***mpi\_jacobi.c :***

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <sys/time.h>

#include "mpi.h"

#include "utils.h"

void Jacobi**(**double **\*\*** u\_previous**,** double **\*\*** u\_current**,** int X\_min**,** int X\_max**,** int Y\_min**,** int Y\_max**)** **{**

int i**,**j**;**

**for** **(**i**=**X\_min**;**i**<**X\_max**;**i**++)**

**for** **(**j**=**Y\_min**;**j**<**Y\_max**;**j**++)**

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**;**

**}**

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 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

double omega**;** //relaxation factor - useless for Jacobi

struct timeval tts**,**ttf**,**tcs**,**tcf**;** //Timers: total-> tts,ttf, computation -> tcs,tcf

double ttotal**=**0**,**tcomp**=**0**,**total\_time**,**comp\_time**;**

double **\*\*** U**,** **\*\*** u\_current**,** **\*\*** u\_previous**,** **\*\*** swap**;** //Global matrix, local current and previous matrices, pointer to swap between current and previous

// initialize mpi and give values to rank and size

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**]);**

**}**

//----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

// creates a non periodic 2d cartesian communicator (Px x Py) with name CART\_COM

MPI\_Cart\_create**(**MPI\_COMM\_WORLD**,**2**,**grid**,**periods**,**0**,&**CART\_COMM**);**

// now let's store x and y of process rank in rank\_grid

MPI\_Cart\_coords**(**CART\_COMM**,**rank**,**2**,**rank\_grid**);**

//----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**];**

**}**

**}**

//Initialization of omega

omega**=**2.0**/(**1**+**sin**(**3.14**/**global**[**0**]));**

//----Allocate global 2D-domain and initialize boundary values----//

//----Rank 0 holds the global 2D-domain----//

**if** **(**rank**==**0**)** **{**

U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

init2d**(**U**,**global**[**0**],**global**[**1**]);**

**}**

**else** U**=**allocate2d**(**1**,**1**);**

//----Allocate local 2D-subdomains u\_current, u\_previous----//

//----Add a row/column on each size for ghost cells----//

u\_previous**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

u\_current**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

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

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

/\*\*\*\*\*The usage of datatypes is optional\*\*\*\*\*/

//----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----//

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 defines positions and counts of local blocks (2D-subdomains) on global matrix----//

int **\*** scatteroffset**,** **\*** scattercounts**;**

**if** **(**rank**==**0**)** **{**

scatteroffset**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

scattercounts**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

**for** **(**i**=**0**;**i**<**grid**[**0**];**i**++)**

**for** **(**j**=**0**;**j**<**grid**[**1**];**j**++)** **{**

scattercounts**[**i**\***grid**[**1**]+**j**]=**1**;**

scatteroffset**[**i**\***grid**[**1**]+**j**]=(**local**[**0**]\***local**[**1**]\***grid**[**1**]\***i**+**local**[**1**]\***j**);**

**}**

**}**

//----Rank 0 scatters the global matrix----//

MPI\_Scatterv**(&(**U**[**0**][**0**]),**scattercounts**,**scatteroffset**,**global\_block**,&(**u\_current**[**1**][**1**]),**1**,**local\_block**,**0**,**MPI\_COMM\_WORLD**);**

**for** **(**i**=**1**;** i**<**local**[**0**]+**1**;** i**++)**

**for(**j**=**1**;** j**<**local**[**1**]+**1**;** j**++)**

u\_previous**[**i**][**j**]=**u\_current**[**i**][**j**];**

**if** **(**rank**==**0**)** free2d**(**U**);**

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

MPI\_Datatype row**;**

MPI\_Type\_contiguous**(**local**[**1**],**MPI\_DOUBLE**,&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**row**);**

MPI\_Type\_commit**(&**row**);**

MPI\_Datatype column**;**

MPI\_Type\_vector**(**local**[**0**],**1**,**local**[**1**]+**2**,**MPI\_DOUBLE**,&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**column**);**

MPI\_Type\_commit**(&**column**);**

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

int north**,** south**,** east**,** west**;**

/\*Make sure you handle non-existing

neighbors appropriately\*/

MPI\_Cart\_shift**(**CART\_COMM**,**0**,**1**,&**north**,&**south**);**

MPI\_Cart\_shift**(**CART\_COMM**,**1**,**1**,&**west**,&**east**);**

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

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**=**2**;**

**}**

**if(**south**<**0**)** **{**

i\_max**=** i\_max**-(**global\_padded**[**0**]-**global**[**0**]);**

**}**

**if(**west**<**0**)** **{**

j\_min**=**2**;**

**}**

**if(**east**<**0**)** **{**

j\_max**=** j\_max**-(**global\_padded**[**1**]-**global**[**1**]);**

**}**

/\*Three types of ranges:

-internal processes

-boundary processes

-boundary processes and padded global array

\*/

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

gettimeofday**(&**tts**,** **NULL);**

MPI\_Status status**;**

#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\*/

**if(**north**>=**0**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**1**]),**1**,**row**,**north**,**rank**,&(**u\_previous**[**0**][**1**]),**1**,**row**,**north**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,&**status**);**

**}**

**if(**south**>=**0**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**local**[**0**]][**1**]),**1**,**row**,**south**,**rank**,&(**u\_previous**[**local**[**0**]+**1**][**1**]),**1**,**row**,**south**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,&**status**);**

**}**

**if(**west**>=**0**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**1**]),**1**,**column**,**west**,**rank**,&(**u\_previous**[**1**][**0**]),**1**,**column**,**west**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,&**status**);**

**}**

**if(**east**>=**0**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**local**[**1**]]),**1**,**column**,**east**,**rank**,&(**u\_previous**[**1**][**local**[**1**]+**1**]),**1**,**column**,**east**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,&**status**);**

**}**

gettimeofday**(&**tcs**,** **NULL)** **;**

Jacobi**(**u\_previous**,** u\_current**,** i\_min**,** i\_max**,** j\_min**,** j\_max**);**

gettimeofday**(&**tcf**,** **NULL);**

tcomp**+=(**tcf**.**tv\_sec**-**tcs**.**tv\_sec**)+(**tcf**.**tv\_usec**-**tcs**.**tv\_usec**)\***0.000001**;**

swap **=** u\_current**;**

u\_current **=** u\_previous**;**

u\_previous **=** swap**;**

/\*Compute and Communicate\*/

/\*Add appropriate timers for computation\*/

**}**

#endif \*/

#ifdef TEST\_CONV

**if** **(**t**%**C**==**0**)** **{**

/\*Test convergence\*/

gettimeofday**(&**tconvs**,** **NULL);**

converged **=** converge**(**u\_previous**,** u\_current**,** local**[**0**],** local**[**1**]);**

gettimeofday**(&**tconvf**,** **NULL);**

MPI\_Allreduce**(&**converged**,** **&**global\_converged**,**1**,** MPI\_INT**,** MPI\_LAND**,** CART\_COMM**);**

tconv **+=** **(**tconvf**.**tv\_sec **-** tconvs**.**tv\_sec**)** **+** **(**tconvf**.**tv\_usec **-** tconvs**.**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**);**

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

**if** **(**rank**==**0**)** U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

MPI\_Gatherv**(&(**u\_current**[**1**][**1**]),**1**,**local\_block**,&(**U**[**0**][**0**]),**scattercounts**,**scatteroffset**,**global\_block**,**0**,**MPI\_COMM\_WORLD**);**

**if** **(**rank**==**0**)** **{**

printf**(**"Jacobi X %d Y %d Px %d Py %d Iter %d ComputationTime %lf TotalTime %lf midpoint %lf\n"**,**global**[**0**],**global**[**1**],**grid**[**0**],**grid**[**1**],**t**,**comp\_time**,**total\_time**,**U**[**global**[**0**]/**2**][**global**[**1**]/**2**]);**

#ifdef PRINT\_RESULTS

char **\*** s**=**malloc**(**50**\*sizeof(**char**));**

sprintf**(**s**,**"resJacobiMPI\_%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**;**

**}**

***mpi\_gauss.c :***

#include <stdio.h>

#include <stdlib.h>

#include <math.h>

#include <sys/time.h>

#include "mpi.h"

#include "utils.h"

void GaussSeidel**(**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**++)**

u\_current**[**i**][**j**]=**u\_previous**[**i**][**j**]+(**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**])\***omega**/**4.0**;**

**}**

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 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

double omega**;** //relaxation factor - useless for Jacobi

struct timeval tts**,**ttf**,**tcs**,**tcf**,**tconvs**,**tconvf**;** //Timers: total-> tts,ttf, computation -> tcs,tcf , converge ->tconvs,tconvf

double ttotal**=**0**,**tcomp**=**0**,**total\_time**,**comp\_time**,**tconv**=**0**,**conv\_time**;**

double **\*\*** U**,** **\*\*** u\_current**,** **\*\*** u\_previous**,** **\*\*** swap**;** //Global matrix, local current and previous matrices, pointer to swap between current and previous

// MPI INIT is used in order to shift left input arguments

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----//

// Input is in form --> executable Array\_X Array\_Y Grid\_X Grid\_Y

// NOTICE --> Number of cores is Grid\_X x Grid\_Y

**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**]);**

**}**

//----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

//----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**];**

**}**

**}**

//Initialization of omega

omega**=**2.0**/(**1**+**sin**(**3.14**/**global**[**0**]));**

//----Allocate global 2D-domain and initialize boundary values----//

//----Rank 0 holds the global 2D-domain----//

**if** **(**rank**==**0**)** **{**

U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

init2d**(**U**,**global**[**0**],**global**[**1**]);**

**}**

**else** U**=**allocate2d**(**1**,**1**);** // allocate for each process or seg fault !

//----Allocate local 2D-subdomains u\_current, u\_previous----//

//----Add a row/column on each size for ghost cells----//

u\_previous**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

u\_current**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

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

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

/\*\*\*\*\*The usage of datatypes is optional\*\*\*\*\*/

//----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----//

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 defines positions and counts of local blocks (2D-subdomains) on global matrix----//

int **\*** scatteroffset**,** **\*** scattercounts**;**

**if** **(**rank**==**0**)** **{**

scatteroffset**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

scattercounts**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

**for** **(**i**=**0**;**i**<**grid**[**0**];**i**++)**

**for** **(**j**=**0**;**j**<**grid**[**1**];**j**++)** **{**

scattercounts**[**i**\***grid**[**1**]+**j**]=**1**;**

scatteroffset**[**i**\***grid**[**1**]+**j**]=(**local**[**0**]\***local**[**1**]\***grid**[**1**]\***i**+**local**[**1**]\***j**);**

**}**

**}**

//----Rank 0 scatters the global matrix----//

/\*Fill your code here\*/

MPI\_Scatterv**(&**U**[**0**][**0**],** scattercounts**,** scatteroffset**,** global\_block**,** **&**u\_current**[**1**][**1**],** 1**,** local\_block**,** 0**,** MPI\_COMM\_WORLD**);**

/\*Make sure u\_current and u\_previous are

both initialized\*/

**for** **(**i**=**0**;** i**<**local**[**0**]+**2**;** i**++)**

**for** **(**j**=**0**;** j**<**local**[**1**]+**2**;** j**++)**

u\_previous**[**i**][**j**]=**u\_current**[**i**][**j**];**

**if** **(**rank**==**0**)** free2d**(**U**);**

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

MPI\_Datatype row**;**

MPI\_Type\_contiguous**(**local**[**1**],**MPI\_DOUBLE**,** **&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**row**);**

MPI\_Type\_commit**(&**row**);**

MPI\_Datatype column**;**

MPI\_Type\_vector**(**local**[**0**],** 1**,** local**[**1**]** **+** 2**,** MPI\_DOUBLE**,** **&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**column**);**

MPI\_Type\_commit**(&**column**);**

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

int north**,** south**,** east**,** west**;**

MPI\_Cart\_shift**(**CART\_COMM**,**0**,**1**,&**north**,&**south**);**

MPI\_Cart\_shift**(**CART\_COMM**,**1**,**1**,&**west**,&**east**);**

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

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**;**

// after setting default values let's check if process is boundary

// upper bound

**if** **(**north**==**MPI\_PROC\_NULL**)** **{**

i\_min**=**2**;**

**}**

// lower bound

**if** **(**south**==**MPI\_PROC\_NULL**)** **{**

i\_max **=** local**[**0**]-(**global\_padded**[**0**]-**global**[**0**])** **;**

**}**

// left bound

**if** **(**west**==**MPI\_PROC\_NULL**)** **{**

j\_min **=** 2**;**

**}**

// right bound

**if** **(**east**==**MPI\_PROC\_NULL**)** **{**

j\_max **=** local**[**1**]-(**global\_padded**[**1**]-**global**[**1**])** **;**

**}**

//----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

/\*Compute and Communicate\*/

swap **=** u\_previous**;**

u\_previous **=** u\_current**;**

u\_current **=** swap**;**

// The first thing we have to do is send previous up and left (if neighbours exist)

**if(**north**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Send**(&(**u\_previous**[**1**][**1**]),**1**,**row**,**north**,**north**,**MPI\_COMM\_WORLD**);**

**}**

**if(**west**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Send**(&(**u\_previous**[**1**][**1**]),**1**,**column**,**west**,**west**,**MPI\_COMM\_WORLD**);**

**}**

// Four messages will be sent to us (if we have four neighbours)

// Then we have to receive previous from right and down

**if(**south**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Recv**(&(**u\_previous**[**local**[**0**]+**1**][**1**]),**1**,**row**,**south**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**east**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Recv**(&(**u\_previous**[**1**][**local**[**1**]+**1**]),**1**,**column**,**east**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

// Then we have to receive current from up and left

**if(**north**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Recv**(&(**u\_current**[**0**][**1**]),**1**,**row**,**north**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,** MPI\_STATUS\_IGNORE**);**

**}**

**if(**west**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Recv**(&(**u\_current**[**1**][**0**]),**1**,**column**,** west**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,** MPI\_STATUS\_IGNORE**);**

**}**

// At this time we have all the necessary info in order to execute Gauss Seidel computation

gettimeofday**(&**tcs**,NULL);**

GaussSeidel**(**u\_previous**,**u\_current**,**i\_min**,**i\_max**,**j\_min**,**j\_max**,**omega**);**

gettimeofday**(&**tcf**,NULL);**

tcomp **+=(**tcf**.**tv\_sec**-**tcs**.**tv\_sec**)+(**tcf**.**tv\_usec**-**tcs**.**tv\_usec**)\***0.000001**;**

// Okay, now let's send current to right and down

**if(**south**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Send**((&**u\_current**[**local**[**0**]][**1**]),**1**,**row**,**south**,**south**,**MPI\_COMM\_WORLD**);**

**}**

**if(**east**!=**MPI\_PROC\_NULL**){**

MPI\_Send**((&**u\_current**[**1**][**local**[**1**]]),**1**,**column**,**east**,**east**,**MPI\_COMM\_WORLD**);**

**}**

#ifdef TEST\_CONV

**if** **(**t**%**C**==**0**)** **{**

/\*Test convergence\*/

gettimeofday**(&**tconvs**,** **NULL);**

converged **=** converge**(**u\_previous**,** u\_current**,** local**[**0**]+**2**,** local**[**1**]+**2**);**

gettimeofday**(&**tconvf**,** **NULL);**

MPI\_Allreduce**(&**converged**,** **&**global\_converged**,**1**,** MPI\_INT**,** MPI\_LAND**,** CART\_COMM**);**

tconv **+=** **(**tconvf**.**tv\_sec **-** tconvs**.**tv\_sec**)** **+** **(**tconvf**.**tv\_usec **-** tconvs**.**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**);**

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

**if** **(**rank**==**0**)** **{**

U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

**}**

MPI\_Gatherv**(&**u\_current**[**1**][**1**],** 1**,** local\_block**,** **&**U**[**0**][**0**],** scattercounts**,** scatteroffset**,** global\_block**,** 0**,** MPI\_COMM\_WORLD**);**

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

**if** **(**rank**==**0**)** **{**

printf**(**"GaussSeidelSOR X %d Y %d Px %d Py %d Iter %d ComputationTime %lf TotalTime %lf ConvergeTime %lf midpoint %lf\n"**,**global**[**0**],**global**[**1**],**grid**[**0**],**grid**[**1**],**t**,**comp\_time**,**total\_time**,**conv\_time**,**U**[**global**[**0**]/**2**][**global**[**1**]/**2**]);**

#ifdef PRINT\_RESULTS

char **\*** s**=**malloc**(**50**\*sizeof(**char**));**

sprintf**(**s**,**"resGaussMPI\_%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**;**

**}**

***mpi\_redblack.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 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

double omega**;** //relaxation factor - useless for Jacobi

struct timeval tts**,**ttf**,**tcs**,**tcf**,**tconvs**,**tconvf**;** //Timers: total-> tts,ttf, computation -> tcs,tcf , converge ->tconvs,tconvf

double ttotal**=**0**,**tcomp**=**0**,**total\_time**,**comp\_time**,**tconv**=**0**,**conv\_time**;**

double **\*\*** U**,** **\*\*** u\_current**,** **\*\*** u\_previous**,** **\*\*** swap**;** //Global matrix, local current and previous matrices, pointer to swap between current and previous

// MPI INIT is used in order to shift left input arguments

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----//

// Input is in form --> executable Array\_X Array\_Y Grid\_X Grid\_Y

// NOTICE --> Number of cores is Grid\_X x Grid\_Y

**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**]);**

**}**

//----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

//----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**];**

**}**

**}**

//Initialization of omega

omega**=**2.0**/(**1**+**sin**(**3.14**/**global**[**0**]));**

//----Allocate global 2D-domain and initialize boundary values----//

//----Rank 0 holds the global 2D-domain----//

**if** **(**rank**==**0**)** **{**

U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

init2d**(**U**,**global**[**0**],**global**[**1**]);**

**}**

**else** U**=**allocate2d**(**1**,**1**);** // allocate for each process or seg fault !

//----Allocate local 2D-subdomains u\_current, u\_previous----//

//----Add a row/column on each size for ghost cells----//

u\_previous**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

u\_current**=**allocate2d**(**local**[**0**]+**2**,**local**[**1**]+**2**);**

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

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

/\*\*\*\*\*The usage of datatypes is optional\*\*\*\*\*/

//----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----//

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 defines positions and counts of local blocks (2D-subdomains) on global matrix----//

int **\*** scatteroffset**,** **\*** scattercounts**;**

**if** **(**rank**==**0**)** **{**

scatteroffset**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

scattercounts**=(**int**\*)**malloc**(**size**\*sizeof(**int**));**

**for** **(**i**=**0**;**i**<**grid**[**0**];**i**++)**

**for** **(**j**=**0**;**j**<**grid**[**1**];**j**++)** **{**

scattercounts**[**i**\***grid**[**1**]+**j**]=**1**;**

scatteroffset**[**i**\***grid**[**1**]+**j**]=(**local**[**0**]\***local**[**1**]\***grid**[**1**]\***i**+**local**[**1**]\***j**);**

**}**

**}**

//----Rank 0 scatters the global matrix----//

/\*Fill your code here\*/

MPI\_Scatterv**(&**U**[**0**][**0**],** scattercounts**,** scatteroffset**,** global\_block**,** **&**u\_current**[**1**][**1**],** 1**,** local\_block**,** 0**,** MPI\_COMM\_WORLD**);**

/\*Make sure u\_current and u\_previous are

both initialized\*/

**for** **(**i**=**0**;** i**<**local**[**0**]+**2**;** i**++)**

**for** **(**j**=**0**;** j**<**local**[**1**]+**2**;** j**++)**

u\_previous**[**i**][**j**]=**u\_current**[**i**][**j**];**

**if** **(**rank**==**0**)** free2d**(**U**);**

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

MPI\_Datatype row**;**

MPI\_Type\_vector**(**1**,** local**[**1**],** local**[**1**]+**2**,** MPI\_DOUBLE**,** **&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**row**);**

MPI\_Type\_commit**(&**row**);**

MPI\_Datatype column**;**

MPI\_Type\_vector**(**local**[**0**],** 1**,** local**[**1**]** **+** 2**,** MPI\_DOUBLE**,** **&**dummy**);**

MPI\_Type\_create\_resized**(**dummy**,**0**,sizeof(**double**),&**column**);**

MPI\_Type\_commit**(&**column**);**

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

int north**,** south**,** east**,** west**;**

MPI\_Cart\_shift**(**CART\_COMM**,**0**,**1**,&**north**,&**south**);**

MPI\_Cart\_shift**(**CART\_COMM**,**1**,**1**,&**west**,&**east**);**

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

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**;**

// after setting default values let's check if process is boundary

// upper bound

**if** **(**north**==**MPI\_PROC\_NULL**)** **{**

i\_min**=**2**;**

**}**

// lower bound

**if** **(**south**==**MPI\_PROC\_NULL**)** **{**

i\_max **=** local**[**0**]-(**global\_padded**[**0**]-**global**[**0**])** **;**

**}**

// left bound

**if** **(**west**==**MPI\_PROC\_NULL**)** **{**

j\_min **=** 2**;**

**}**

// right bound

**if** **(**east**==**MPI\_PROC\_NULL**)** **{**

j\_max **=** local**[**1**]-(**global\_padded**[**1**]-**global**[**1**])** **;**

**}**

//----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

/\*Compute and Communicate\*/

swap **=** u\_previous**;**

u\_previous **=** u\_current**;**

u\_current **=** swap**;**

// Before Red is executed we have to send black cells

// As a result we send receive black cells we all the neighbors we have (if there are any neighbors)

**if(**north**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**1**]),**1**,**row**,**north**,**rank**,&(**u\_previous**[**0**][**1**]),**1**,**row**,**north**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**west**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**1**]),**1**,**column**,**west**,**rank**,&(**u\_previous**[**1**][**0**]),**1**,**column**,**west**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**south**!=**MPI\_PROC\_NULL**){**

MPI\_Sendrecv**(&(**u\_previous**[**local**[**0**]][**1**]),**1**,**row**,**south**,**rank**,&(**u\_previous**[**local**[**0**]+**1**][**1**]),**1**,**row**,**south**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**east**!=**MPI\_PROC\_NULL**){**

MPI\_Sendrecv**(&(**u\_previous**[**1**][**local**[**1**]]),**1**,**column**,**east**,**rank**,&(**u\_previous**[**1**][**local**[**1**]+**1**]),**1**,**column**,**east**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

// After message passing has been completed we compute red cells

gettimeofday**(&**tcs**,NULL);**

RedSOR**(**u\_previous**,**u\_current**,**i\_min**,**i\_max**,**j\_min**,**j\_max**,**omega**);**

gettimeofday**(&**tcf**,NULL);**

tcomp **+=(**tcf**.**tv\_sec**-**tcs**.**tv\_sec**)+(**tcf**.**tv\_usec**-**tcs**.**tv\_usec**)\***0.000001**;**

// Now we have to send receive reds to all our neighbours

**if(**north**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Sendrecv**(&(**u\_current**[**1**][**1**]),**1**,**row**,**north**,**rank**,&(**u\_current**[**0**][**1**]),**1**,**row**,**north**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**west**!=**MPI\_PROC\_NULL**)** **{**

MPI\_Sendrecv**(&(**u\_current**[**1**][**1**]),**1**,**column**,**west**,**rank**,&(**u\_current**[**1**][**0**]),**1**,**column**,**west**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**south**!=**MPI\_PROC\_NULL**){**

MPI\_Sendrecv**(&(**u\_current**[**local**[**0**]][**1**]),**1**,**row**,**south**,**rank**,&(**u\_current**[**local**[**0**]+**1**][**1**]),**1**,**row**,**south**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

**if(**east**!=**MPI\_PROC\_NULL**){**

MPI\_Sendrecv**(&(**u\_current**[**1**][**local**[**1**]]),**1**,**column**,**east**,**rank**,&(**u\_current**[**1**][**local**[**1**]+**1**]),**1**,**column**,**east**,**MPI\_ANY\_TAG**,**MPI\_COMM\_WORLD**,**MPI\_STATUS\_IGNORE**);**

**}**

// After message passing has been completted we compute black cells

gettimeofday**(&**tcs**,NULL);**

BlackSOR**(**u\_previous**,**u\_current**,**i\_min**,**i\_max**,**j\_min**,**j\_max**,**omega**);**

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**(&**tconvs**,** **NULL);**

converged **=** converge**(**u\_previous**,**u\_current**,**local**[**0**]+**2**,**local**[**1**]+**2**);**

gettimeofday**(&**tconvf**,NULL);**

MPI\_Allreduce**(&**converged**,&**global\_converged**,**1**,**MPI\_INT**,**MPI\_LAND**,**CART\_COMM**);**

tconv**+=(**tconvf**.**tv\_sec**-**tconvs**.**tv\_sec**)+(**tconvf**.**tv\_usec**-**tconvs**.**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**);**

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

**if** **(**rank**==**0**)** **{**

U**=**allocate2d**(**global\_padded**[**0**],**global\_padded**[**1**]);**

**}**

/\*Fill your code here\*/

MPI\_Gatherv**(&**u\_current**[**1**][**1**],** 1**,** local\_block**,** **&**U**[**0**][**0**],** scattercounts**,** scatteroffset**,** global\_block**,** 0**,** MPI\_COMM\_WORLD**);**

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

**if** **(**rank**==**0**)** **{**

printf**(**"RebBlackSOR X %d Y %d Px %d Py %d Iter %d ComputationTime %lf TotalTime %lf ConvergeTime %lf midpoint %lf\n"**,**global**[**0**],**global**[**1**],**grid**[**0**],**grid**[**1**],**t**,**comp\_time**,**total\_time**,**conv\_time**,**U**[**global**[**0**]/**2**][**global**[**1**]/**2**]);**

#ifdef PRINT\_RESULTS

char **\*** s**=**malloc**(**50**\*sizeof(**char**));**

sprintf**(**s**,**"resRedBlackMPI\_%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**;**

**}**