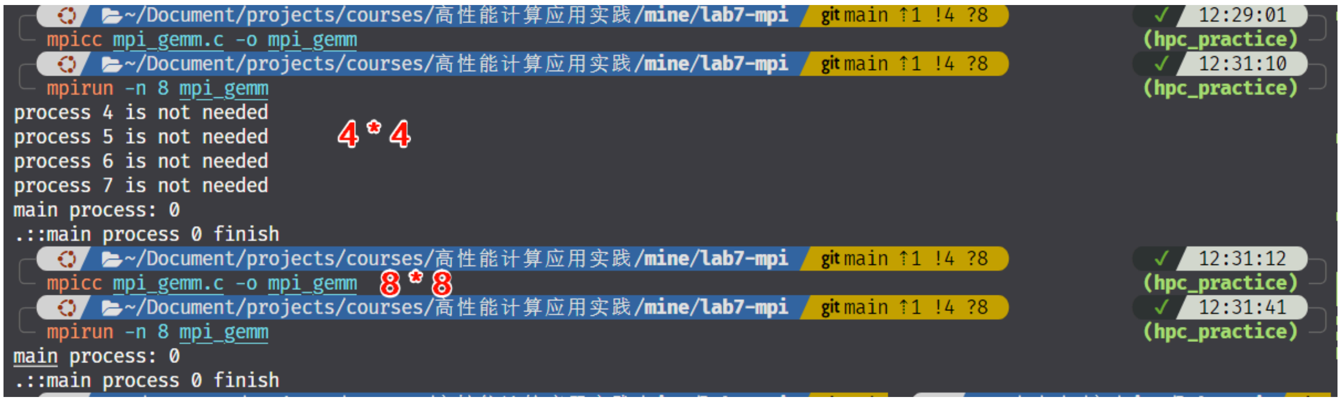
lab7 - mpi

截图



```
~/Document/projects/courses/高性能计算应用实践/mine/lab7-mpi  git main ↑1 !4 ?8          ✓  12:29:01
mpicc mpi_gemm.c -o mpi_gemm                                                           (hpc_practice)
~/Document/projects/courses/高性能计算应用实践/mine/lab7-mpi  git main ↑1 !4 ?8          ✓  12:31:10
mpirun -n 8 mpi_gemm                                                                   (hpc_practice)
process 4 is not needed
process 5 is not needed
process 6 is not needed
process 7 is not needed
main process: 0
.::main process 0 finish
~/Document/projects/courses/高性能计算应用实践/mine/lab7-mpi  git main ↑1 !4 ?8          ✓  12:31:12
mpicc mpi_gemm.c -o mpi_gemm                                                           (hpc_practice)
~/Document/projects/courses/高性能计算应用实践/mine/lab7-mpi  git main ↑1 !4 ?8          ✓  12:31:41
mpirun -n 8 mpi_gemm                                                                   (hpc_practice)
main process: 0
.::main process 0 finish
```

运行结果如下：

```
main process: 4 * 4

A={
-1.000000e+00, -2.707955e-01, 5.350056e-02, 8.634630e-01,
-9.980292e-01, -8.173388e-01, -9.113315e-02, 1.361192e-01,
-9.167380e-01, -8.154047e-01, -5.336431e-01, 1.121887e-01,
-6.467147e-01, -2.556555e-02, 6.625836e-01, -8.983362e-01,
}

B={
5.341023e-01, 7.519617e-01, 6.208589e-01, -8.464509e-01,
-9.621704e-01, 6.311373e-02, -6.231595e-01, 6.305478e-01,
-4.952805e-01, 8.405219e-01, 7.726289e-01, 9.697820e-01,
-4.036057e-01, 3.086230e-02, 1.412280e-01, -7.632966e-01,
}

C={
-6.485473e-01, -6.974358e-01, -2.888289e-01, 6.850690e-02,
2.435675e-01, -8.744635e-01, -1.614911e-01, 1.371329e-01,
5.139494e-01, -1.185891e+00, -4.575017e-01, -3.413287e-01,
-2.864045e-01, 4.127308e-02, -5.261487e-04, 1.859551e+00,
}

main process: 8 * 8

A={
1.900078e-02, 8.548636e-02, -3.314322e-01, 2.440604e-02, -6.461445e-01, -7.460251e-01,
5.706861e-01, -6.391164e-02,
-3.681220e-01, 3.359572e-01, -8.358069e-01, 1.730207e-01, -7.762620e-01, -9.918196e-01,
-1.155304e-01, -6.060053e-01,
4.034051e-01, -6.577163e-01, -4.593707e-01, 4.775171e-01, 7.053617e-01, -9.989944e-01,
-4.241922e-01, -1.056461e-01,
-2.655067e-01, 6.615583e-01, -8.514753e-01, 7.433776e-01, 6.223437e-01, -6.140962e-01,
7.067059e-01, 4.568796e-01,
7.526247e-01, 7.238108e-01, 3.524515e-01, 1.559346e-01, 9.097746e-01, 1.735603e-01,
5.194394e-02, 9.321860e-01,
-2.580107e-01, 5.384423e-01, 6.203171e-01, 9.294072e-01, 8.085664e-01, -2.500817e-01,
4.568334e-01, 5.136816e-01,
5.050480e-02, -8.948411e-01, -7.157754e-02, -3.329414e-01, -7.221695e-01, -9.464281e-
01, -4.673714e-01, -6.788623e-02,
-1.443309e-01, 9.334593e-01, 1.383701e-01, -2.143441e-01, 3.723747e-01, -7.046941e-02,
5.669680e-01, 7.379329e-01,
}

B={
-7.319215e-01, 8.165651e-01, -4.029224e-02, 8.852518e-01, 5.361704e-01, -2.435351e-01,
-5.578819e-01, -3.161686e-01,
6.921052e-01, 5.488294e-01, -9.377434e-01, -2.239507e-01, 5.038825e-01, -1.625671e-01,
5.086909e-02, -8.637625e-02,
-9.638247e-01, -2.678706e-01, 2.168604e-01, 8.610245e-01, -8.182718e-01, -7.231037e-01,
-1.094956e-02, 1.762435e-01,
8.817096e-01, 8.885760e-01, 5.845223e-01, 8.227684e-01, 7.079334e-01, -2.383066e-01,
7.650440e-01, 1.702999e-01,
3.401642e-01, 7.696433e-01, -6.350064e-01, -3.245633e-01, -6.507111e-01, -4.004399e-01,
5.476602e-01, -8.601459e-01,
5.999431e-01, -1.096578e-01, 5.024259e-02, 6.562465e-02, -9.604593e-01, -9.579758e-01,
-8.819950e-01, 6.378629e-01,
-7.135664e-01, -6.739431e-01, 7.975177e-01, 4.041545e-01, -1.618930e-01, -5.223948e-01,
-7.603972e-01, -1.936786e-01,
2.209206e-01, -1.503654e-01, -3.824202e-01, 9.420213e-01, -4.072602e-01, -9.257813e-01,
9.560251e-02, 1.709661e-01,
}

C={
-7.024896e-01, -6.175928e-01, 7.138585e-01, 6.358140e-02, 1.412361e+00, 9.497799e-01,
-1.198863e-01, -1.091875e-01,
5.495472e-01, -5.828320e-02, 2.023827e-01, -1.409116e+00, 2.501541e+00, 2.480544e+00,
8.435441e-01, -7.664718e-02,
3.326340e-02, 1.469986e+00, -1.598574e-02, -6.368557e-02, 1.211029e+00, 1.221015e+00,
1.691703e+00, -1.250213e+00,
1.568243e+00, 1.036256e+00, -3.969601e-01, -3.099131e-02, 1.298360e+00, -5.740629e-02,
1.148578e+00, -9.824464e-01,
3.303513e-01, 1.561958e+00, -1.425547e+00, 1.551175e+00, -5.565088e-01, -2.013690e+00,
1.271709e-01, -7.343229e-01,
6.956050e-01, 1.009126e+00, -1.748625e-01, 1.339494e+00, -2.857657e-01, -1.493150e+00,
1.240700e+00, -5.529886e-01,
-1.375815e+00, -8.533851e-01, 6.912177e-01, -1.710137e-01, 8.812949e-01, 1.767113e+00,
4.605126e-01, 8.840338e-02,
2.721833e-01, -3.181251e-02, -1.034846e+00, 4.047739e-01, -4.389421e-01, -1.226528e+00,
-1.319781e-01, -3.960062e-01,
}
```

代码

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include "mpi.h"

#define min(a, b) ((a) < (b) ? (a) : (b))

#define A(i, j) a[(i) * lda + (j)]
#define B(i, j) b[(i) * ldb + (j)]
#define C(i, j) c[(i) * ldc + (j)]

#define MATRIX_SIZE 8
#define SECTION_NUM min(8, MATRIX_SIZE)

void print_matrix(int m, int n, double *a, int lda, FILE *f) {
    fprintf(f, "{\n");
    for (int i=0; i<m; i++) {
        for (int j=0; j<n; j++) {
            fprintf(f, "%e, ", A(i, j));
        }
        fprintf(f, "\n");
    }
    fprintf(f, "}\n");
}


void copy_matrix(int m, int n, double *a, int lda, double *b, int ldb)
{
  int i, j;
  for (j = 0; j < n; j++)
    for (i = 0; i < m; i++)
      B(i, j) = A(i, j);
}

int main(int argc, char *argv[]) {
    int process_num, rank;
    int m = MATRIX_SIZE, n = MATRIX_SIZE, k = MATRIX_SIZE;
    int lda = MATRIX_SIZE, ldb = MATRIX_SIZE, ldc = MATRIX_SIZE;

    MPI_Init(0, 0);
    MPI_Comm_size(MPI_COMM_WORLD, &process_num);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);


    if (rank == 0) {
        // here is main process
        double *a = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        double *b = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        double *c = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);

        // double ta[MATRIX_SIZE * MATRIX_SIZE] = {
        //     -1.000000e+00, -2.707955e-01, 5.350056e-02, 8.634630e-01,
        //     -9.980292e-01, -8.173388e-01, -9.113315e-02, 1.361192e-01,
        //     -9.167380e-01, -8.154047e-01, -5.336431e-01, 1.121887e-01,
        //     -6.467147e-01, -2.556555e-02, 6.625836e-01, -8.983362e-01,
        // };
        // double tb[MATRIX_SIZE * MATRIX_SIZE] = {
        //     5.341023e-01, 7.519617e-01, 6.208589e-01, -8.464509e-01,
        //     -9.621704e-01, 6.311373e-02, -6.231595e-01, 6.305478e-01,
        //     -4.952805e-01, 8.405219e-01, 7.726289e-01, 9.697820e-01,
        //     -4.036057e-01, 3.086230e-02, 1.412280e-01, -7.632966e-01,
        // };
        double ta[MATRIX_SIZE * MATRIX_SIZE] = {
            1.900078e-02, 8.548636e-02, -3.314322e-01, 2.440604e-02, -6.461445e-01,
-7.460251e-01, 5.706861e-01, -6.391164e-02,
            -3.681220e-01, 3.359572e-01, -8.358069e-01, 1.730207e-01, -7.762620e-01,
-9.918196e-01, -1.155304e-01, -6.060053e-01,
            4.034051e-01, -6.577163e-01, -4.593707e-01, 4.775171e-01, 7.053617e-01,
-9.989944e-01, -4.241922e-01, -1.056461e-01,
            -2.655067e-01, 6.615583e-01, -8.514753e-01, 7.433776e-01, 6.223437e-01,
-6.140962e-01, 7.067059e-01, 4.568796e-01,
            7.526247e-01, 7.238108e-01, 3.524515e-01, 1.559346e-01, 9.097746e-01,
1.735603e-01, 5.194394e-02, 9.321860e-01,
            -2.580107e-01, 5.384423e-01, 6.203171e-01, 9.294072e-01, 8.085664e-01,
-2.500817e-01, 4.568334e-01, 5.136816e-01,
            5.050480e-02, -8.948411e-01, -7.157754e-02, -3.329414e-01, -7.221695e-01,
-9.464281e-01, -4.673714e-01, -6.788623e-02,
            -1.443309e-01, 9.334593e-01, 1.383701e-01, -2.143441e-01, 3.723747e-01,
-7.046941e-02, 5.669680e-01, 7.379329e-01,
        };
        double tb[MATRIX_SIZE * MATRIX_SIZE] = {
            -7.319215e-01, 8.165651e-01, -4.029224e-02, 8.852518e-01, 5.361704e-01,
-2.435351e-01, -5.578819e-01, -3.161686e-01,
            6.921052e-01, 5.488294e-01, -9.377434e-01, -2.239507e-01, 5.038825e-01,
-1.625671e-01, 5.086909e-02, -8.637625e-02,
            -9.638247e-01, -2.678706e-01, 2.168604e-01, 8.610245e-01, -8.182718e-01,
-7.231037e-01, -1.094956e-02, 1.762435e-01,
            8.817096e-01, 8.885760e-01, 5.845223e-01, 8.227684e-01, 7.079334e-01,
-2.383066e-01, 7.650440e-01, 1.702999e-01,
            3.401642e-01, 7.696433e-01, -6.350064e-01, -3.245633e-01, -6.507111e-01,
-4.004399e-01, 5.476602e-01, -8.601459e-01,
            5.999431e-01, -1.096578e-01, 5.024259e-02, 6.562465e-02, -9.604593e-01,
-9.579758e-01, -8.819950e-01, 6.378629e-01,
            -7.135664e-01, -6.739431e-01, 7.975177e-01, 4.041545e-01, -1.618930e-01,
-5.223948e-01, -7.603972e-01, -1.936786e-01,
            2.209206e-01, -1.503654e-01, -3.824202e-01, 9.420213e-01, -4.072602e-01,
-9.257813e-01, 9.560251e-02, 1.709661e-01,
        };
        double tc[MATRIX_SIZE * MATRIX_SIZE] = {
            0
        };

        memcpy(a, ta, sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        memcpy(b, tb, sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        memcpy(c, tc, sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);

        int section_y_size = ceil(MATRIX_SIZE / SECTION_NUM);
        int section_y_begin, section_y_end;
        printf("main process: %d\n", rank);
        for (int i=1; i<SECTION_NUM; i++) {
            section_y_begin = i * section_y_size;
            section_y_end = min((i + 1) * section_y_size, MATRIX_SIZE);
            // printf("main section_y_begin: %d, section_y_end: %d\n", section_y_begin,
section_y_end);
            int calc_info[2] = {section_y_begin, section_y_end};
            MPI_Send(a, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                    i, 0, MPI_COMM_WORLD);
            MPI_Send(b, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                    i, 1, MPI_COMM_WORLD);
            MPI_Send(c, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                    i, 2, MPI_COMM_WORLD);
            MPI_Send(calc_info, 2, MPI_INT,
                    i, 3, MPI_COMM_WORLD);
            // printf("main process send data to process %d\n", i);
        }

        // calculate the first section
        section_y_begin = 0;
        section_y_end = min((1) * section_y_size, MATRIX_SIZE);
        for (int i=section_y_begin; i<section_y_end; i++) {
            for (int j=0; j<n; j++) {
                for (int p=0; p<k; p++) {
                    C(i, j) += A(i, p) * B(p, j);
                }
            }
        }

        // wait for sub process
        for (int i=1; i<SECTION_NUM; i++) {
            MPI_Recv(c + i * section_y_size * n, n * (min((i + 1) * section_y_size,
MATRIX_SIZE) - i * section_y_size),
                    MPI_DOUBLE, i, 4, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
            // printf("main process receive data from process %d\n", i);
        }

        FILE* f = fopen("output.m", "a");
        fprintf(f, "main process: %d * %d\n", MATRIX_SIZE, MATRIX_SIZE);
        fprintf(f, "\nA="); print_matrix(MATRIX_SIZE, MATRIX_SIZE, a, MATRIX_SIZE, f);
        fprintf(f, "\nB="); print_matrix(MATRIX_SIZE, MATRIX_SIZE, b, MATRIX_SIZE, f);
        fprintf(f, "\nC="); print_matrix(MATRIX_SIZE, MATRIX_SIZE, c, MATRIX_SIZE, f);
        free(a), free(b), free(c);
        printf(".::main process %d finish\n", rank);
    } else {
        if (rank >= SECTION_NUM) {
            printf("process %d is not needed\n", rank);
            MPI_Finalize();
            return 0;
        }
        // here is sub process
        double* a = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        double* b = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        double* c = (double*)malloc(sizeof(double) * MATRIX_SIZE * MATRIX_SIZE);
        int calc_info[2];

        MPI_Recv(a, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                0, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(b, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                0, 1, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(c, MATRIX_SIZE * MATRIX_SIZE, MPI_DOUBLE,
                0, 2, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Recv(calc_info, 2, MPI_INT,
                0, 3, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        int section_y_begin = calc_info[0], section_y_end = calc_info[1];

        for (int i=section_y_begin; i<section_y_end; i++) {
            for (int j=0; j<n; j++) {
                for (int p=0; p<k; p++) {
                    C(i, j) += A(i, p) * B(p, j);
                }
            }
        }

        MPI_Send(c + section_y_begin * n, n * (section_y_end - section_y_begin),
MPI_DOUBLE,
                0, 4, MPI_COMM_WORLD);
        free(a), free(b), free(c);
        // printf("[info]sub process %d finish\n", rank);
    }

MPI_Finalize();

}
```