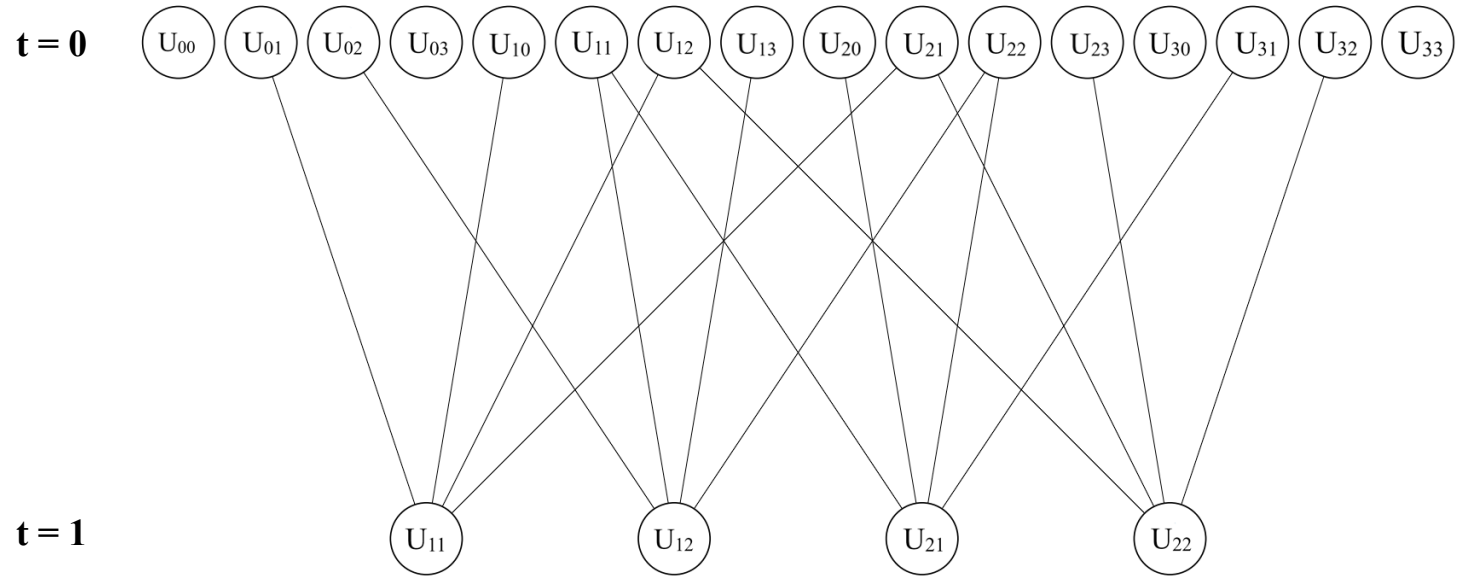
Ο πίνακας U έχει διαστάσεις $X \times Y$. Θεωρώντας για ευκολία σχεδίασης ότι $X = Y = 4$ έχουμε (το οποίο μπορεί να επεκταθεί έτσι ώστε U_{ij} = μπλοκ μεγέθους x, y):

U =

U_{00}	U_{01}	U_{02}	U_{03}
U_{10}	U_{11}	U_{12}	U_{13}
U_{20}	U_{21}	U_{22}	U_{23}
U_{30}	U_{31}	U_{32}	U_{33}

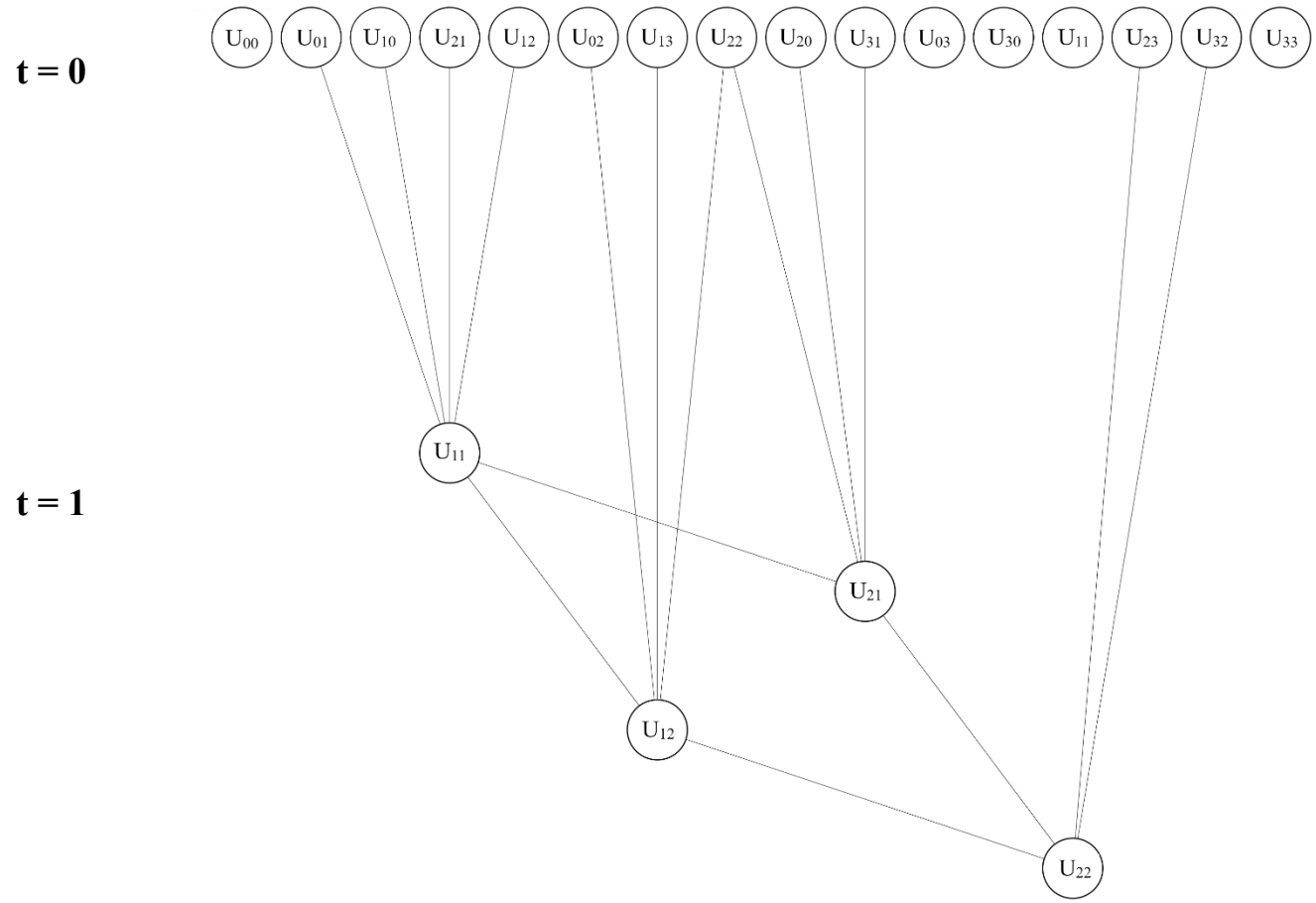
Μέθοδος Jacobi

Θα σχεδιάσουμε το task graph για τις δύο πρώτες χρονικές στιγμές. Χαρακτηρίζουμε ως task την θερμότητα σε κάθε θέση (ή μπλοκ θέσεων) για μία χρονική στιγμή:



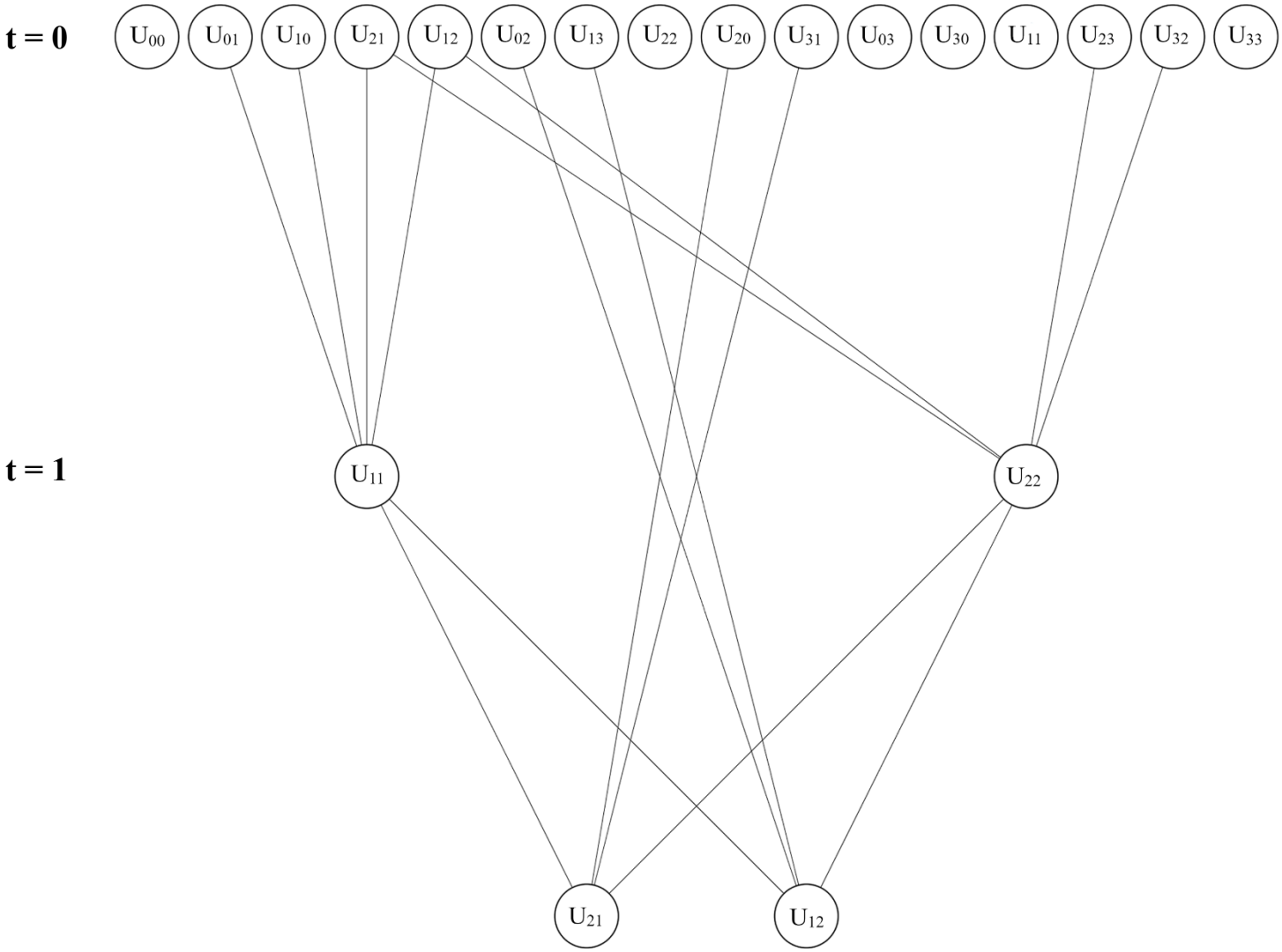
Μέθοδος Gauss-Seidel SOR

Όμοια και δω θα σχεδιάσουμε το task graph για τις δύο πρώτες χρονικές στιγμές. Θεωρούμε, όπως και πριν, ως task την θερμότητα σε κάθε θέση (ή μπλοκ θέσεων) για μία χρονική στιγμή:



Μέθοδος Red-Black SOR

Τέλος, το task graph για τις δύο πρώτες χρονικές στιγμές για την μέθοδο Red-Black SOR, θεωρώντας πάλι ως task την θερμότητα σε κάθε θέση (ή μπλοκ θέσεων) για μία χρονική στιγμή, είναι:



Ανάπτυξη παράλληλων προγραμμάτων

Πίνακας U

Και για τις 3 υλοποιήσεις, ο αρχικός πίνακας U χωρίζεται σε block, ένα για κάθε διεργασία. Σε κάθε επανάληψη, οι διεργασίες ανταλλάσσουν δεδομένα μεταξύ τους και ύστερα υπολογίζουν τις νέες τιμές. Όταν όλα τα block συγκλίνουν στις τελικές τιμές τους, τότε η μέθοδος συγκλίνει. Στο τέλος, οι τιμές των επιμέρους block συνενώνονται στον αρχικό πίνακα. Οι ακόλουθες διαδικασίες εφαρμόζονται και στις 3 υλοποιήσεις:

1. Δημιουργία 2D καρτεσιανού communicator.

```
43    //----Create 2D-cartesian communicator----//
44    //----Usage of the cartesian communicator is optional----//
45
46    MPI_Comm CART_COMM;         //CART_COMM: the new 2D-cartesian communicator
47    int periods[2]={0,0};       //periods={0,0}: the 2D-grid is non-periodic
48    int rank_grid[2];           //rank_grid: the position of each process on the new communicator
49
50    MPI_Cart_create(MPI_COMM_WORLD,2,grid,periods,0,&CART_COMM);    //communicator creation
51    MPI_Cart_coords(CART_COMM,rank,2,rank_grid);                  //rank mapping on the new communicator
```

2. Υπολογισμός διαστάσεων του επιμέρους πίνακα κάθε block (χρήση padding στον αρχικό πίνακα σε περίπτωση που ο διαμοιρασμός των block δεν χωράει ακριβώς).

```
54    //----Compute local 2D-subdomain dimensions----//
55    //----Test if the 2D-domain can be equally distributed to all processes----//
56    //----If not, pad 2D-domain----//
57
58    for (i=0;i<2;i++) {
59        if (global[i]%grid[i]==0) {
60            local[i]=global[i]/grid[i];
61            global_padded[i]=global[i];
62        }
63        else {
64            local[i]=(global[i]/grid[i])+1;
65            global_padded[i]=local[i]*grid[i];
66        }
67    }
```

3. Διαμοιρασμός του αρχικού global πίνακα U στον local πίνακα κάθε διεργασίας.

```
84      //----Distribute global 2D-domain from rank 0 to all processes----//
85
86      //----Appropriate datatypes are defined here----//
87      /*****The usage of datatypes is optional*****/
88
89      //----Datatype definition for the 2D-subdomain on the global matrix----//
90
91      MPI_Datatype global_block;
92      MPI_Type_vector(local[0],local[1],global_padded[1],MPI_DOUBLE,&dummy);
93      MPI_Type_create_resized(dummy,0,sizeof(double),&global_block);
94      MPI_Type_commit(&global_block);
95
96      //----Datatype definition for the 2D-subdomain on the local matrix----//
97
98      MPI_Datatype local_block;
99      MPI_Type_vector(local[0],local[1],local[1]+2,MPI_DOUBLE,&dummy);
100     MPI_Type_create_resized(dummy,0,sizeof(double),&local_block);
101     MPI_Type_commit(&local_block);
102
103     //----Rank 0 defines positions and counts of local blocks (2D-subdomains) on global matrix----//
104     int * scatteroffset, * scattercounts;
105     if (rank==0) {
106         U_start = &(U[0][0]);
107         scatteroffset=(int*)malloc(size*sizeof(int));
108         scattercounts=(int*)malloc(size*sizeof(int));
109         for (i=0;i<grid[0];i++)
110             for (j=0;j<grid[1];j++) {
111                 scattercounts[i*grid[1]+j]=1;
112                 scatteroffset[i*grid[1]+j]=(local[0]*local[1]*grid[1]*i+local[1]*j);
113             }
114     }
115
116     //----Rank 0 scatters the global matrix----//
117
118     MPI_Scatterv(U_start, scattercounts, scatteroffset, global_block, &(u_previous[1][1]), 1, local_block, 0,
          MPI_COMM_WORLD);
119     MPI_Scatterv(U_start, scattercounts, scatteroffset, global_block, &(u_current[1][1]), 1, local_block, 0,
          MPI_COMM_WORLD);
```

4. Ορισμός μιας δομής που αντιστοιχεί σε ένα local column, προκειμένου να διευκολύνει την μεταφορά μιας στήλης ενός πίνακα μεταξύ των διεργασιών.

```
133     /* Define a datatype that corresponds to a collumn of a local block */
134     MPI_Datatype column;
135     MPI_Type_vector(local[0], 1,local[1]+2,MPI_DOUBLE,&dummy);
136     MPI_Type_create_resized(dummy,0,sizeof(double),&column);
137     MPI_Type_commit(&column);
```

5. Για κάθε διεργασία υπολογίζουμε το rank των γειτόνων της ώστε να μπορεί να επικοινωνήσει μαζί τους.

```
143     //----Find the 4 neighbors with which a process exchanges messages----//
144
145     int north, south, east, west;
146     MPI_Cart_shift(CART_COMM, 0, 1, &north, &south);
147     MPI_Cart_shift(CART_COMM, 1, 1, &west, &east);
```

6. Ορισμός του range πάνω στο οποίο η διεργασία θα ανανεώνει τον local πίνακά της. Εδώ, πρέπει να λάβουμε υπόψιν τόσο τα ghost cell που έχουν τοποθετηθεί για την διευκόλυνση της επικοινωνίας όσο και τα πιθανά padding cells.

```
150     /*Make sure you handle non-existing neighbors appropriately*/
151
152     /* In case of non-existing neighbors, the function returns MPI_PROC_NULL */
153
154     //**********************************//
155
156     //---Define the iteration ranges per process-----//
157
158     int i_min, i_max, j_min, j_max;
159
160     /* internal process (ghost cell only) */
161     i_min = 1;
162     i_max = local[0] + 1;
163
164     /* boundary process - no possible padding */
165     if (north == MPI_PROC_NULL) {
166         i_min = 2;  // ghost cell + boundary
167     }
168
169     /* boundary process and padded global array */
170     if (south == MPI_PROC_NULL){
171         i_max -= (global_padded[0] - global[0]) + 1;
172     }
173
174     /* internal process (ghost cell only) */
175     j_min = 1;
176     j_max = local[1] + 1;
177
178     /* boundary process - no possible padding */
179     if (west == MPI_PROC_NULL) {
180         j_min = 2;  //ghost cell + boundary
181     }
182
183     /* boundary process and padded global array */
184     if (east == MPI_PROC_NULL){
185         j_max -= (global_padded[1] - global[1]) + 1;
186     }
```

7. Έλεγχος σύγκλισης, (προϋπόθεση όλα τα local block να έχουν συγκλίνει).

```
#ifdef TEST_CONV
        if (t%C==0) {
            /*Test convergence*/
            gettimeofday(&tcvs, NULL);

            converged=converge(u_previous,u_current,local[0],local[1]);

            MPI_Allreduce(&converged, &global_converged, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
            gettimeofday(&tcvf, NULL);
            tconv += (tcvf.tv_sec-tcvs.tv_sec)+(tcvf.tv_usec-tcvs.tv_usec)*0.000001;
        }
#endif
```

8. Συνένωση των επιμέρους local πινάκων στον αρχικό global πίνακα U, τον οποίο χειρίζεται η διεργασία με rank = 0

```
    //----Rank 0 gathers local matrices back to the global matrix----//

    if (rank==0) {
        U=allocate2d(global_padded[0],global_padded[1]);
        U_start = &(U[0][0]);
    }


    /*Fill your code here*/

    MPI_Gatherv(&u_current[1][1], 1, local_block, U_start, scattercounts, scatteroffset, global_block, 0,
      MPI_COMM_WORLD);
```

Jacobi

Σε αυτήν την περίπτωση, κάθε διεργασία που επεξεργάζεται ένα block πρέπει να επικοινωνήσει πρώτα με της γειτονικές διεργασίες της και να μάθει τα σύνορα των γειτονικών blocks. Οι τιμές αυτές αποθηκεύονται στα ghost cells που αναφέρθηκαν προηγουμένως, με αποτέλεσμα να μην χρειάζεται να αλλάξει καθόλου ο βρόγχος ανανέωσης του σειριακού προγράμματος. Μετά το τέλος της επικοινωνίας, κάθε κελί θα έχει λάβει τις σωστές τιμές από τους γείτονές του.

```
    //----Computational core----//
    gettimeofday(&tts, NULL);
#ifdef TEST_CONV
    for (t=0;t<T && !global_converged;t++) {
#endif
#ifndef TEST_CONV
#undef T
#define T 256
    for (t=0;t<T;t++) {
#endif


        /*Fill your code here*/
        swap=u_previous;
        u_previous=u_current;
        u_current=swap;
```

```
219         /*Compute and Communicate*/
220
221         // Communicate with north
222         if (north != MPI_PROC_NULL){
223             MPI_Sendrecv(&u_previous[1][1], local[1], MPI_DOUBLE, north, 0, &u_previous[0][1],
224                          local[1], MPI_DOUBLE, north, 0, MPI_COMM_WORLD, &status );
225         }
226
227         // Communicate with south
228         if (south != MPI_PROC_NULL){
229             MPI_Sendrecv(&u_previous[local[0]][1], local[1], MPI_DOUBLE, south, 0, &u_previous[local[0]+1][1],
230                          local[1], MPI_DOUBLE, south, 0, MPI_COMM_WORLD, &status );
231         }
232
233         // Communicate with east
234         if (east != MPI_PROC_NULL){
235             MPI_Sendrecv(&u_previous[1][local[1]], 1, column, east, 0, &u_previous[1][local[1]+1],
236                          1, column, east, 0, MPI_COMM_WORLD, &status );
237         }
238
239         // Communicate with west
240         if (west != MPI_PROC_NULL){
241             MPI_Sendrecv(&u_previous[1][1], 1, column, west, 0, &u_previous[1][0],
242                          1, column, west, 0, MPI_COMM_WORLD, &status );
243         }
244
245
246         /*Add appropriate timers for computation*/
247         gettimeofday(&tcs, NULL);
248
249         for (i=i_min;i<i_max;i++)
250             for (j=j_min;j<j_max;j++)
251                 u_current[i][j]=(u_previous[i-1][j]+u_previous[i+1][j]+u_previous[i][j-1]+u_previous[i][j+1])/4.0;
252
253         gettimeofday(&tcf, NULL);
254
255         tcomp += (tcf.tv_sec-tcs.tv_sec)+(tcf.tv_usec-tcs.tv_usec)*0.000001;
256
257
258 #ifdef TEST_CONV
259         if (t%C==0) {
260             /*Test convergence*/
261             gettimeofday(&tcvs, NULL);
262
263             converged=converge(u_previous,u_current,local[0],local[1]);
264
265             MPI_Allreduce(&converged, &global_converged, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
266             gettimeofday(&tcvf, NULL);
267             tconv += (tcvf.tv_sec-tcvs.tv_sec)+(tcvf.tv_usec-tcvs.tv_usec)*0.000001;
268         }
269 #endif
270
271     }
272     gettimeofday(&ttf,NULL);
273
274     ttotal=(ttf.tv_sec-tts.tv_sec)+(ttf.tv_usec-tts.tv_usec)*0.000001;
275
276     MPI_Reduce(&ttotal,&total_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
277     MPI_Reduce(&tcomp,&comp_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
278     MPI_Reduce(&tconv,&conv_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
```

Gauss-Siedel SOR

Στην μέθοδο αυτή, η υλοποίηση της επικοινωνίας (πρωτού γίνουν οι υπολογισμοί) γίνεται ως εξής για κάθε block:

Η κάθε διεργασία στέλνει στον πάνω και στον αριστερά γείτονα τα δεδομένα της προηγούμενης χρονικής στιγμής. Στη συνέχεια λαμβάνει από τον πάνω και τον αριστερά γείτονα τα δεδομένα της τωρινής χρονικής στιγμής. Επιπλέον, λαμβάνει από τον δεξιά και τον κάτω γείτονα τα δεδομένα της προηγούμενης χρονικής στιγμής. Τέλος, μετά τον υπολογισμό, στέλνει στον δεξιά και στον κάτω γείτονα τα δεδομένα της τωρινής χρονική στιγμής.

Πιο γενικά, όλες οι διεργασίες, εκτός από αυτές που βρίσκονται πάνω και αριστερά στο ταμπλό, θα μπλοκάρουν περιμένοντας τα τωρινά δεδομένα από τον πάνω και τον αριστερά γείτονά τους. Οι διεργασίες που δεν έχουν γείτονα πάνω και αριστερά θα υπολογίσουν τις νέες τιμές και θα τις στείλουν στον κάτω και δεξιά γείτονά τους. Η διαδικασία αυτή θα επαναληφθεί μέχρι όλα τα block να ανανεωθούν. Για τον λόγο αυτό, χρησιμοποιήσαμε non-blocking send και receive συναρτήσεις και θέσαμε στα σημεία όπου χρειαζόμαστε τις γειτονικές τιμές το κατάλληλο wait.

```
25      MPI_Request before_req[6];
26      MPI_Status before_status[6];
27      MPI_Request after_req[2];
28      MPI_Status after_status[2];
29      int before_req_len = 0;
30      int after_req_len = 0;
```

```
207     //----Computational core----//
208     gettimeofday(&tts, NULL);
209     #ifdef TEST_CONV
210     for (t=0;t<T && !global_converged;t++) {
211     #endif
212     #ifndef TEST_CONV
213     #undef T
214     #define T 256
215     for (t=0;t<T;t++) {
216     #endif
217
218         before_req_len = 0;
219         after_req_len = 0;
220
221         swap=u_previous;
222         u_previous=u_current;
223         u_current=swap;
224
225         /* Send data to north */
226         if (north != MPI_PROC_NULL){
227             MPI_Isend(&u_previous[1][1], local[1], MPI_DOUBLE, north, 0, MPI_COMM_WORLD, &before_req[before_req_len]);
228             MPI_Irecv(&u_current[0][1], local[1], MPI_DOUBLE, north, 0, MPI_COMM_WORLD, &before_req[before_req_len + 1]);
229             before_req_len += 2;
230         }
231
232         /* Send data to west */
233         if (west != MPI_PROC_NULL){
234             MPI_Isend(&u_previous[1][1], 1, column, west, 0, MPI_COMM_WORLD, &before_req[before_req_len]);
235             MPI_Irecv(&u_current[1][0], 1, column, west, 0, MPI_COMM_WORLD, &before_req[before_req_len + 1]);
236             before_req_len += 2;
237         }
238
239
240         /* Get data from south */
241         if (south != MPI_PROC_NULL){
242             MPI_Irecv(&u_previous[local[0]+1][1], local[1], MPI_DOUBLE, south, 0, MPI_COMM_WORLD, &before_req[before_req_len]);
243             before_req_len ++;
244         }
245         /* Get data from east */
246         if (east != MPI_PROC_NULL){
247             MPI_Irecv(&u_previous[1][local[1]+1], 1, column, east, 0, MPI_COMM_WORLD, &before_req[before_req_len]);
248             before_req_len ++;
249         }
250
251
252         /* Add barrier */
253
254         MPI_Waitall(before_req_len, before_req, before_status);
```

```
256         /*Compute and Communicate*/
257
258         /*Add appropriate timers for computation*/
259         gettimeofday(&tcs, NULL);
260
261         for (i=i_min;i<i_max;i++)
262                 for (j=j_min;j<j_max;j++)
263                         u_current[i][j]=u_previous[i][j] + (omega/4.0)*(u_current[i-1][j]+u_previous[i+1][j]+u_current[i][j-1]+u_previous[i][j+1]-4*u_previous[i][j]);
264
265         gettimeofday(&tcf, NULL);
266
267         tcomp += (tcf.tv_sec-tcs.tv_sec)+(tcf.tv_usec-tcs.tv_usec)*0.000001;
268
269         /* Send data to south */
270         if (south != MPI_PROC_NULL){
271             MPI_Isend(&u_current[local[0]][1], local[1], MPI_DOUBLE, south, 0, MPI_COMM_WORLD, &after_req[after_req_len]);
272             after_req_len ++;
273         }
274
275         /* Send data to east */
276         if (east != MPI_PROC_NULL){
277             MPI_Isend(&u_current[1][local[1]], 1, column, east, 0, MPI_COMM_WORLD, &after_req[after_req_len]);
278             after_req_len ++;
279         }
280
281         MPI_Waitall(after_req_len, after_req, after_status);
282
283 #ifdef TEST_CONV
284         if (t%C==0) {
285             /*Test convergence*/
286             gettimeofday(&tcvs, NULL);
287             converged=converge(u_previous,u_current,local[0],local[1]);
288
289             MPI_Allreduce(&converged, &global_converged, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
290             gettimeofday(&tcvf, NULL);
291             tconv += (tcvf.tv_sec-tcvs.tv_sec)+(tcvf.tv_usec-tcvs.tv_usec)*0.000001;
292         }
293 #endif
294
295     }
296     gettimeofday(&ttf,NULL);
297
298     ttotal=(ttf.tv_sec-tts.tv_sec)+(ttf.tv_usec-tts.tv_usec)*0.000001;
299
300     MPI_Reduce(&ttotal,&total_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
301     MPI_Reduce(&tcomp,&comp_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
302     MPI_Reduce(&tconv,&conv_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
```

Red-Black SOR

Στην περίπτωση της μεθόδου Red-Black SOR έχουμε δύο φάσεις υπολογισμού. Στην πρώτη φάση, τα κόκκινα στοιχεία πρέπει να γνωρίζουν τους γείτονές τους, ενώ στην δεύτερη φάση τα μαύρα στοιχεία πρέπει να κάνουν το ίδιο. Μιας και τα κόκκινα και τα μαύρα στοιχεία τοποθετούνται εναλλάξ στις άρτιες και περιττές θέσεις, και στις δύο φάσεις όλα τα block χρειάζεται να επικοινωνήσουν με τους γείτονές τους. Συνεπώς, η διαδικασία τώρα μοιάζει με αυτήν του Jacobi, με την διαφορά ότι τώρα θα γίνει 2 φορές η επικοινωνία, μία για να ενημερωθεί ο πίνακας u_previous και να υπολογιστούν τα κόκκινα και μία για να ενημερωθεί ο πίνακας u_current και να υπολογιστούν τα μαύρα στοιχεία. Ακολουθεί η υλοποίηση του κώδικα:

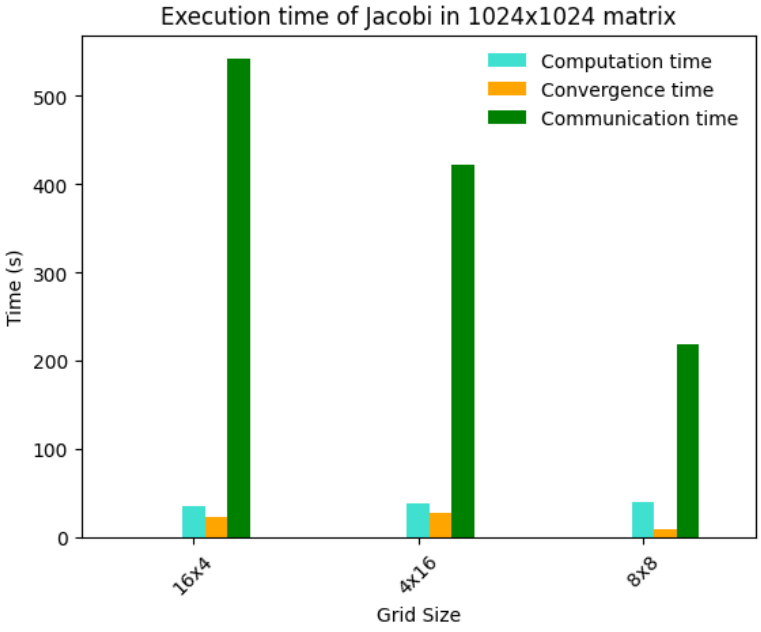
```
200         //----Computational core----//
201         gettimeofday(&tts, NULL);
202     #ifdef TEST_CONV
203         for (t=0;t<T && !global_converged;t++) {
204     #endif
205     #ifndef TEST_CONV
206     #undef T
207     #define T 256
208         for (t=0;t<T;t++) {
209     #endif
210
211             swap=u_previous;
212             u_previous=u_current;
213             u_current=swap;
```

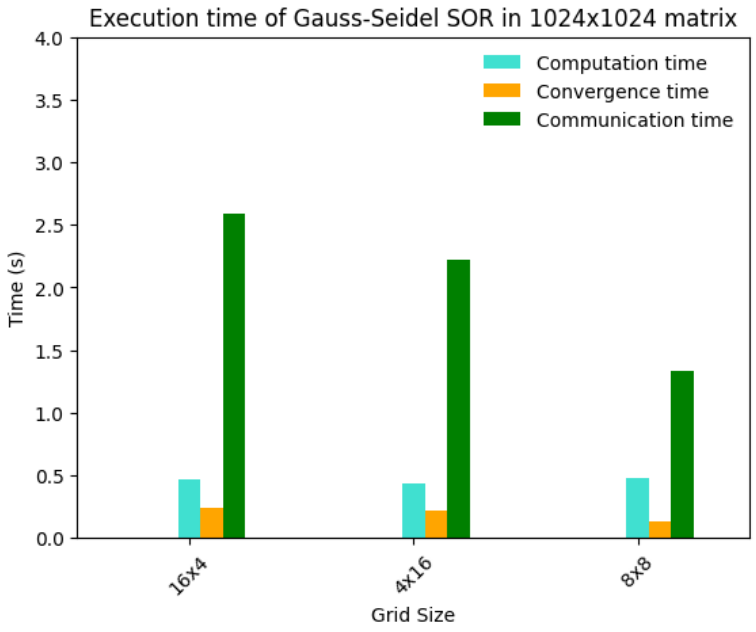
```
        /*Compute and Communicate*/

        if (north != MPI_PROC_NULL){
            MPI_Sendrecv(&u_previous[1][1], local[1], MPI_DOUBLE, north, 0, &u_previous[0][1],
                         local[1], MPI_DOUBLE, north, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with south
        if (south != MPI_PROC_NULL){
            MPI_Sendrecv(&u_previous[local[0]][1], local[1], MPI_DOUBLE, south, 0, &u_previous[local[0]+1][1],
                         local[1], MPI_DOUBLE, south, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with east
        if (east != MPI_PROC_NULL){
            MPI_Sendrecv(&u_previous[1][local[1]], 1, column, east, 0, &u_previous[1][local[1]+1],
                         1, column, east, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with west
        if (west != MPI_PROC_NULL){
            MPI_Sendrecv(&u_previous[1][1], 1, column, west, 0, &u_previous[1][0],
                         1, column, west, 0, MPI_COMM_WORLD, &status );
        }

        /*Add appropriate timers for computation*/
        gettimeofday(&tcs, NULL);

        for (i=i_min;i<i_max;i++)
            for (j=j_min;j<j_max;j++)
                if ((i+j)%2==0)
                    u_current[i][j]=u_previous[i][j]+(omega/4.0)*(u_previous[i-1][j]+u_previous[i+1][j]+u_previous[i][j-1]+u_previous[i][j+1]-4*u_previous[i]
                      [j]);

        gettimeofday(&tcf, NULL);

        tcomp += (tcf.tv_sec-tcs.tv_sec)+(tcf.tv_usec-tcs.tv_usec)*0.000001;

        // Communicate with north
        if (north != MPI_PROC_NULL){
            MPI_Sendrecv(&u_current[1][1], local[1], MPI_DOUBLE, north, 0, &u_current[0][1],
                         local[1], MPI_DOUBLE, north, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with south
        if (south != MPI_PROC_NULL){
            MPI_Sendrecv(&u_current[local[0]][1], local[1], MPI_DOUBLE, south, 0, &u_current[local[0]+1][1],
                         local[1], MPI_DOUBLE, south, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with east
        if (east != MPI_PROC_NULL){
            MPI_Sendrecv(&u_current[1][local[1]], 1, column, east, 0, &u_current[1][local[1]+1],
                         1, column, east, 0, MPI_COMM_WORLD, &status );
        }

        // Communicate with west
        if (west != MPI_PROC_NULL){
            MPI_Sendrecv(&u_current[1][1], 1, column, west, 0, &u_current[1][0],
                         1, column, west, 0, MPI_COMM_WORLD, &status );
        }

        gettimeofday(&tcs, NULL);

        for (i=i_min;i<i_max;i++)
            for (j=j_min;j<j_max;j++)
                if ((i+j)%2==1)
                    u_current[i][j]=u_previous[i][j]+(omega/4.0)*(u_current[i-1][j]+u_current[i+1][j]+u_current[i][j-1]+u_current[i][j+1]-4*u_previous[i]
                      [j]);

        gettimeofday(&tcf, NULL);

        tcomp += (tcf.tv_sec-tcs.tv_sec)+(tcf.tv_usec-tcs.tv_usec)*0.000001;



#ifdef TEST_CONV
        if (t%C==0) {
            /*Test convergence*/
            gettimeofday(&tcvs, NULL);

            converged=converge(u_previous,u_current,local[0],local[1]);

            MPI_Allreduce(&converged, &global_converged, 1, MPI_INT, MPI_MIN, MPI_COMM_WORLD);
            gettimeofday(&tcvf, NULL);
            tconv += (tcvf.tv_sec-tcvs.tv_sec)+(tcvf.tv_usec-tcvs.tv_usec)*0.000001;
        }
#endif
        //**********************************//
    }
    gettimeofday(&ttf,NULL);

    ttotal=(ttf.tv_sec-tts.tv_sec)+(ttf.tv_usec-tts.tv_usec)*0.000001;

    MPI_Reduce(&ttotal,&total_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
    MPI_Reduce(&tcomp,&comp_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
    MPI_Reduce(&tconv,&conv_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
```

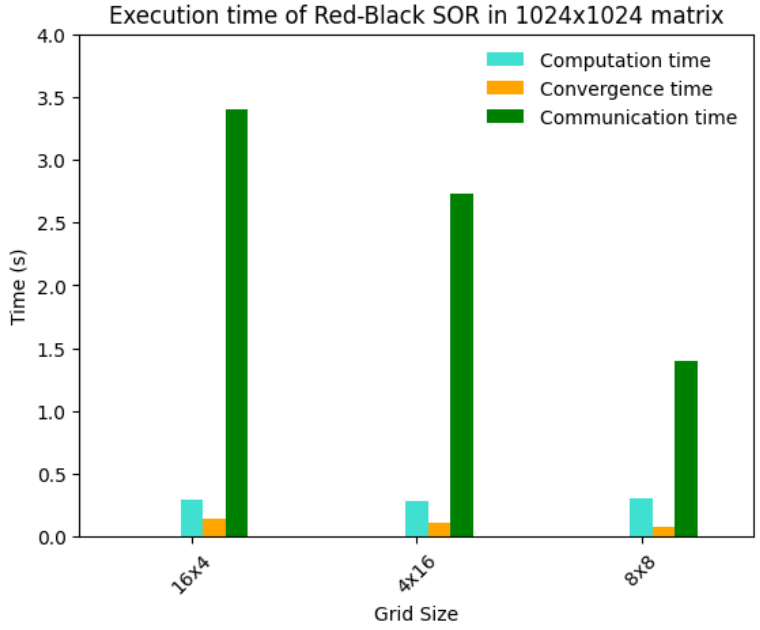
Μετρήσεις επίδοσης

Μετρήσεις με έλεγχο σύγκλισης

Στο κομμάτι αυτό, θα πραγματοποιήσαμε μετρήσεις με έλεγχο σύγκλισης για τα παράλληλα προγράμματά που υλοποιήσαμε σε MPI, για μέγεθος πίνακα 1024×1024. Γι' αυτό το μέγεθος πίνακα, λάβαμε μετρήσεις (συνολικός χρόνος, χρόνος υπολογισμών, χρόνος ελέγχου σύγκλισης) για 64 MPI διεργασίες. Συνοψίζουμε τις μετρήσεις στα ακόλουθα διαγράμματα για κάθε μέθοδο:







Εύκολα παρατηρούμε ότι στις υλοποιήσεις που έχουμε grid size 8×8 η εκτέλεση είναι πιο γρήγορη. Αυτό συμβαίνει διότι τα μηνύματα που ανταλλάσσονται μεταξύ των διεργασιών έχουν ίδιο μέγεθος, αφού είναι συμμετρικό το grid size, ενώ στις περιπτώσεις των 16×4 και 4×16, η εκτέλεση είναι πιο αργή, καθώς η αποστολή μηνυμάτων στην πλευρά ενός μπλοκ προκαλεί σημαντικές καθυστερήσεις.

Για κάθε μέθοδο συγκεκριμένα έχουμε τις εξής παρατηρήσεις:

Jacobi

Η μέθοδος Jacobi έχει ως μειονέκτημα ότι απαιτεί πολλές επαναλήψεις του αλγόριθμου προκειμένου να συγκλίνει, γεγονός που την καθιστά την πιο αργή από τις τρεις μεθόδους. Ως εκ τούτου, ο ρυθμός σύγκλισής του, ο χρόνος υπολογισμού και ο χρόνος επικοινωνίας του είναι αρκετά μεγαλύτεροι σε σύγκριση με τις άλλες δύο μεθόδους.

Gauss-Siedel SOR & Red-Black SOR

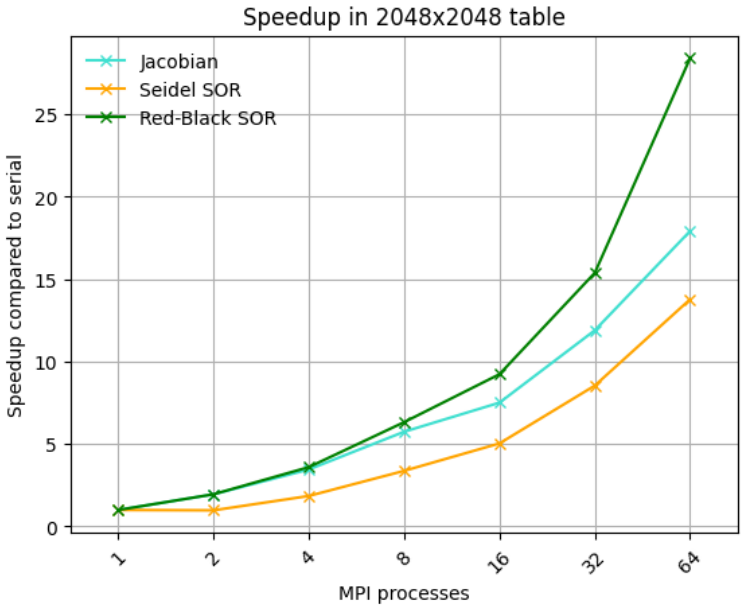
Οι δυο αυτές μέθοδοι μοιάζουν αρκετά στην υλοποίηση. Η μόνη διαφορά τους έγκειται στον χώρο ο σύγκλισής και στον χρόνο επανάληψης. Πιο ειδικά, ο Gauss-Siedel SOR πραγματοποιεί γρηγορότερος υπολογισμούς και απαιτεί μικρότερο χρόνο για επικοινωνία. Από την άλλη, η Red-Black SOR συγκλίνει αρκετά πιο γρήγορα, μιας και απαιτεί λιγότερες επαναλήψεις. Συνολικά παρατηρούμε ότι η μέθοδος Red-Black SOR φαίνεται να υπερισχύει, πράγμα που την καθιστά την γρηγορότερη από τις 3 μεθόδους.

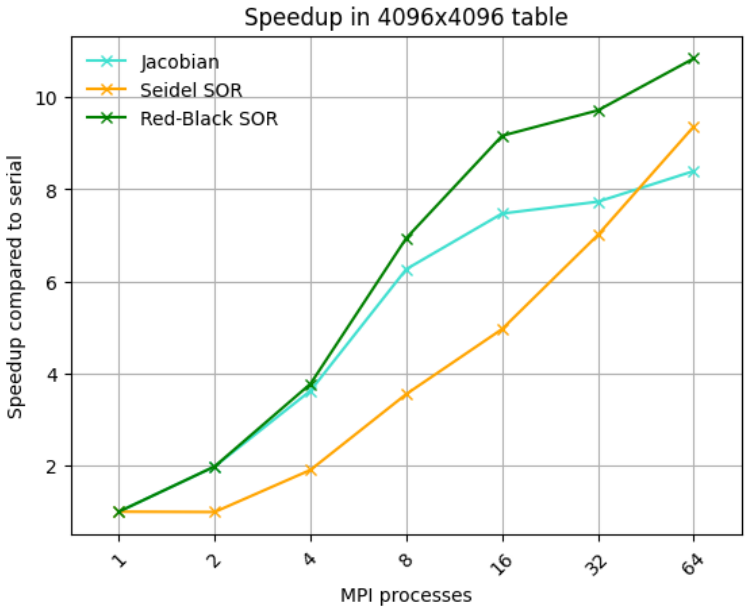
Τέλος, αξίζει να σημειωθεί ότι σημαντικός παράγοντας είναι και το δίκτυο διασύνδεσης. Ειδικότερα, βλέπουμε ότι ο χρόνος επικοινωνίας επιφέρει τις μεγαλύτερες καθυστερήσεις σε κάθε εκτέλεση. Αν και στην συγκεκριμένη αρχιτεκτονική φαίνεται ότι η μέθοδος Red-Black SOR λειτουργεί αποδοτικότερα, δεν σημαίνει πως πάντα θα ισχύει αυτό. Αν το δίκτυο διασύνδεσης είχε περισσότερη κίνηση ενδέχεται η μέθοδος Gauss-Siedel SOR να ήταν πιο γρήγορη μιας και χρειάζεται λιγότερο χρόνο για επικοινωνία.

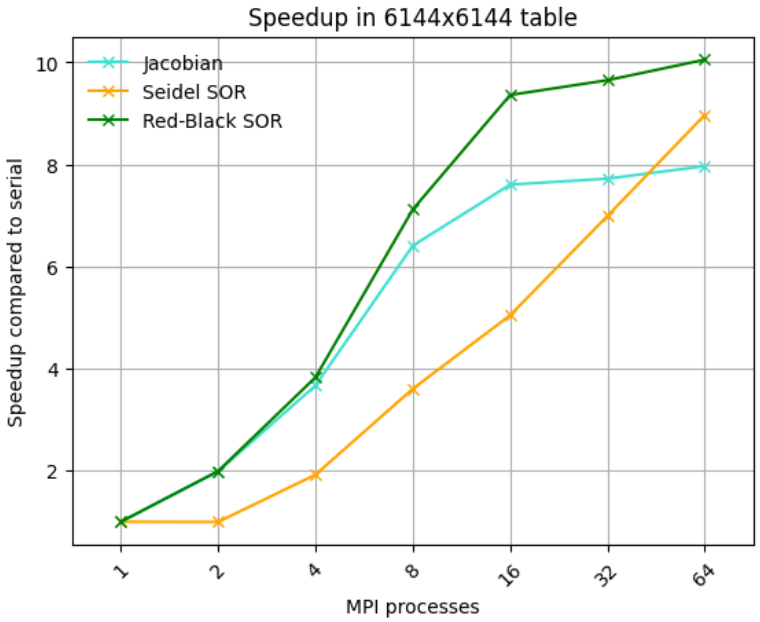
Μετρήσεις χωρίς έλεγχο σύγκλισης

Θα πραγματοποιήσαμε τώρα μετρήσεις για το συνολικό χρόνο και το χρόνο υπολογισμών, απενεργοποιώντας τον έλεγχο σύγκλισης, για σταθερό αριθμό επαναλήψεων (T=256), για μεγέθη πίνακα 2048×2048, 4096×4096, 6144×6144, για 1, 2, 4, 8, 16, 32 και 64 MPI διεργασίες.

- Σχεδιάζουμε ένα διάγραμμα επιτάχυνσης για κάθε μέγεθος πίνακα α (x-axis: MPI διεργασίες, y- που απεικονίζει τις τρεις μεθόδους.



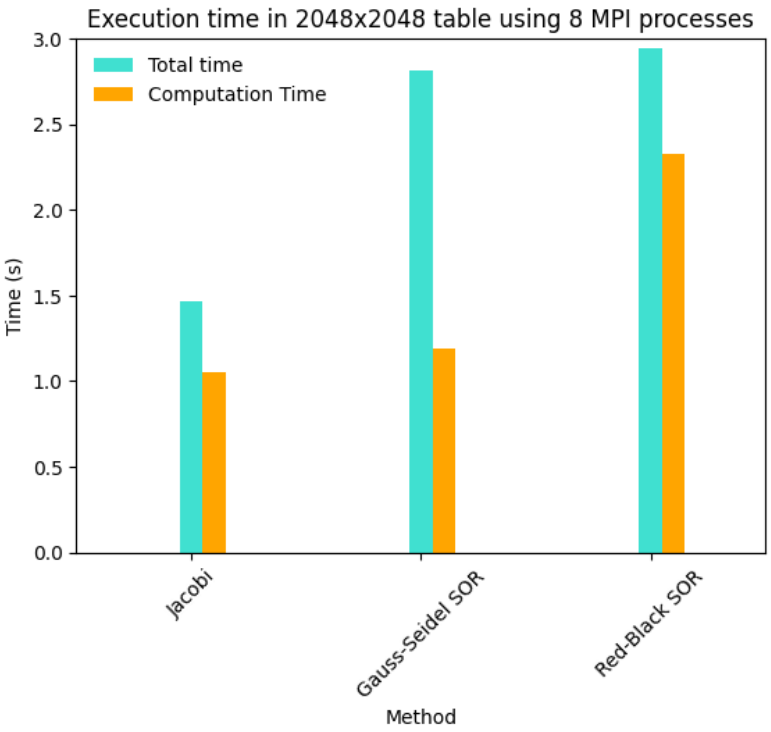


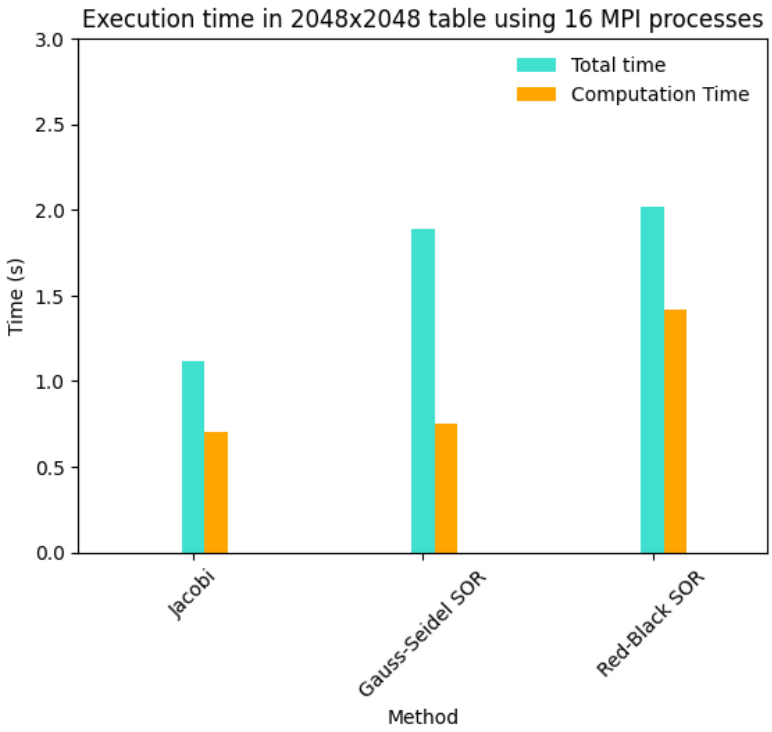


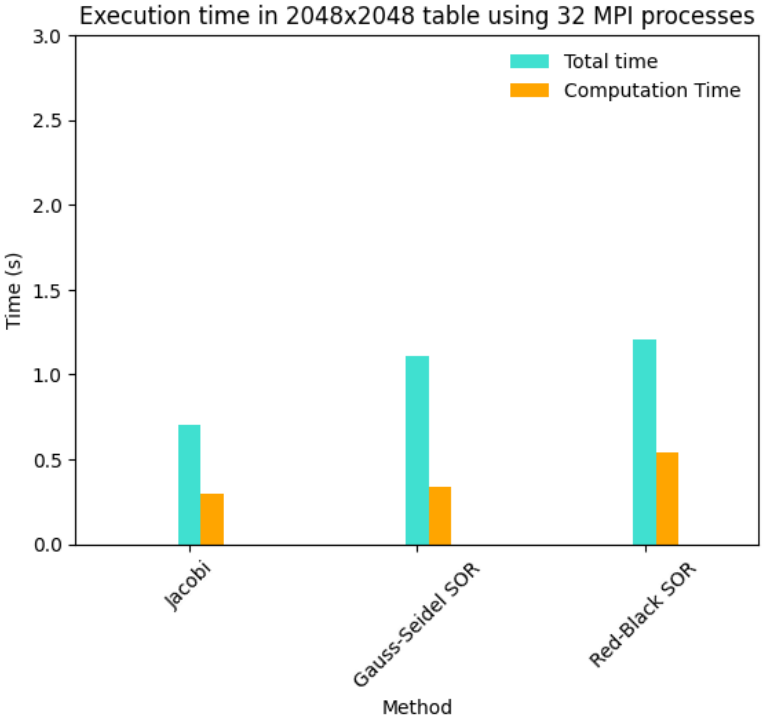
- Κατασκευάζουμε διαγράμματα με μπάρες (1 για κάθε μέγεθος πίνακα και αριθμό επεξεργαστών) που θα απεικονίζουν το συνολικό χρόνο εκτέλεσης και το χρόνο υπολογισμού για κάθε μία από τις τρεις μεθόδους (x-axis: μέθοδος, y-axis: χρόνος).

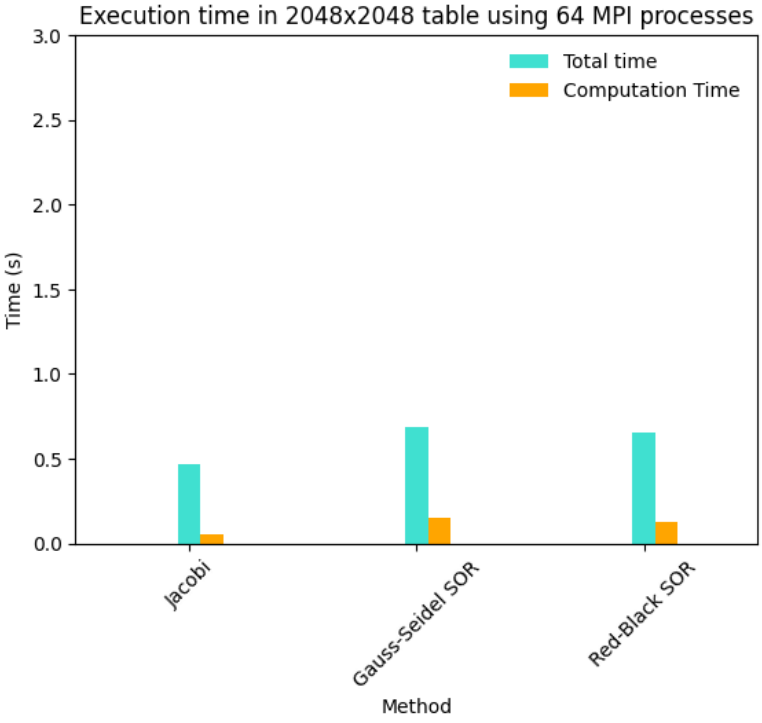
  Για λόγους καλύτερης εποπτείας, κατασκευάζουμε διαγράμματα μόνο για 8, 16, 32 και 64 MPI διεργασίες και κρατάμε κοινή κλίμακα στον άξονα y ανά μέγεθος πίνακα.

2048×2048

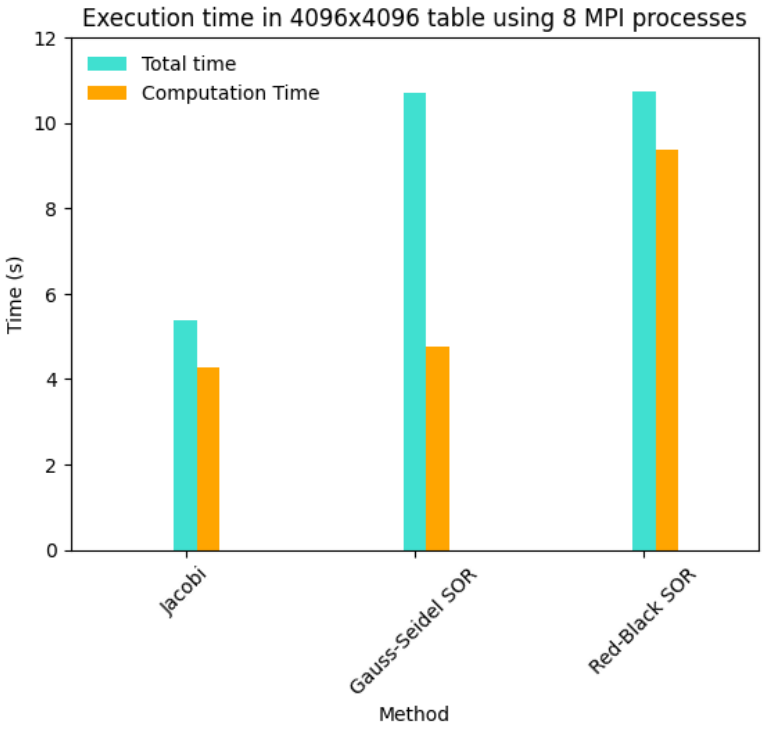

Execution time in 2048x2048 table using 8 MPI processes
Total time
Computation Time
Time (s)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

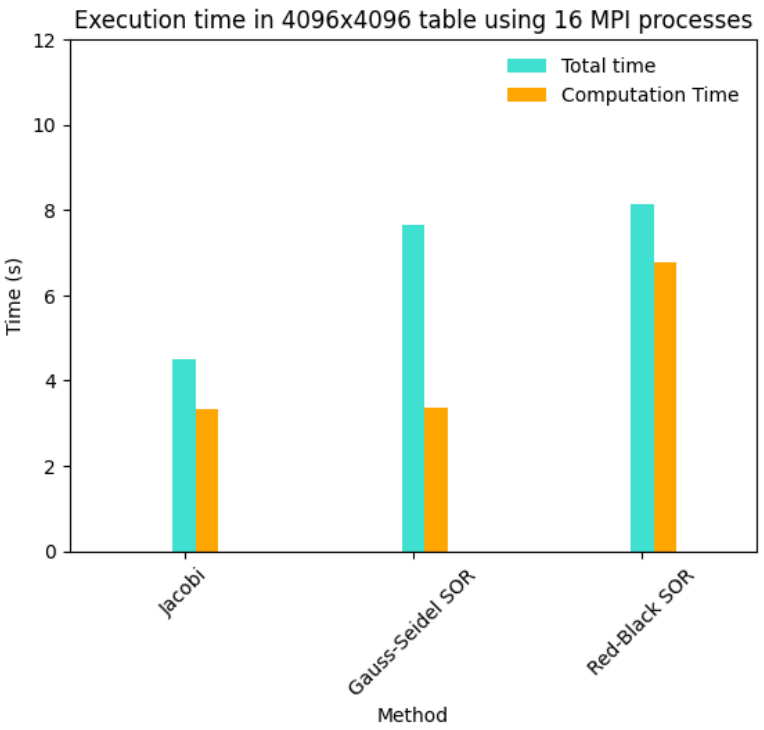

Execution time in 2048x2048 table using 16 MPI processes
Total time
Computation Time
Time (s)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

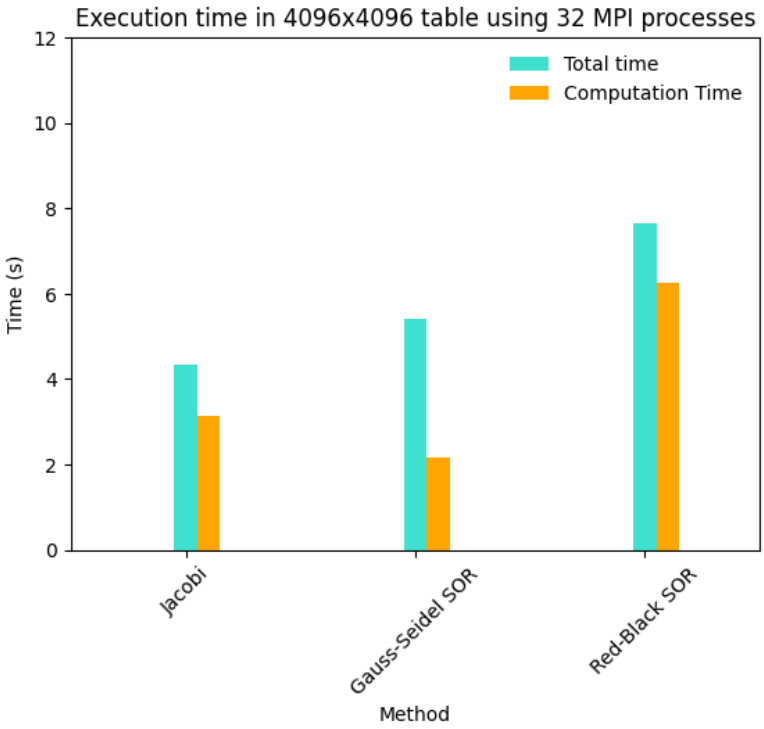

Execution time in 2048x2048 table using 32 MPI processes
Total time
Computation Time
Time (s)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

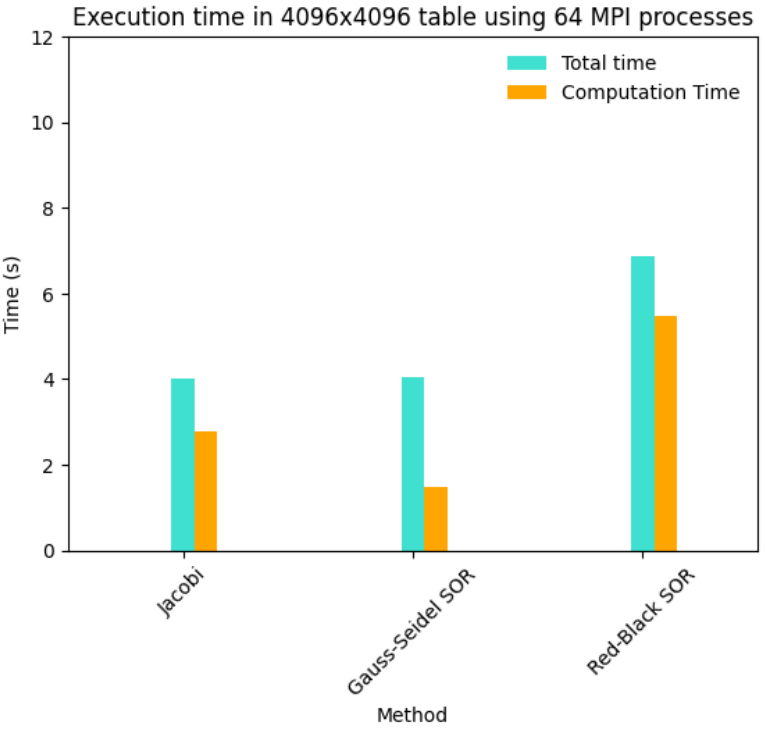

Execution time in 2048x2048 table using 64 MPI processes
Total time
Computation Time
Time (s)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

4096×4096

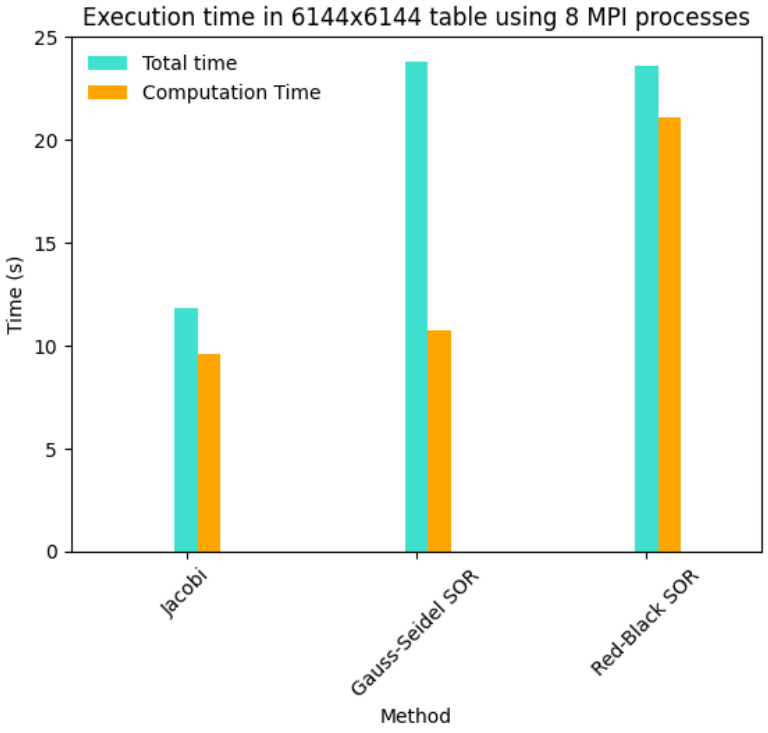

Execution time in 4096x4096 table using 8 MPI processes
Total time
Computation Time
Time (s)
0
2
4
6
8
10
12
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

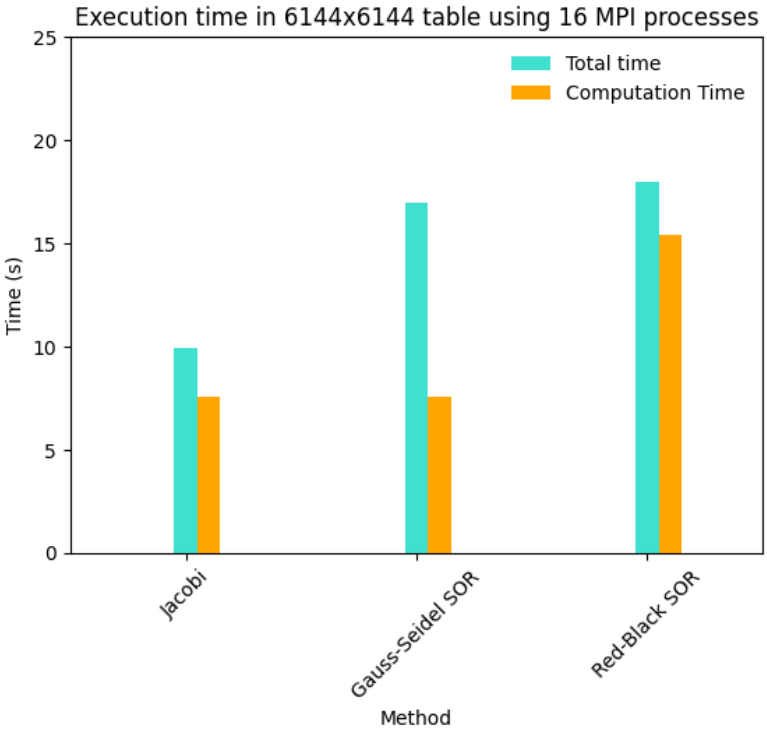

Execution time in 4096x4096 table using 16 MPI processes
Total time
Computation Time
Time (s)
0
2
4
6
8
10
12
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

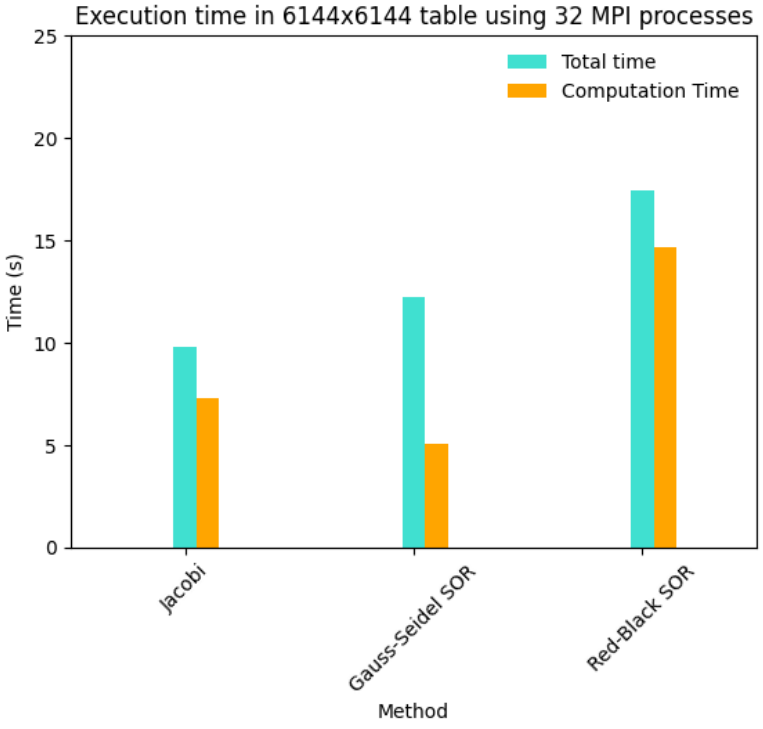

Execution time in 4096x4096 table using 32 MPI processes
Total time
Computation Time
Time (s)
0
2
4
6
8
10
12
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

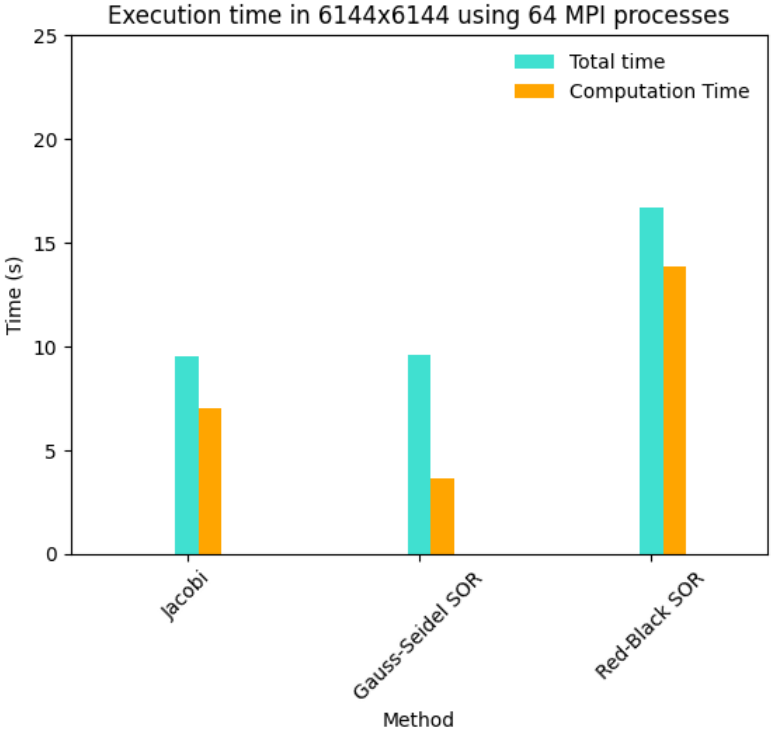

Execution time in 4096x4096 table using 64 MPI processes
Total time
Computation Time
Time (s)
0
2
4
6
8
10
12
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

6144×6144


Execution time in 6144x6144 table using 8 MPI processes
Total time
Computation Time
Time (s)
0
5
10
15
20
25
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method


Execution time in 6144x6144 table using 16 MPI processes
Total time
Computation Time
Time (s)
0
5
10
15
20
25
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method


Execution time in 6144x6144 table using 32 MPI processes
Total time
Computation Time
Time (s)
0
5
10
15
20
25
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method


Execution time in 6144x6144 using 64 MPI processes
Total time
Computation Time
Time (s)
0
5
10
15
20
25
Jacobi
Gauss-Seidel SOR
Red-Black SOR
Method

Από τα παραπάνω διαγράμματα προκύπτουν οι εξής παρατηρήσεις:

Είναι φανερό πως η μέθοδος Jacobi είναι ταχύτερη από τις άλλες δύο μεθόδους όταν έχουμε σταθερό αριθμό επαναλήψεων. Αυτό συμβαίνει διότι έχει μικρότερο υπολογιστικό κόστος και μικρότερο χρόνο επικοινωνίας συγκριτικά με τις μεθόδους Gauss-Siedel και Red-Black. Επιπρόσθετα, παρατηρούμε ότι για μέγεθος πίνακα 2048×2048, όλες οι μέθοδοι έχουν πολύ καλή κλιμάκωση, ενώ για μεγέθη καλύτερα από την Jacobi, ειδικά η Gauss-Siedel η οποία στα τελευταία bar plots φτάνει σχεδόν τον χρόνο της Jacobi, παρά τον μεγαλύτερο χρόνο επικοινωνίας που έχει. Πρέπει να σημειωθεί ακόμα ότι η Red-Black έχει σχεδόν διπλάσιο υπολογιστικό κόστος από τις άλλες δύο μεθόδους, γεγονός που εμποδίζει την περεταίρω κλιμάκωσή της, αν και παρουσιάζει μεγαλύτερο speedup.

Συνολικά, λοιπόν, ο Gauss-Seidel SOR, αν και έχει μεγάλο κόστος επικοινωνίας, κλιμακώνει καλύτερα μειώνοντας συνεχώς το συνολικό του χρόνο, κάτι που οφείλεται σίγουρα και στην non-blocking διαχείριση της επικοινωνίας. Από την άλλη, ο Red-Black SOR και ο Jacobi δεν κλιμακώνουν το ίδιο καλά μιας και τα blocking SendRecv δεν διευκολύνουν τη μείωση του χρόνου επικοινωνίας.