3η Εργαστηριακή Άσκηση

Ομάδα 01

Μπουγουλιάς Παναγιώτης 03112025
Χορτομάνης Ιωάννης 03110825

Αποτελέσματα των μετρήσεων για τα 3 μεγέθη πινάκων:

(χωρίς έλεγχο σύγκλισης όπως και τα 15 επόμενα διαγράμματα)

2048x2048

Αριθμός Πυρήνων	Χαρακτηριστικά Προσο Μοίωσης	Total Time Jacobi	Επιτάχυνση Jacobi	Communication Time Jacobi	Computation Time Jacobi	Total Time Gauss-Seidel SOR	Επιτάχυνση Gauss-Seidel SOR	Communication Time Gauss-Seidel SOR	Computation Time Gauss-Seidel SOR	Total Time Red-Black SOR	Επιτάχυνση Red-Black SOR	Communication Time Red-Black SOR	Computation Time Red-Black SOR
1	Serial	8,401	1,000			12,095	1,000			18,598			
1	1 MPI διεργασία	8,401	1,000			12,095	1,000			18,598	1,000		
2	2 MPI διεργασίες	4,369	1,923			12,338	0,980			9,529	1,952		
4	4 MPI διεργασίες	2,379	3,532			6,464	1,871			5,028	3,699		
8	8 MPI διεργασίες	1,396	6,020	0,334	1,061	3,496	3,459	1,982	1,516	2,774	6,705	0,451	2,329
16	16 MPI διεργασίες	1,139	7,373	0,523	0,839	2,079	5,817	1,295	0,837	1,802	10,318	0,497	1,353
32	32 MPI διεργασίες	0,696	12,078	0,534	0,368	1,239	9,762	0,849	0,405	1,084	17,161	0,536	0,608
64	64 MPI διεργασίες	0,388	21,652	0,336	0,059	0,769	15,730	0,579	0,191	0,545	34,097	0,419	0,128
Communication Time	Computation Time												
Jacobi	Gauss-Seidel SOR	Red-Black SOR											

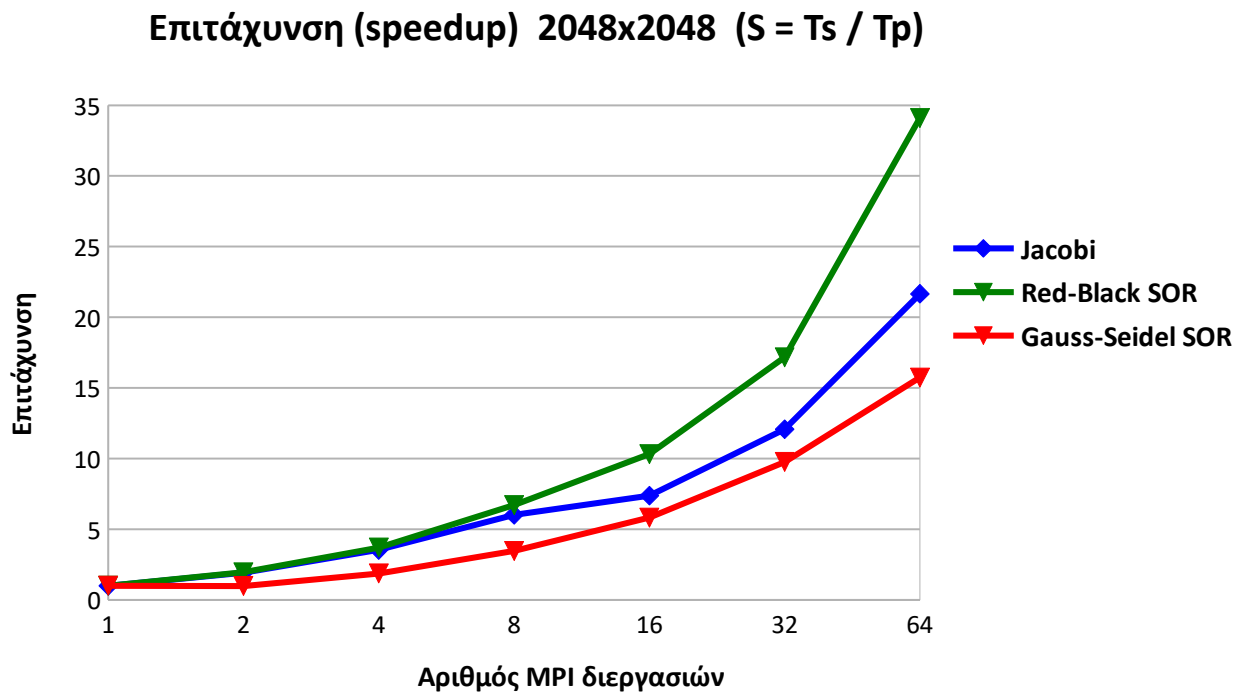
4096x4096

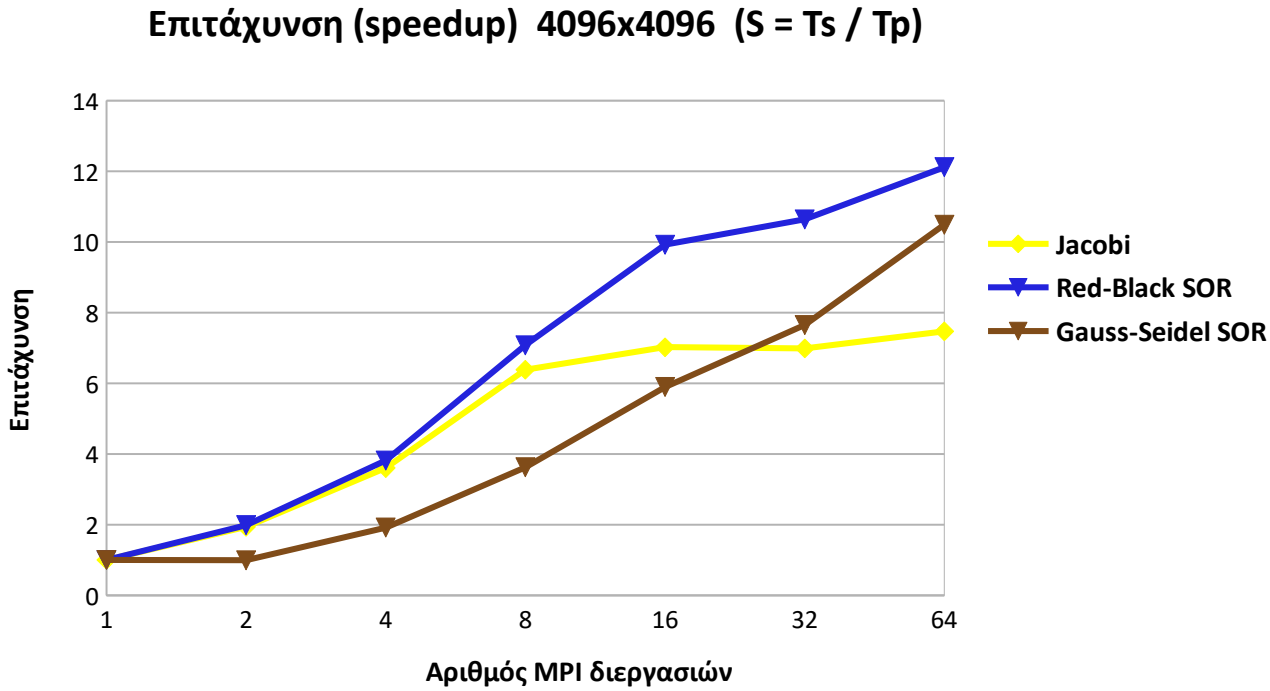
Αριθμός Πυρήνων	Χαρακτηριστικά Προσο Μοίωσης	Total Time Jacobi	Επιτάχυνση Jacobi	Communication Time Jacobi	Computation Time Jacobi	Total Time Gauss-Seidel SOR	Επιτάχυνση Gauss-Seidel SOR	Communication Time Gauss-Seidel SOR	Computation Time Gauss-Seidel SOR	Total Time Red-Black SOR	Επιτάχυνση Red-Black SOR	Communication Time Red-Black SOR	Computation Time Red-Black SOR
1	Serial	33,549	1,000			48,367	1,000			74,416			
1	1 MPI διεργασία	33,549	1,000			48,367	1,000			74,416	1,000		
2	2 MPI διεργασίες	17,302	1,939			48,756	0,992			37,542	1,982		
4	4 MPI διεργασίες	9,312	3,603			25,238	1,916			19,509	3,814		
8	8 MPI διεργασίες	5,256	6,383	0,938	4,318	13,349	3,623	7,273	6,076	10,516	7,076	1,205	9,329
16	16 MPI διεργασίες	4,778	7,022	1,934	3,722	8,204	5,895	4,974	3,709	7,500	9,922	1,379	6,285
32	32 MPI διεργασίες	4,800	6,989	3,232	3,632	6,321	7,652	4,534	3,052	6,994	10,639	2,834	5,714
64	64 MPI διεργασίες	4,490	7,471	3,974	3,200	4,611	10,489	3,818	1,712	6,142	12,117	4,809	4,834
Communication Time	Computation Time												
Jacobi	Gauss-Seidel SOR	Red-Black SOR											

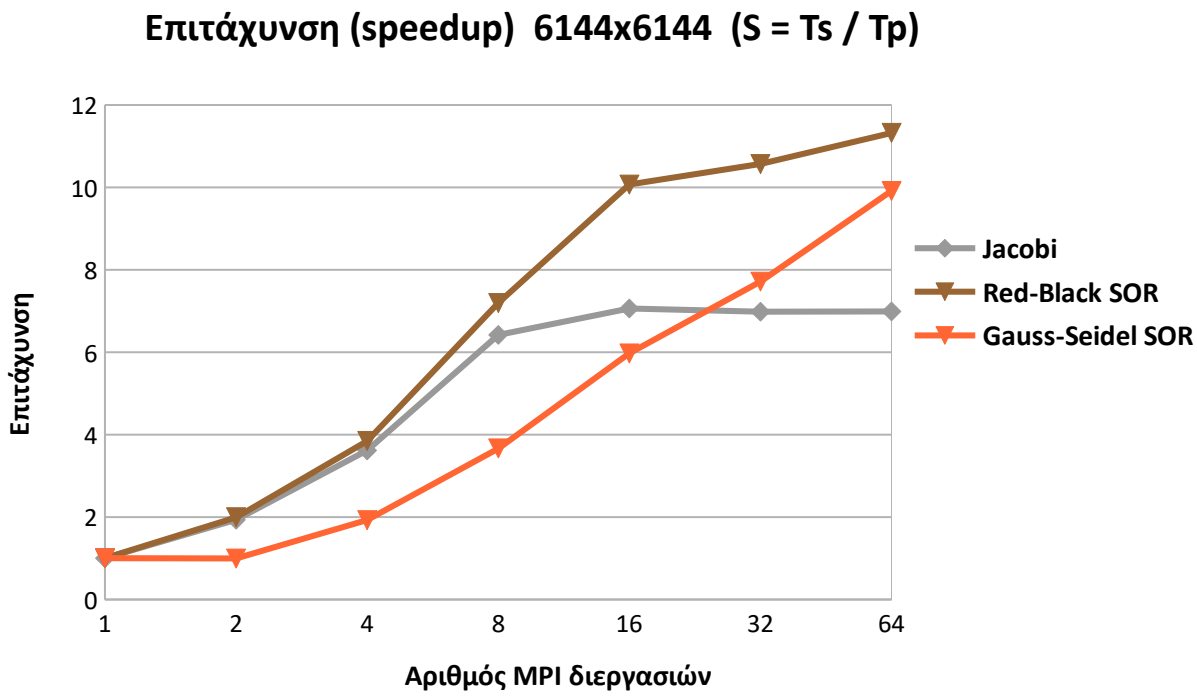
6144x6144

Αριθμός Πυρήνων	Χαρακτηριστικά Προσο Μοίωσης	Total Time Jacobi	Επιτάχυνση Jacobi	Communication Time Jacobi	Computation Time Jacobi	Total Time Gauss-Seidel SOR	Επιτάχυνση Gauss-Seidel SOR	Communication Time Gauss-Seidel SOR	Computation Time Gauss-Seidel SOR	Total Time Red-Black SOR	Επιτάχυνση Red-Black SOR	Communication Time Red-Black SOR	Computation Time Red-Black SOR
1	Serial	75,278	1,000			108,821	1,000			167,898			
1	1 MPI διεργασία	75,278	1,000			108,821	1,000			167,898	1,000		
2	2 MPI διεργασίες	38,865	1,937			109,531	0,994			84,249	1,993		
4	4 MPI διεργασίες	20,829	3,614			56,526	1,925			43,654	3,846		
8	8 MPI διεργασίες	11,728	6,419	2,027	9,700	29,699	3,664	16,031	13,668	23,334	7,195	2,394	20,973
16	16 MPI διεργασίες	10,664	7,059	4,210	8,390	18,222	5,972	10,936	8,383	16,678	10,067	2,844	14,214
32	32 MPI διεργασίες	10,780	6,983	7,888	8,307	14,105	7,715	10,080	6,954	15,889	10,567	6,806	13,220
64	64 MPI διεργασίες	10,769	6,990	9,357	7,874	10,976	9,914	9,044	4,231	14,832	11,320	12,338	12,073
Communication Time	Computation Time												
Jacobi	Gauss-Seidel SOR	Red-Black SOR											

Speedup diagrams







Τα προγράμματα εκτελέστηκαν για 256 επαναλήψεις οι οποίες είναι βέβαια ενδεικτικές για την συμπεριφορά των αλγορίθμων αλλά ίσως δεν είναι απόλυτα αντιπροσωπευτικές για να φανούν σε όλη τους την έκταση οι διαφορές των αλγορίθμων.

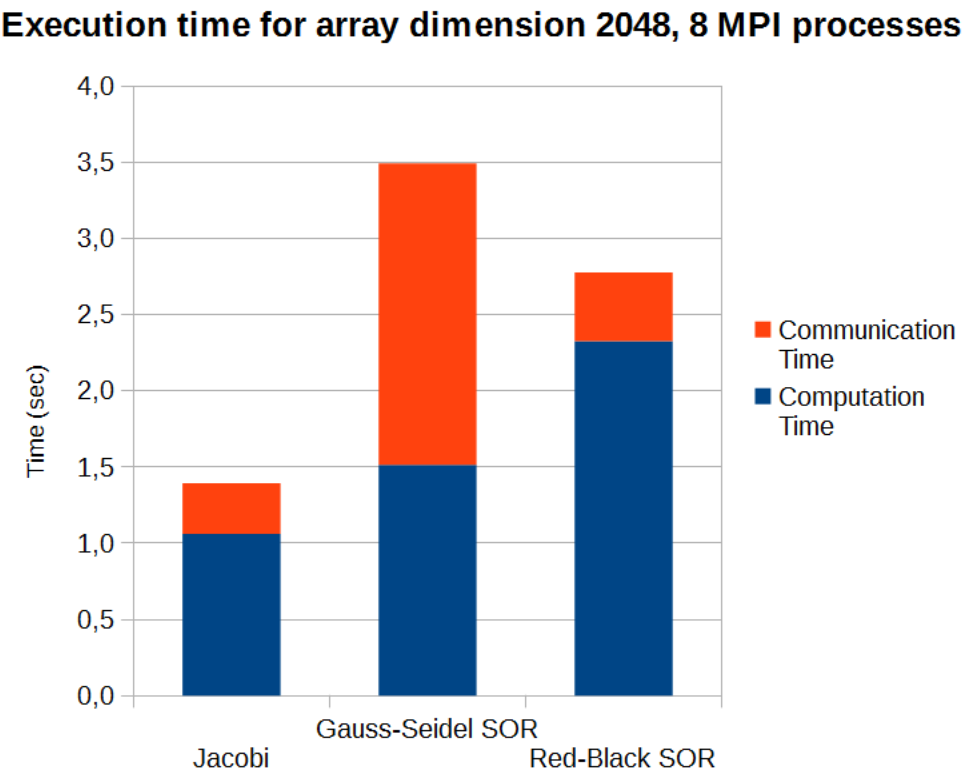
Παρατηρούμε ότι στο μοντέλο κατανεμημένης μνήμης με ανταλλαγή μηνυμάτων τα προγράμματα κλιμακώνουν και στους 3 αλγορίθμους με φθίνον ρυθμό βέβαια (ιδιαίτερα εμφανής στην περίπτωση του Jacobi).

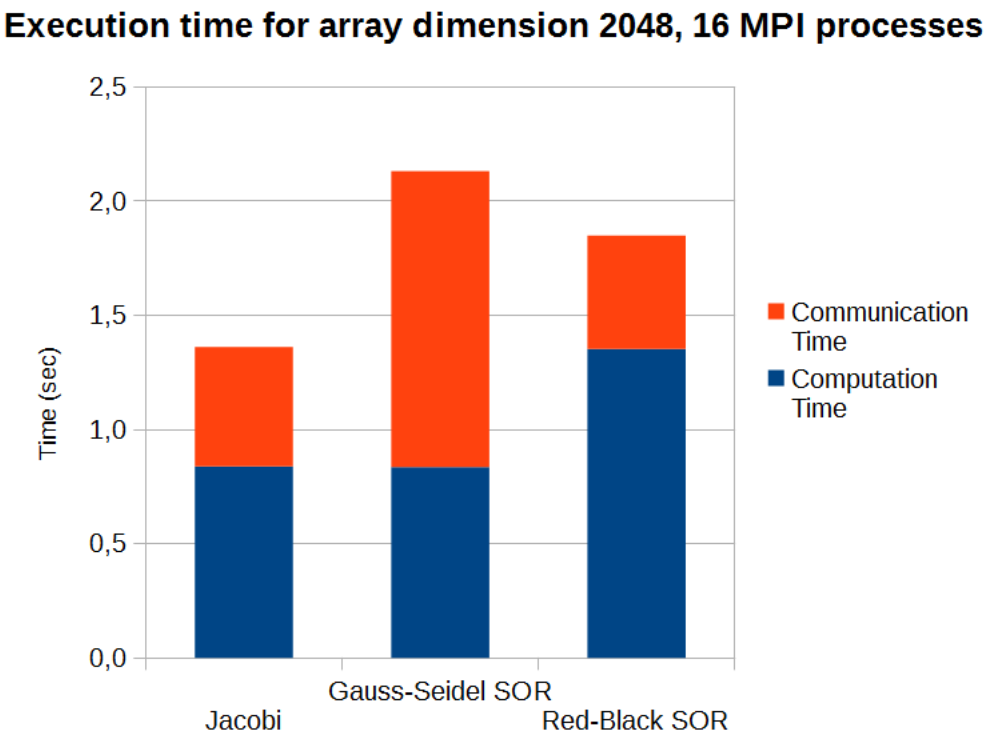
Ο Red-Black SOR πετυχαίνει το μέγιστο speedup από όλους τους αλγορίθμους για οποιονδήποτε αριθμό MPI διεργασιών.

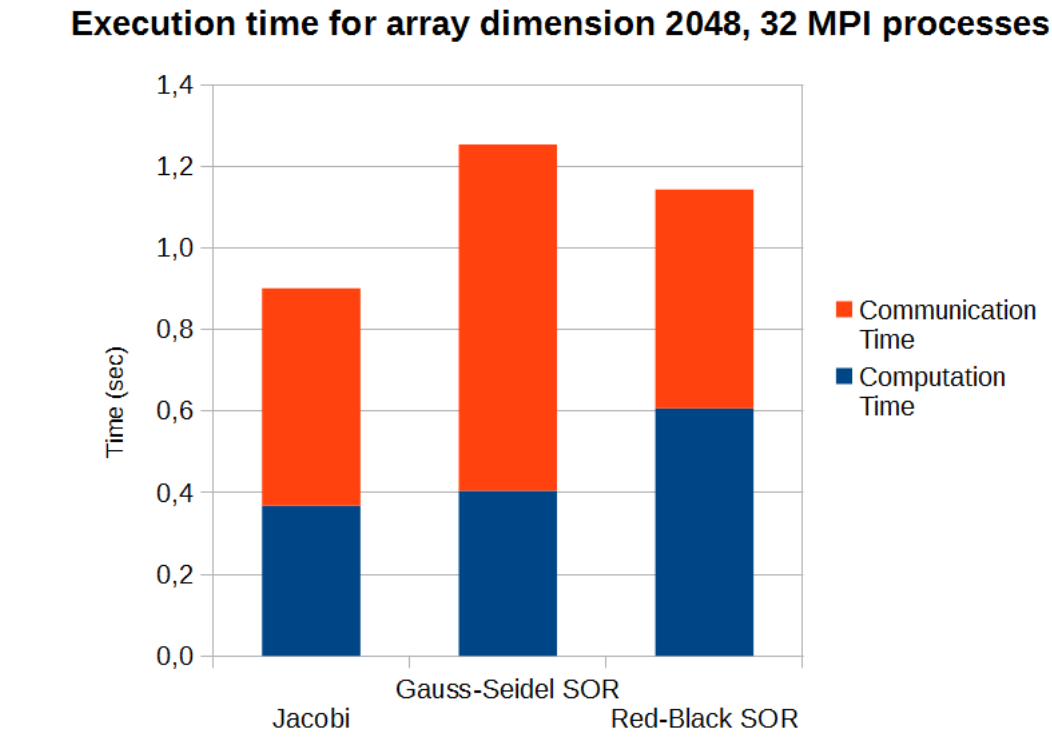
Παρατηρούμε ότι ο ρυθμός ανόδου αυξάνεται στον Gauss Seidel όσο αυξάνεται ο αριθμός των MPI διεργασιών για όλα τα μεγέθη πινάκων.

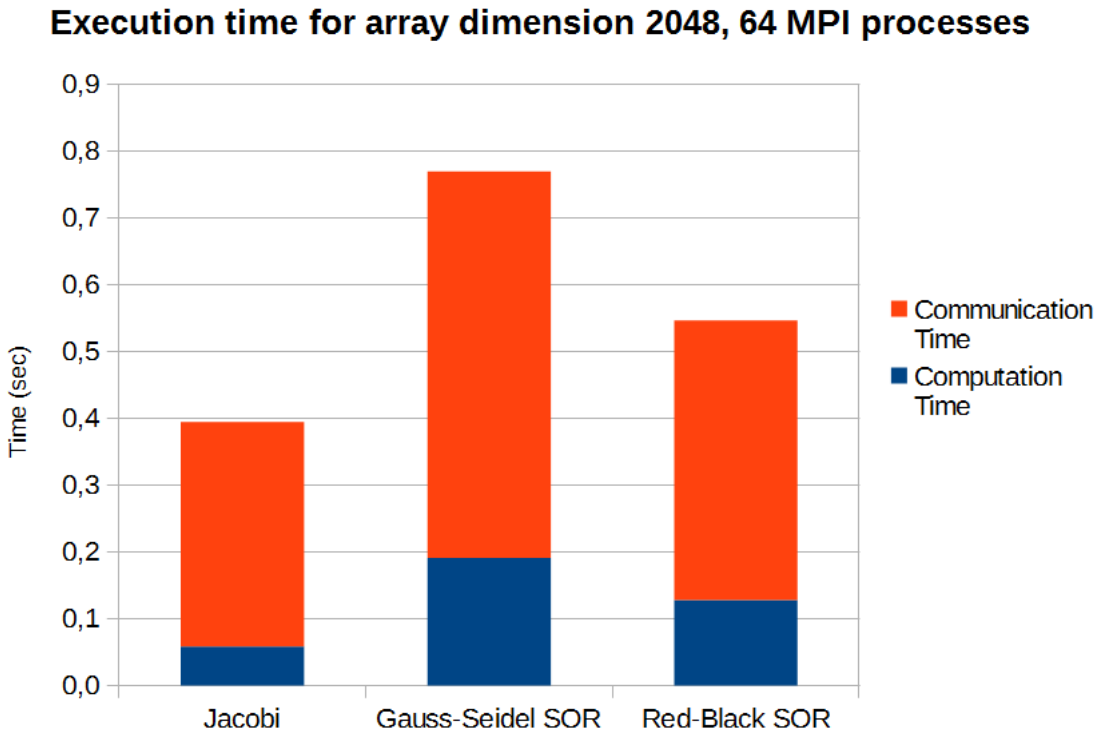
Στην περίπτωση του Jacobi παρατηρείται ότι για μεγάλα μεγέθη πινάκων υπάρχει απότομη κάμψη της αύξησης της επιτάχυνσης από έναν αριθμό διεργασιών και πάνω.

Bar diagrams

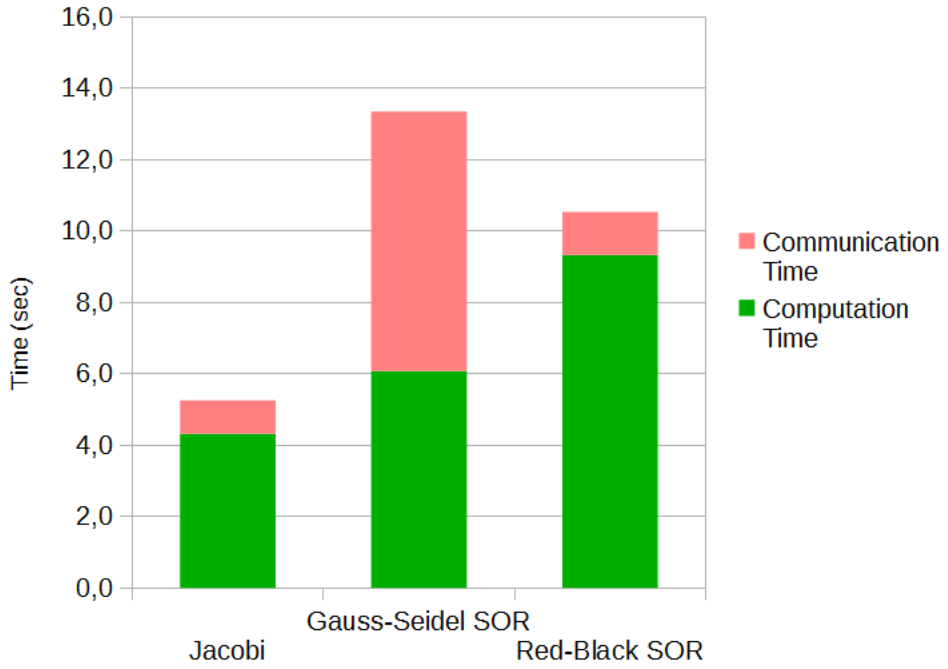




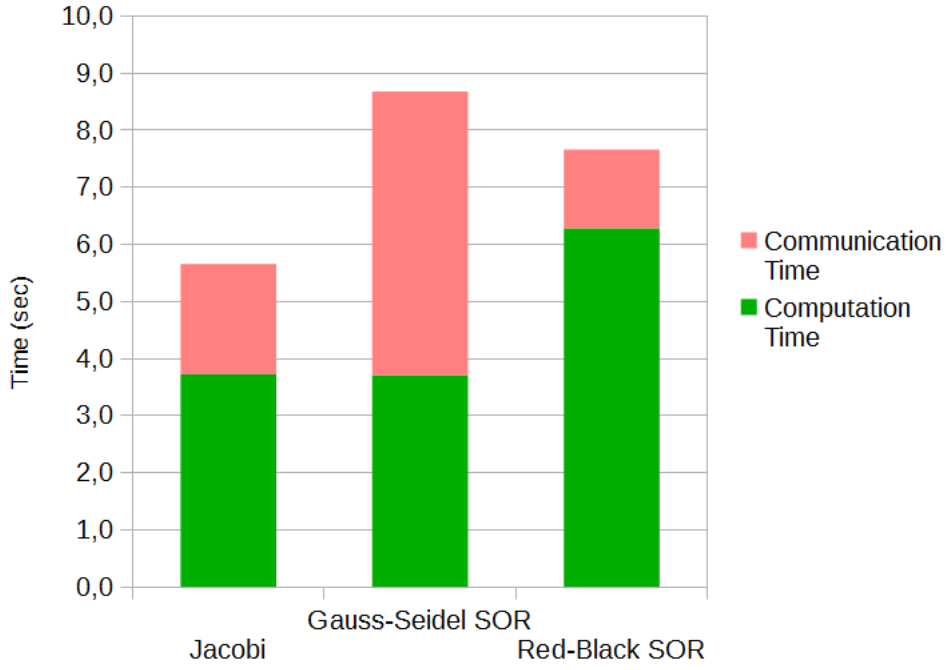

Execution time for array dimension 2048, 32 MPI processes
1,4
1,2
1,0
0,8
0,6
0,4
0,2
0,0
Time (sec)
Communication Time
Computation Time
Jacobi
Gauss-Seidel SOR
Red-Black SOR


Execution time for array dimension 2048, 64 MPI processes
0,9
0,8
0,7
0,6
0,5
0,4
0,3
0,2
0,1
0,0
Time (sec)
Communication Time
Computation Time
Jacobi
Gauss-Seidel SOR
Red-Black SOR

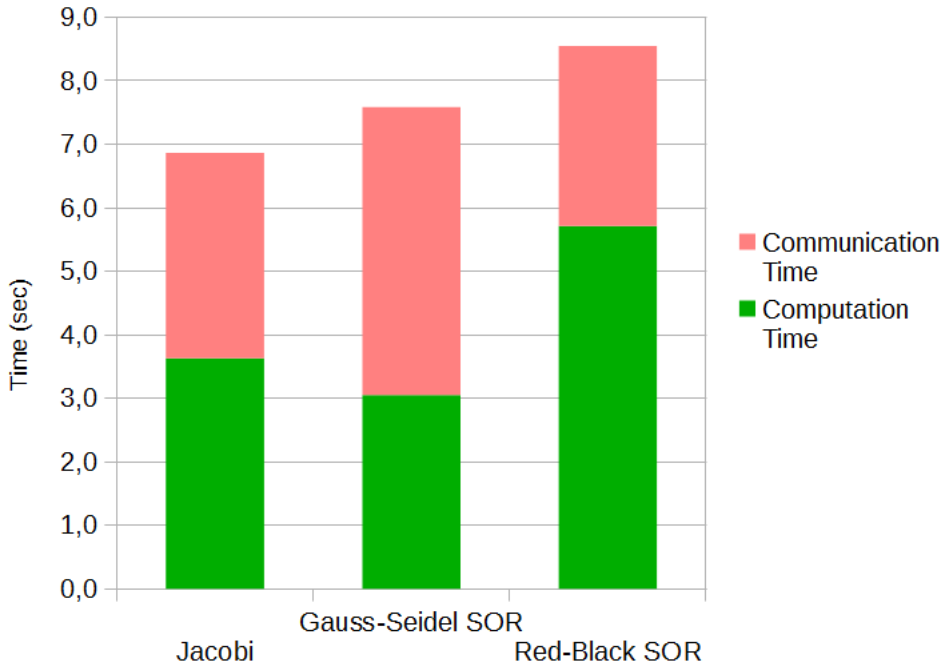
Execution time for array dimension 4096, 8 MPI processes



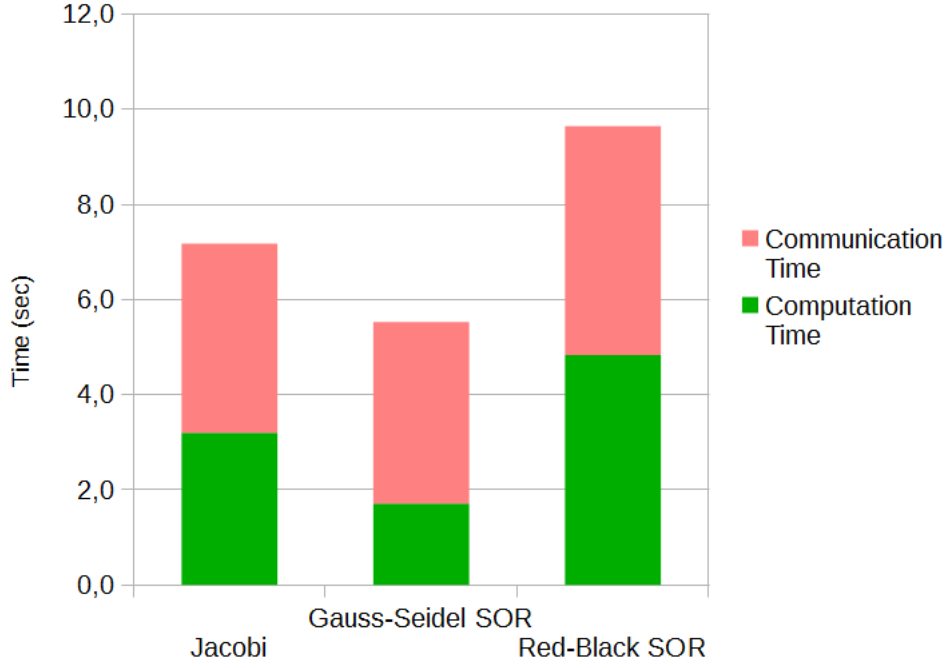
Execution time for array dimension 4096, 16 MPI processes



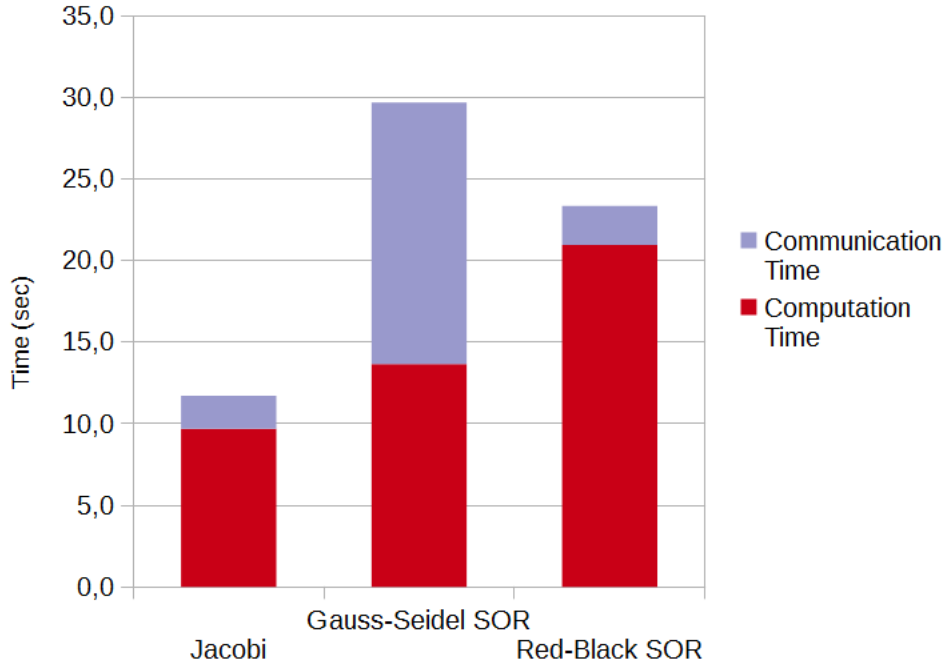
Execution time for array dimension 4096, 32 MPI processes



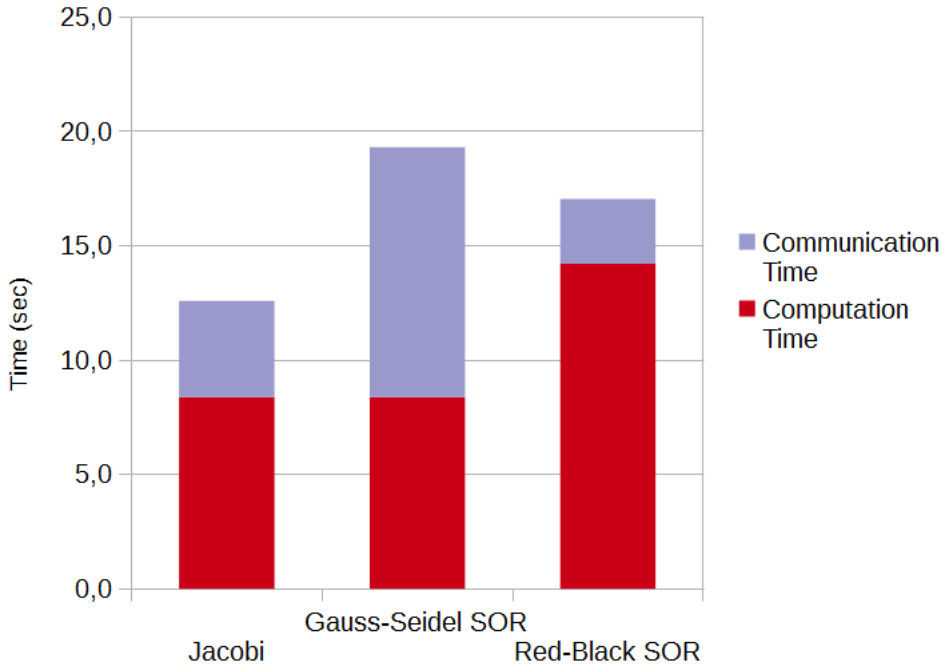
Execution time for array dimension 4096, 64 MPI processes



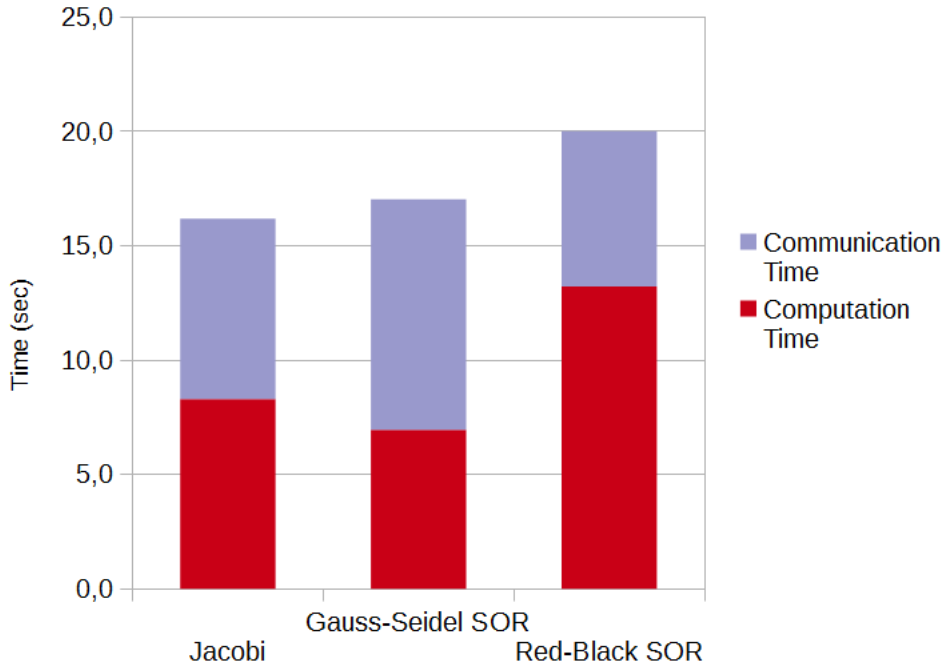
Execution time for array dimension 6144, 8 MPI processes



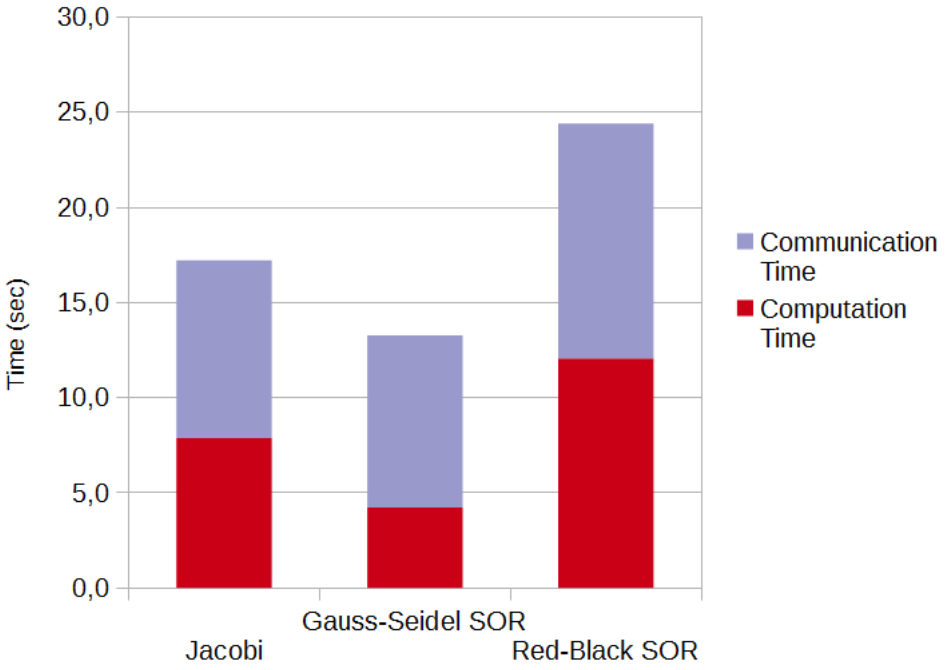
Execution time for array dimension 6144, 16 MPI processes



Execution time for array dimension 6144, 32 MPI processes



Execution time for array dimension 6144, 64 MPI processes



Κατ' αρχήν παρατηρούμε πως γενικά τους καλύτερους χρόνους τους πετυχαίνει ο αλγόριθμος Jacobi. Αυτό δεν πρέπει να παρερμηνευθεί καθώς το ότι ο συγκεκριμένος αλγόριθμος πέτυχε τον μικρότερο χρόνο για τον συγκεκριμένο αριθμό επαναλήψεων δεν σημαίνει πως ο αλγόριθμος αυτός στον χρόνο αυτό πραγματοποίησε και το μεγαλύτερο ποσοστό προόδου όσον αφορά την σύγκλισή του στην τελική λύση (όπως φαίνεται και από το σχετικό ερώτημα με τον έλεγχο σύγκλισης, ο αλγόριθμος Jacobi απαιτεί πολύ περισσότερες επαναλήψεις – άρα και χρόνο – για να συγκλίνει σε σχέση με τους υπόλοιπους).

Σε όλες τις περιπτώσεις παρατηρείται το (αναμενόμενο) χαρακτηριστικό του να αυξάνεται το ποσοστό του χρόνου επικοινωνίας (στο σύνολο του χρόνου εκτέλεσης) συναρτήσει του αριθμού των διεργασιών, ενώ αντιστοίχως μειώνεται το ποσοστό του χρόνου υπολογισμών. Αυτό είναι φυσιολογικό καθώς με την αύξηση του αριθμού των MPI διεργασιών (άρα και των επεξεργαστών που εμπλέκονται στους υπολογισμούς) υπάρχει μεγαλύτερη ανάγκη για επικοινωνία μεταξύ τους καθώς οι υπολογισμοί κατανέμονται σε περισσότερους “εργάτες”.

Επίσης είναι φανερό πως ο συνολικός χρόνος υπολογισμών που απαιτείται μειώνεται με την αύξηση του αριθμού των διεργασιών-επεξεργαστών. Αυτά είναι γενικά συμπεράσματα που ισχύουν και για τους τρεις αλγόριθμους.

Μετρήσεις με έλεγχο σύγκλισης

parlab01@clone1:~/mpi/ask3_par$ mpirun --mca btl tcp,self -np 64 --map-by node ./redblacksor_par 1024 1024 8 8
RedBlackSOR X 1024 Y 1024 Px 8 Py 8 Iter 2501 CommunicationTime 1.163045 ComputationTime 0.307194 TotalTime **1.486874** midpoint 5.642974
parlab01@clone1:~/mpi/ask3_par$ mpirun --mca btl tcp,self -np 64 --map-by node ./seidelsor_par 1024 1024 8 8
GaussSeidel X 1024 Y 1024 Px 8 Py 8 Iter 3201 CommunicationTime 1.516756 ComputationTime 0.603248 TotalTime **2.179169** midpoint 5.642998
parlab01@clone1:~/mpi/ask3_par$ mpirun --mca btl tcp,self -np 64 --map-by node ./jacobi_par 1024 1024 8 8
Jacobi X 1024 Y 1024 Px 8 Py 8 Iter 798201 CommunicationTime 178.432104 ComputationTime 39.779461 TotalTime **221.683259** midpoint 5.431022

Παρατηρούμε ότι και ο Gauss-Seidel SOR και ο Red-Black SOR συγκλίνουν σχετικά πιο γρήγορα από τον Jacobi, ο οποίος χρειάζεται δυο τάξεις μεγέθους παραπάνω επαναλήψεις από τους άλλους δύο για να συγκλίνει στο αποτέλεσμα που πρέπει.

Ο πιο γρήγορος αλγόριθμος είναι ο Red-Black SOR αφού συγκλίνει πιο γρήγορα από τον Gauss-Seidel. Επίσης είναι αυτός που θα προτιμούσαμε στο μοντέλο κατανεμημένης μνήμης για παραλληλοποίηση, αφού εκτός από τη φυσική του ταχύτητα, έχει λιγότερες απαιτήσεις συγχρονισμού από τον Gauss-Seidel.

Λόγω μεγάλης έκτασης των προγραμμάτων παρατίθεται ενδεικτικά μόνο ο κώδικας από το Red-Black SOR.

redblack_sor.c

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include "mpi.h"
#include "utils.h"


void RedSOR(double ** u_previous, double ** u_current, int X_min, int X_max, int Y_min, int Y_max, double omega) {
	int i,j;
	for (i=X_min;i<X_max;i++)
		for (j=Y_min;j<Y_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]);
}

void BlackSOR(double ** u_previous, double ** u_current, int X_min, int X_max, int Y_min, int Y_max, double omega) {
	int i,j;
	for (i=X_min;i<X_max;i++)
		for (j=Y_min;j<Y_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]);
}



int main(int argc, char ** argv) {
   int rank,size;
   int global[2],local[2]; //global matrix dimensions and local matrix dimensions (2D-domain, 2D-subdomain)
      int global_padded[2];     //padded global matrix dimensions (if padding is not needed, global_padded=global)
   int grid[2];            //processor grid dimensions
   int padded[2] = {0,0};
```

```
int i,j,t;
  int global_converged=0, converged=0; //flags for convergence, global and per process
 MPI_Datatype dummy;      //dummy datatype used to align user-defined datatypes in memory
int omega;
struct timeval totals,totalf,comps,compf,comms,commf;
double ttotal=0,tcomp=0,tcomm=0,total_time,comp_time,comm_time;

double ** U = malloc(1), ** u_current, ** u_previous, ** swap;
  //Global matrix, local current and previous matrices, pointer to swap between current and previous


MPI_Init(&argc,&argv);
MPI_Comm_size(MPI_COMM_WORLD,&size);
MPI_Comm_rank(MPI_COMM_WORLD,&rank);

//----Read 2D-domain dimensions and process grid dimensions from stdin----//

if (argc!=5) {
   fprintf(stderr,"Usage: mpirun .... ./exec X Y Px Py");
   exit(-1);
}
else {
   global[0]=atoi(argv[1]);
   global[1]=atoi(argv[2]);
   grid[0]=atoi(argv[3]);
   grid[1]=atoi(argv[4]);
}

omega=2.0/(1+sin(3.14/global[0]));
//----Create 2D-cartesian communicator----//
//----Usage of the cartesian communicator is optional----//

  MPI_Comm CART_COMM;          //CART_COMM: the new 2D-cartesian communicator
int periods[2]={0,0};      //periods={0,0}: the 2D-grid is non-periodic
  int rank_grid[2];          //rank_grid: the position of each process on the new communicator

      MPI_Cart_create(MPI_COMM_WORLD,2,grid,periods,0,&CART_COMM); //communicator creation
```

```
   MPI_Cart_coords(CART_COMM,rank,2,rank_grid);          //rank mapping on the new communicator
                  //printf("rank:%d with rank_grid[0]:%d and rank_grid[1]: %d\n",rank,rank_grid[0],rank_grid[1]);

   //----Compute local 2D-subdomain dimensions----//
   //----Test if the 2D-domain can be equally distributed to all processes----//
   //----If not, pad 2D-domain----//

   for (i=0;i<2;i++) {
      if (global[i]%grid[i]==0) {
         local[i]=global[i]/grid[i];
         global_padded[i]=global[i];
      }
      else {
         local[i]=(global[i]/grid[i])+1;
         global_padded[i]=local[i]*grid[i];
           padded[i] = 1;
      }
   }



   //----Allocate global 2D-domain and initialize boundary values----//
   //----Rank 0 holds the global 2D-domain----//

   if (rank==0) {
        free(U);
      U=allocate2d(global_padded[0],global_padded[1]);
      init2d(U,global[0],global[1]);
   }

   //----Allocate local 2D-subdomains u_current, u_previous----//
   //----Add a row/column on each size for ghost cells----//

   u_current = allocate2d(local[0] + 2, local[1] + 2);
   u_previous = allocate2d(local[0] + 2, local[1] + 2);

   //----Distribute global 2D-domain from rank 0 to all processes----//

   //----Appropriate datatypes are defined here----//

   //----Datatype definition for the 2D-subdomain on the global matrix----//
```

```
MPI_Datatype global_block;
MPI_Type_vector(local[0],local[1],global_padded[1],MPI_DOUBLE,&dummy);
MPI_Type_create_resized(dummy,0,sizeof(double),&global_block);
MPI_Type_commit(&global_block);

//----Datatype definition for the 2D-subdomain on the local matrix----//
//----Note: this datatype assumes that the local matrix is extended
//----      by 2 rows and columns to accomodate received data--------//

MPI_Datatype local_block;
MPI_Type_vector(local[0],local[1],local[1]+2,MPI_DOUBLE,&dummy);
MPI_Type_create_resized(dummy,0,sizeof(double),&local_block);
MPI_Type_commit(&local_block);

//----Rank 0 scatters the global matrix----//
/* Make sure u_current and u_previous are both initialized (not with init2d)*/

int group_size = grid[0] * grid[1];
int * sendcounts = malloc(group_size * sizeof(int));
int * displs = malloc(group_size * sizeof(int));


if (rank == 0) {
     for (i = 0; i < grid[0]; i++) {
             for (j = 0; j < grid[1]; j++) {
                     displs[grid[1]*i + j] = (local[0]*global_padded[1]*i+ local[1]*j);
                    sendcounts[grid[1]*i + j] = 1;
             }
     }
}



 MPI_Scatterv(&(U[0][0]), sendcounts, displs, global_block, &(u_previous[1][1]),
1, local_block, 0, MPI_COMM_WORLD);
   MPI_Scatterv(&(U[0][0]), sendcounts, displs, global_block, &(u_current[1][1]), 1,
local_block, 0, MPI_COMM_WORLD);


if (rank==0)
   free2d(U,global_padded[0],global_padded[1]);
```

```
//----Define datatypes or allocate buffers for message passing----//

//----Datatype definition for the 2D-subdomain on the global matrix----//

MPI_Datatype column;
MPI_Type_vector(local[0] + 2, 1, local[1] + 2 , MPI_DOUBLE, &dummy);
MPI_Type_create_resized(dummy, 0, sizeof(double), &column);
MPI_Type_commit(&column);

MPI_Datatype row;
MPI_Type_contiguous(local[1] + 2, MPI_DOUBLE, &row);
MPI_Type_commit(&row);


//----Find the 4 neighbors with which a process exchanges messages----//

int north, south, east, west;

 MPI_Cart_shift(CART_COMM, 0, 1, &north, &south); // If <0 then there is no such neighbor
 MPI_Cart_shift(CART_COMM, 1, 1, &west, &east);  // Boundary processes and maybe in padding

//printf("--------------------------\n");
                //printf("rank:%d north:%d south:%d east:%d west: %d\n",rank,north,south,east,west);
    /*Make sure you handle non-existing
        neighbors appropriately*/

//---Define the iteration ranges per process-----//

    /*Three types of ranges:
        -internal processes
        -boundary processes
        -boundary processes and padded global array
    */

 int i_min,i_max,j_min,j_max;

    i_min = 1;
    i_max = local[0]+1;
    j_min = 1;
    j_max = local[1]+1;
```

```
if (north < 0) {
        i_min++;
}
if (east < 0) {
        if (padded[1] == 1)
                j_max -= 2;
        else
                j_max --;
}
if (south < 0) {
        if (padded[0] == 1)
                i_max -= 2;
        else
                i_max--;
}
if (west < 0) {
        j_min++;
}

MPI_Request *request = malloc(8 * sizeof(MPI_Request));
MPI_Status *status = malloc(8 * sizeof(MPI_Status));

gettimeofday(&totals,NULL);

//----Computational core----//

        #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
        /*Compute and Communicate*/


        gettimeofday(&comms,NULL);
        int len=0;

        if (north >= 0) {
                MPI_Isend(&u_previous[1][0],        1,        row,        north,        t,
```

```
MPI_COMM_WORLD, &request[len]);
                len++;
                        MPI_Irecv(&u_previous[0][0], 1, row, north, t,
MPI_COMM_WORLD, &request[len]);
                len++;
        }
        if (south >= 0) {
                MPI_Isend(&u_previous[local[0]][0], 1, row, south, t,
MPI_COMM_WORLD, &request[len]);
                len++;
                MPI_Irecv(&u_previous[local[0]+1][0], 1, row, south, t,
MPI_COMM_WORLD, &request[len]);
                len++;
        }
        if (west >= 0) {
                MPI_Isend(&u_previous[0][1], 1, column, west, t,
MPI_COMM_WORLD, &request[len]);
                len++;
                MPI_Irecv(&u_previous[0][0], 1, column, west, t,
MPI_COMM_WORLD, &request[len]);
                len++;
        }
        if (east >= 0) {
                MPI_Isend(&u_previous[0][local[1]], 1, column, east, t,
MPI_COMM_WORLD, &request[len]);
                len++;
                MPI_Irecv(&u_previous[0][local[1]+1], 1, column, east, t,
MPI_COMM_WORLD, &request[len]);
                len++;
        }

        MPI_Waitall(len, request, status);
        gettimeofday(&commf,NULL);

        tcomm+=(commf.tv_sec-comms.tv_sec)+(commf.tv_usec-
comms.tv_usec)*0.000001;

        gettimeofday(&comps,NULL);
        RedSOR(u_previous,u_current,i_min,i_max,j_min,j_max,omega);
        gettimeofday(&compf,NULL);
        tcomp+=(compf.tv_sec-comps.tv_sec)+(compf.tv_usec-
comps.tv_usec)*0.000001;
```

```
            gettimeofday(&comms,NULL);
            len=0;

            if (north >= 0) {
                  MPI_Isend(&u_current[1][0], 1, row, north, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
                MPI_Irecv(&u_current[0][0], 1, row, north, t, MPI_COMM_WORLD,
&request[len]);
                  len++;
            }
            if (south >= 0) {
                  MPI_Isend(&u_current[local[0]][0], 1, row, south, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
                  MPI_Irecv(&u_current[local[0]+1][0], 1, row, south, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
            }
            if (west >= 0) {
                  MPI_Isend(&u_current[0][1], 1, column, west, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
                  MPI_Irecv(&u_current[0][0], 1, column, west, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
            }
            if (east >= 0) {
                  MPI_Isend(&u_current[0][local[1]], 1, column, east, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
                  MPI_Irecv(&u_current[0][local[1]+1], 1, column, east, t,
MPI_COMM_WORLD, &request[len]);
                  len++;
            }

            MPI_Waitall(len, request, status);
            gettimeofday(&commf,NULL);

            tcomm+=(commf.tv_sec-comms.tv_sec)+(commf.tv_usec-
comms.tv_usec)*0.000001;
```

```
                gettimeofday(&comps,NULL);
                BlackSOR(u_previous,u_current,i_min,i_max,j_min,j_max,omega);
                gettimeofday(&compf,NULL);
                tcomp+=(compf.tv_sec-comps.tv_sec)+(compf.tv_usec-
comps.tv_usec)*0.000001;

                #ifdef TEST_CONV
        if (t%C==0) {
                        converged = converge(u_previous, u_current, i_min, i_max,
j_min, j_max);
                        MPI_Allreduce(&converged, &global_converged, 1, MPI_INT,
MPI_PROD, MPI_COMM_WORLD);
                }
                #endif

                swap=u_previous;
                u_previous=u_current;
                u_current=swap;


    }
    gettimeofday(&totalf,NULL);

    ttotal=(totalf.tv_sec-totals.tv_sec)+(totalf.tv_usec-totals.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(&tcomm,&comm_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COM
M_WORLD);

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

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

    MPI_Gatherv(&(u_previous[1][1]), 1, local_block, &(U[0][0]), sendcounts, displs,
global_block, 0, MPI_COMM_WORLD);

    //----Printing results----//
```

```
   if (rank==0) {
            printf("RedBlackSOR X %d Y %d Px %d Py %d Iter %d
CommunicationTime %lf ComputationTime %lf TotalTime %lf midpoint %lf\n",


global[0],global[1],grid[0],grid[1],t,comm_time,comp_time,total_time,U[global[0]/2]
[global[1]/2]);

            #ifdef PRINT_RESULTS
                char * s=malloc(50*sizeof(char));
                                    sprintf(s,"resRedBlackSORMPI_%dx%d_%dx
%d",global[0],global[1],grid[0],grid[1]);
                 fprint2d(s,U,global[0],global[1]);
               free(s);
            #endif

   }

   MPI_Finalize();
   return 0;
}
```