```cpp
//------------------------------------------------------------------------
// 2E_Jacobi_parallel_submatrices.cpp
//------------------------------------------------------------------------
#if (defined __linux__) || (defined _AIX) || (defined __APPLE__)
    #include <sys/types.h>
    #include <sys/stat.h>
    #include <unistd.h>
#elif (defined _WIN32) || (defined _WIN64)
    #include <conio.h>
    #include <direct.h>
#endif

#include <mpi.h>
#include <math.h>
#include <time.h>
#include <iostream>
using namespace std;

#define N 20
//------------------------------------------------------------------------
int main(int argc, char* argv[])
{
    int iterations, i, j, k, rank, size, N_sqrt;
    int row_region, column_region, row_start, row_end, column_start, column_end;
    double my_time, A[N + 2][N + 2], B[N + 2][N + 2];
    MPI_Datatype mytype1, mytype2, mytype3, mytype4;

    int src, dest;
    int dims[2], periods[2];
    MPI_Comm mycomm_cart;

    MPI_Init(&argc, &argv);
    MPI_Barrier(MPI_COMM_WORLD);
    my_time = -MPI_Wtime();

    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if (rank == 0) {
        printf("There are %d processes.\n", size);
        fflush(stdout);
    }

    if (!(size == 4 || size == 16 || size == 25 || size == 100)) {
        if (rank == 0)
            printf("The number of processes should be 4, 16, 25, or 100!\n");
        goto end;
    }

    switch (size) {
    case 4: iterations = 10000000; break;
    case 16: iterations = 1000; break;
    case 25: iterations = 1000; break;
    case 100: iterations = 100; break;
    default: break;
    }

    N_sqrt = (int)sqrt(size);
    row_region = N / N_sqrt;
    column_region = N / N_sqrt;
    row_start = (rank % N_sqrt) * row_region + 1;
    row_end = row_start + row_region - 1;
    column_start = (rank / N_sqrt) * column_region + 1;
    column_end = column_start + column_region - 1;

    MPI_Type_vector(row_region, 1, 1, MPI_DOUBLE, &mytype1);
    MPI_Type_commit(&mytype1);
    MPI_Type_vector(column_region, 1, N + 2, MPI_DOUBLE, &mytype2);
    MPI_Type_commit(&mytype2);
    MPI_Type_vector(column_region, row_region, N + 2, MPI_DOUBLE, &mytype3);
    MPI_Type_commit(&mytype3);
    MPI_Type_vector((N + 2) * (N + 2), 1, 1, MPI_DOUBLE, &mytype4);
    MPI_Type_commit(&mytype4);

```

```
74        dims[0] = size / N_sqrt;
75        dims[1] = N_sqrt;
76        periods[0] = 1;
77        periods[1] = 1;
78
79        MPI_Cart_create(MPI_COMM_WORLD, 2, dims, periods, true, &mycomm_cart);
80
81        printf("Process %d: row_start %d, row_end %d column_start %d, column_end %d.\n",
82            rank, row_start, row_end, column_start, column_end);
83        fflush(stdout);
84
85        for (i = 0; i < N + 2; i++)
86            for (j = 0; j < N + 2; j++)
87                A[i][j] = B[i][j] = 0;
88
89        if (rank == 0) {
90            srand((unsigned)time(NULL));
91            for (i = 0; i < N + 2; i++)
92                for (j = 0; j < N + 2; j++)
93                    A[i][j] = rand() % 100;
94        }
95
96        if (rank == 0)
97            for (k = 1; k < size; k++)
98                MPI_Ssend(&(A[0][0]), 1, mytype4, k, 0, MPI_COMM_WORLD);
99        if (rank > 0)
100           MPI_Recv(&(A[0][0]), 1, mytype4, 0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
101
102       if (rank == 0) {
103           cout << "\nStarting values of a matrix A:\n";
104           for (i = 0; i < N + 2; i++) {
105               for (j = 0; j < N + 2; j++)
106                   printf("%6.2f ", A[i][j]);
107               cout << endl;
108           }
109       }
110       fflush(stdout);
111
112       for (k = 0; k < iterations; k++) {
113           for (i = row_start; i <= row_end; i++)
114               for (j = column_start; j <= column_end; j++)
115                   B[i][j] = 0.25 * (A[i - 1][j] + A[i + 1][j] + A[i][j - 1] + A[i][j +
                      1]);
116
117           for (i = row_start; i <= row_end; i++)
118               for (j = column_start; j <= column_end; j++)
119                   A[i][j] = B[i][j];
120
121           // posielanie riadkovych halo po vertikalach
122           // zhora dole
123           MPI_Cart_shift(mycomm_cart, 1, 1, &src, &dest);
124           if (rank < dest)
125               MPI_Ssend(&(B[row_end][column_start]), 1, mytype1, dest, rank,
                      mycomm_cart);
126           if (src < rank)
127               MPI_Recv(&(A[row_start - 1][column_start]), 1, mytype1, src, src,
128                   mycomm_cart, MPI_STATUS_IGNORE);
129
130           // zdola hore
131           MPI_Cart_shift(mycomm_cart, 1, -1, &src, &dest);
132           if (rank > dest)
133               MPI_Ssend(&(B[row_start][column_start]), 1, mytype1, dest, size + rank,
134                   mycomm_cart);
135           if (src > rank)
136               MPI_Recv(&(A[row_end + 1][column_start]), 1, mytype1, src, size + src,
137                   mycomm_cart, MPI_STATUS_IGNORE);
138
139           // posielanie stlpcovych halo po horizontalach
140           // zlava doprava
141           MPI_Cart_shift(mycomm_cart, 0, 1, &src, &dest);
142           if (rank < dest)
143               MPI_Ssend(&(B[row_start][column_end]), 1, mytype2, dest, 2 * size + rank,
144                   mycomm_cart);
```

```
145             if (src < rank)
146                 MPI_Recv(&(A[row_start][column_start - 1]), 1, mytype2, src, 2 * size +
                    src,
147                     mycomm_cart, MPI_STATUS_IGNORE);
148
149             // zprava dolava
150             MPI_Cart_shift(mycomm_cart, 0, -1, &src, &dest);
151             if (rank > dest && dest >= 0)
152                 MPI_Ssend(&(B[row_start][column_start]), 1, mytype2, dest, 3 * size +
                    rank,
153                     mycomm_cart);
154             if (src > rank)
155                 MPI_Recv(&(A[row_start][column_end + 1]), 1, mytype2, src, 3 * size +
                    src,
156                     mycomm_cart, MPI_STATUS_IGNORE);
157
158             MPI_Barrier(MPI_COMM_WORLD);
159         }
160
161         if (rank > 0) {
162             MPI_Ssend(&(A[row_start][column_start]), 1, mytype3, 0, 0, MPI_COMM_WORLD);
163         }
164         if (rank == 0)
165             for (k = 1; k < size; k++)
166                 MPI_Recv(&(A[(k % N_sqrt) * row_region + 1][(k / N_sqrt) * column_region
167                     + 1]), 1, mytype3, k, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
168
169         if (rank == 0) {
170             cout << "\nFinal values of the matrix A:\n";
171             for (i = 0; i < N + 2; i++) {
172                 for (j = 0; j < N + 2; j++)
173                     printf("%6.2f ", A[i][j]);
174                 cout << endl;
175             }
176         }
177
178     end:
179         my_time += MPI_Wtime();
180         if (rank == 0)
181             printf("\nTime = %10.6f s\n", my_time);
182
183         fflush(stdout);
184         MPI_Finalize();
185
186         return 0;
187     }
```