a6/heat_transfer/mpi/mpi_jacobi.c

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

int main(int argc, char ** argv) {
    int rank,size;
    int global[2],local[2]; //global matrix dimensions and local matrix dimensions
    int global_padded[2];   //padded global matrix dimensions
    int grid[2];            //processor grid dimensions
    int i,j,t;
    int global_converged=0,converged=0; //flags for convergence
    MPI_Datatype dummy;     //dummy datatype
    double omega;           //relaxation factor - useless for Jacobi

    struct timeval tts,ttf,tcs,tcf;   //Timers
    double ttotal=0,tcomp=0,total_time,comp_time;
    double t_conv=0.0;
    double t1,t2;

    double ** U, ** u_current, ** u_previous, ** swap;

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

    //----Read arguments----//
    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----//
    MPI_Comm CART_COMM;
    int periods[2]={0,0};
    int rank_grid[2];

    MPI_Cart_create(MPI_COMM_WORLD,2,grid,periods,0,&CART_COMM);
    MPI_Cart_coords(CART_COMM,rank,2,rank_grid);

    //----Compute local dimensions & Padding----//
    for (i=0;i<2;i++) {
        if (global[i]%grid[i]==0) {
```

```
52              local[i]=global[i]/grid[i];
53              global_padded[i]=global[i];
54          }
55          else {
56              local[i]=(global[i]/grid[i])+1;
57              global_padded[i]=local[i]*grid[i];
58          }
59      }
60
61      //Initialization of omega
62      omega=2.0/(1+sin(3.14/global[0]));
63
64      //----Allocate global 2D-domain----//
65      if (rank==0) {
66          U=allocate2d(global_padded[0],global_padded[1]);
67          init2d(U,global[0],global[1]);
68      }
69
70      //----Allocate local 2D-subdomains----//
71      u_previous=allocate2d(local[0]+2,local[1]+2);
72      u_current=allocate2d(local[0]+2,local[1]+2);
73
74      //----Datatypes Definition----//
75      MPI_Datatype global_block;
76      MPI_Type_vector(local[0],local[1],global_padded[1],MPI_DOUBLE,&dummy);
77      MPI_Type_create_resized(dummy,0,sizeof(double),&global_block);
78      MPI_Type_commit(&global_block);
79
80      MPI_Datatype local_block;
81      MPI_Type_vector(local[0],local[1],local[1]+2,MPI_DOUBLE,&dummy);
82      MPI_Type_create_resized(dummy,0,sizeof(double),&local_block);
83      MPI_Type_commit(&local_block);
84
85      //----Scatter parameters----//
86      int * scatteroffset, * scattercounts;
87      if (rank==0) {
88          scatteroffset=(int*)malloc(size*sizeof(int));
89          scattercounts=(int*)malloc(size*sizeof(int));
90          for (i=0;i<grid[0];i++)
91              for (j=0;j<grid[1];j++) {
92                  scattercounts[i*grid[1]+j]=1;
93                  scatteroffset[i*grid[1]+j]=(local[0]*local[1]*grid[1]*i+local[1]*j);
94              }
95      }
96
97      //----Scatter----//
98      MPI_Scatterv(rank == 0 ? &U[0][0] : NULL,
99                   scattercounts, scatteroffset, global_block,
100                  &u_previous[1][1], 1, local_block,
101                  0, MPI_COMM_WORLD);
102
103     // Init u_current
104     for (i = 1; i <= local[0]; i++)
105         for (j = 1; j <= local[1]; j++)
```

```
106             u_current[i][j] = u_previous[i][j];
107
108     if (rank==0)
109         free2d(U);
110
111     //----Communication Datatypes----//
112     MPI_Datatype row_type, col_type;
113     MPI_Type_contiguous(local[1], MPI_DOUBLE, &row_type);
114     MPI_Type_commit(&row_type);
115     MPI_Type_vector(local[0], 1, local[1] + 2, MPI_DOUBLE, &col_type);
116     MPI_Type_commit(&col_type);
117
118     //----Find Neighbors----//
119     int north, south, east, west;
120     MPI_Cart_shift(CART_COMM, 0, 1, &north, &south);
121     MPI_Cart_shift(CART_COMM, 1, 1, &west, &east);
122
123     //---Define iteration ranges-----//
124     int i_min,i_max,j_min,j_max;
125
126     i_min = 1;
127     i_max = local[0];
128     j_min = 1;
129     j_max = local[1];
130
131     if (rank_grid[0] == 0) i_min = 2;
132     if (rank_grid[0] == grid[0] - 1) {
133         i_max = local[0] - (global_padded[0] - global[0]) - 1;
134         if (global_padded[0] == global[0]) i_max = local[0] - 1;
135     }
136
137     if (rank_grid[1] == 0) j_min = 2;
138     if (rank_grid[1] == grid[1] - 1) {
139          j_max = local[1] - (global_padded[1] - global[1]) - 1;
140          if (global_padded[1] == global[1]) j_max = local[1] - 1;
141     }
142
143     //----Computational core----//
144     gettimeofday(&tts, NULL);
145
146     #ifdef TEST_CONV
147     for (t=0;t<T && !global_converged;t++) {
148     #endif
149     #ifndef TEST_CONV
150     #undef T
151     #define T 256
152     for (t=0;t<T;t++) {
153     #endif
154
155         // 1. Swap
156         swap = u_previous;
157         u_previous = u_current;
158         u_current = swap;
159
```

```
160         // 2. Communication
161         MPI_Request reqs[8];
162         MPI_Status stats[8];
163         int req_cnt = 0;
164 
165         MPI_Isend(&u_previous[1][1], 1, row_type, north, 1, CART_COMM, &reqs[req_cnt++]);
166         MPI_Irecv(&u_previous[0][1], 1, row_type, north, 2, CART_COMM, &reqs[req_cnt++]);
167 
168         MPI_Isend(&u_previous[local[0]][1], 1, row_type, south, 2, CART_COMM,
    &reqs[req_cnt++]);
169         MPI_Irecv(&u_previous[local[0]+1][1], 1, row_type, south, 1, CART_COMM,
    &reqs[req_cnt++]);
170 
171         MPI_Isend(&u_previous[1][1], 1, col_type, west, 3, CART_COMM, &reqs[req_cnt++]);
172         MPI_Irecv(&u_previous[1][0], 1, col_type, west, 4, CART_COMM, &reqs[req_cnt++]);
173 
174         MPI_Isend(&u_previous[1][local[1]], 1, col_type, east, 4, CART_COMM,
    &reqs[req_cnt++]);
175         MPI_Irecv(&u_previous[1][local[1]+1], 1, col_type, east, 3, CART_COMM,
    &reqs[req_cnt++]);
176 
177         MPI_Waitall(req_cnt, reqs, stats);
178 
179         // 3. Computation
180         gettimeofday(&tcs, NULL);
181         for (i = i_min; i <= i_max; i++) {
182             for (j = j_min; j <= j_max; j++) {
183                 u_current[i][j] = (u_previous[i-1][j] + u_previous[i+1][j] +
184                                    u_previous[i][j-1] + u_previous[i][j+1]) / 4.0;
185             }
186         }
187         gettimeofday(&tcf, NULL);
188         tcomp += (tcf.tv_sec - tcs.tv_sec) + (tcf.tv_usec - tcs.tv_usec) * 0.000001;
189 
190         // 4. Convergence Check
191         #ifdef TEST_CONV
192         if (t % C == 0) {
193             converged = converge(u_previous, u_current, i_min, i_max, j_min, j_max);
194 
195         t1=MPI_Wtime(); //
196             MPI_Allreduce(&converged, &global_converged, 1, MPI_INT, MPI_LAND, CART_COMM);
197 
198         t2=MPI_Wtime();
199             t_conv+=(t2-t1);
200         }
201         #endif
202 
203     }
204     gettimeofday(&ttf,NULL);
205 
206     ttotal=(ttf.tv_sec-tts.tv_sec)+(ttf.tv_usec-tts.tv_usec)*0.000001;
207 
208     MPI_Reduce(&ttotal,&total_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
209     MPI_Reduce(&tcomp,&comp_time,1,MPI_DOUBLE,MPI_MAX,0,MPI_COMM_WORLD);
```

```
210
211     //----Gather results----//
212     if (rank==0) {
213             U=allocate2d(global_padded[0],global_padded[1]);
214     }
215
216     MPI_Gatherv(&u_current[1][1], 1, local_block,
217                 rank == 0 ? &U[0][0] : NULL,
218                 scattercounts, scatteroffset, global_block,
219                 0, MPI_COMM_WORLD);
220
221     //----Printing results----//
222     if (rank==0) {
223         printf("Jacobi X %d Y %d Px %d Py %d Iter %d ComputationTime %lf TotalTime %lf
    ConvergenceTime %lf midpoint %lf\n",
224
    global[0],global[1],grid[0],grid[1],t,comp_time,total_time,t_conv,U[global[0]/2]
    [global[1]/2]);
225
226         #ifdef PRINT_RESULTS
227         char * s=malloc(50*sizeof(char));
228         sprintf(s,"resJacobiMPI_%dx%d_%dx%d",global[0],global[1],grid[0],grid[1]);
229         fprint2d(s,U,global[0],global[1]);
230         free(s);
231         #endif
232     }
233
234     // Free Datatypes before Finalize
235     MPI_Type_free(&row_type);
236     MPI_Type_free(&col_type);
237     MPI_Type_free(&global_block);
238     MPI_Type_free(&local_block);
239     if(rank==0){
240         free(scattercounts);
241         free(scatteroffset);
242     }
243
244     MPI_Finalize();
245     return 0;
246 }
247
```