**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет информационных технологий**

**Кафедра параллельных вычислений**

# ОТЧЕТ

## О ВЫПОЛНЕНИИ ПРАКТИЧЕСКОЙ РАБОТЫ

«Умножение матрицы на матрицу в MPI 2D решетке»

студента 2 курса, группы 21206

**Мельникова Никиты Сергеевича**

Направление 09.03.01 – «Информатика и вычислительная техника»

Преподаватель:
к.т.н., доцент
А.Ю. Власенко

Новосибирск 2023

# СОДЕРЖАНИЕ

# ЦЕЛЬ

Освоение концепции MPI-коммуникаторов и декартовых топологий, а также концепции произвольных типов данных.

# ЗАДАНИЕ

1. Реализовать параллельный алгоритм умножения матрицы на матрицу при 2D решетке процессов (приложение 1).
2. Исследовать производительность параллельной программы при фиксированном размере матрицы в зависимости от размера решетки: 2x12, 3x8, 4x6, 6x4, 8x3, 12x2. Размер матриц подобрать таким образом, чтобы худшее из времен данного набора было не менее 30 сек.
3. Выполнить профилирование программы при использовании 8-и ядер с решетками 2x4, 4x2.

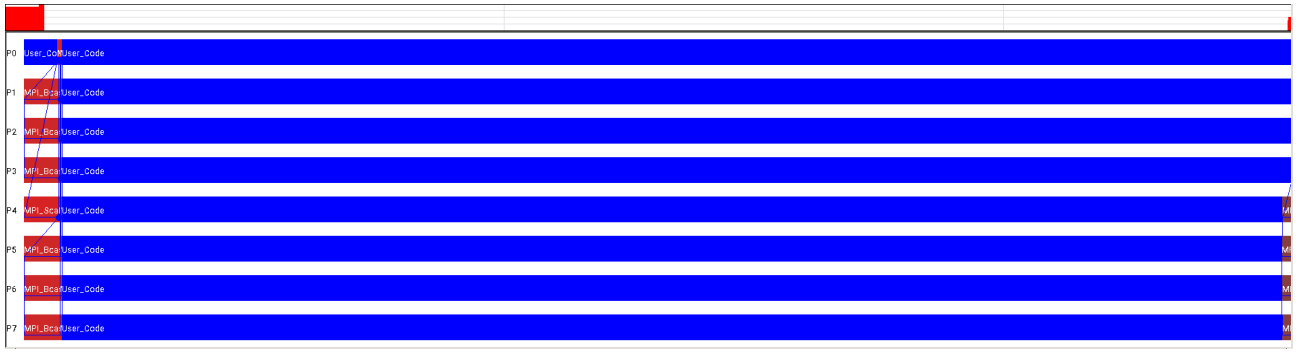
# ОПИСАНИЕ РАБОТЫ

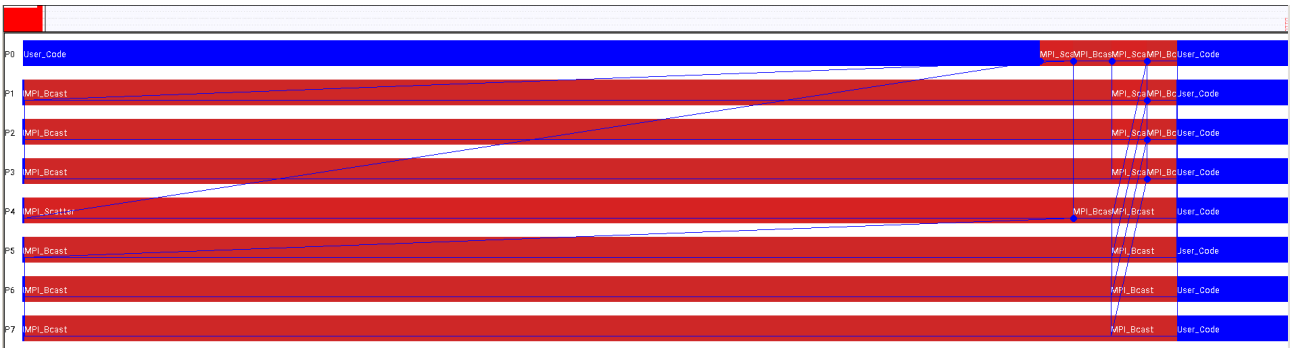
1. Был реализован (листинг 1) параллельный алгоритм  умножения матрицы на матрицу при 2D решетке процессов (приложение 1).
2. Было замерено время работы программы на кластере НГУ в зависимости от размера решетки: 2x12, 3x8, 4x6, 6x4, 8x3, 12x2. Размер матриц подобран таким образом, чтобы худшее из времен данного набора было не менее 30 сек (N1 = 10200, N2 = 3000, N3 = 12000).

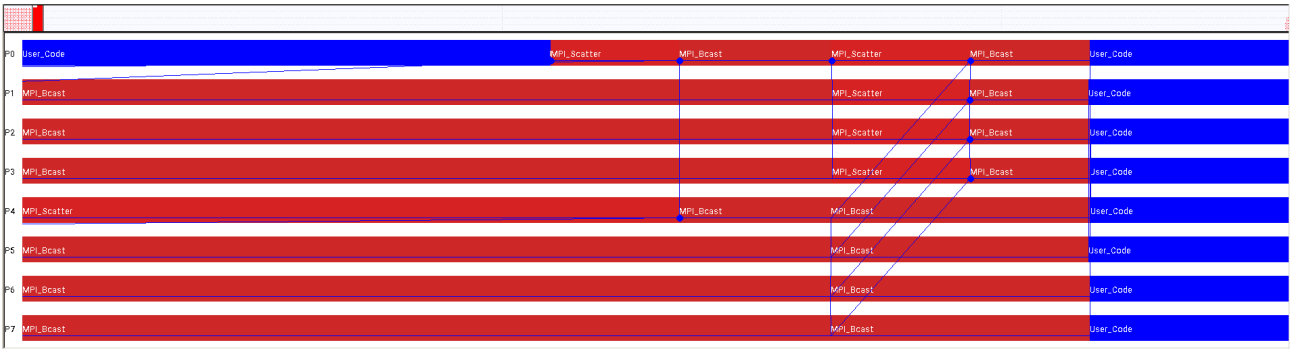
| Решётка P1xP2 | Время, с. |
|---|---|
| 2x12 | 44,576077 |
| 3x8 | 44,666096 |
| 4x6 | 44,691857 |
| 6x4 | 44,458719 |
| 8x3 | 44,605291 |
| 12x2 | 45,174802 |

3. Было выполнено профилирование программы при использовании 8-и ядер с решетками 2x4, 4x2 средствами Intel Trace Analyzer and Collector (ITAC).

2x4

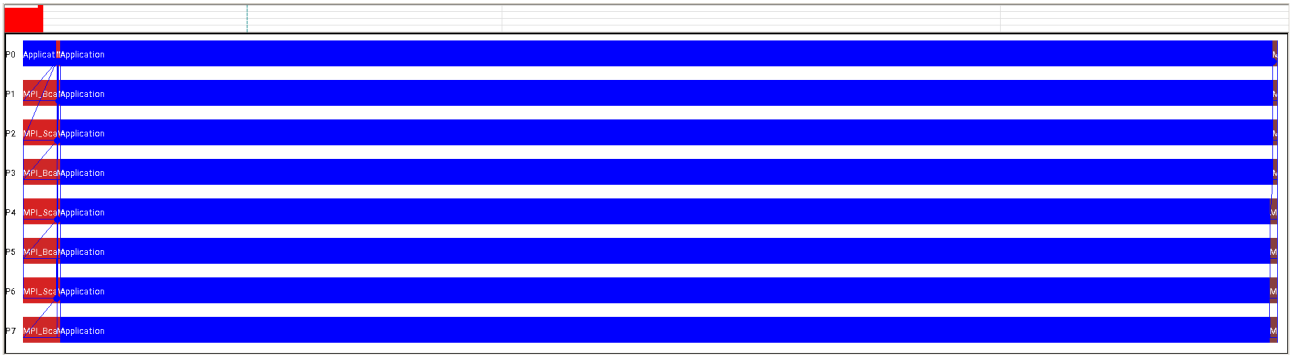


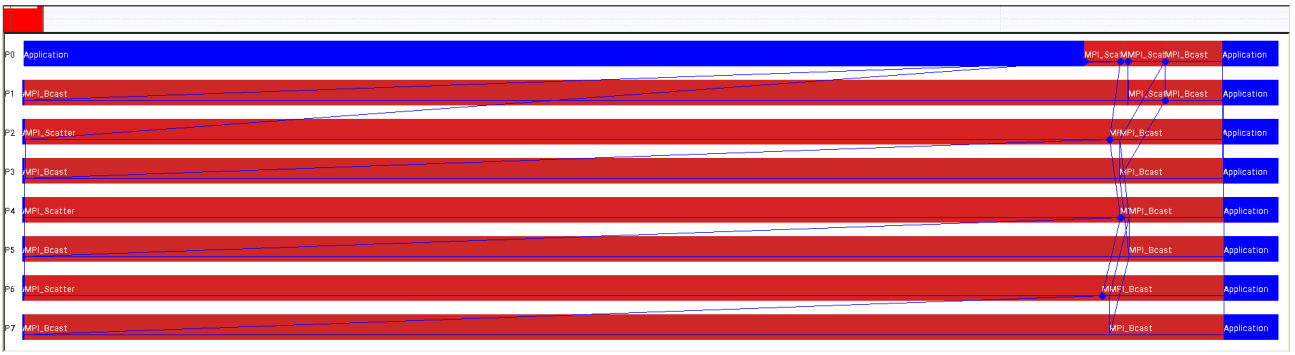


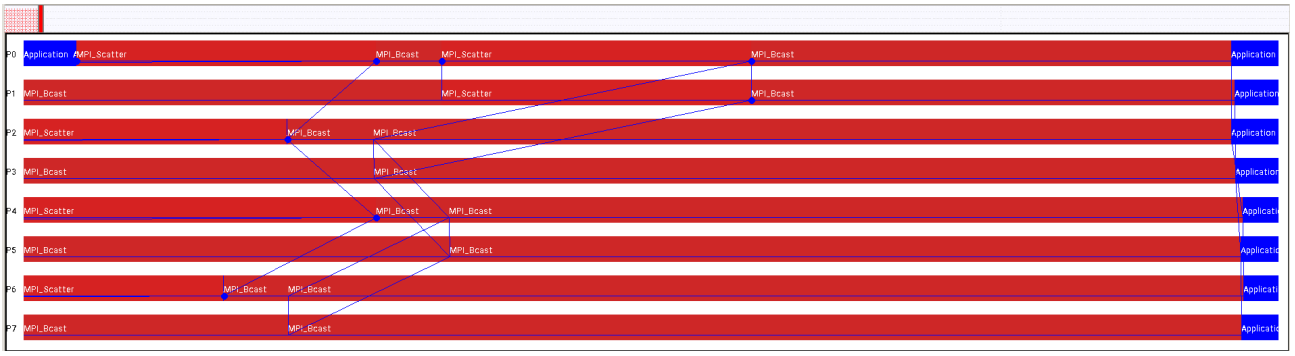




| Name | TSelf | TSelf | TTotal | #Calls | TSelf /Call |
|---|---|---|---|---|---|
| Group All_Processes | | | | | |
| User_Code | 803.731 s | | 832.479 s | 8 | 100.466 s |
| MPI_Cart_sub | 376e-6 s | | 376e-6 s | 16 | 23.5e-6 s |
| MPI_Comm_size | 5e-6 s | | 5e-6 s | 8 | 625e-9 s |
| MPI_Comm_rank | 5e-6 s | | 5e-6 s | 8 | 625e-9 s |
| MPI_Type_contiguous | 11e-6 s | | 11e-6 s | 8 | 1.375e-6 s |
| MPI_Finalize | 2.848e-3 s | | 2.848e-3 s | 8 | 356e-6 s |
| MPI_Scatter | 3.29567 s | | 3.29567 s | 6 | 549.278e-3 s |
| MPI_Bcast | 18.8501 s | | 18.8501 s | 16 | 1.17813 s |
| MPI_Cart_create | 633e-6 s | | 633e-6 s | 8 | 79.125e-6 s |
| MPI_Type_free | 124e-6 s | | 124e-6 s | 40 | 3.1e-6 s |
| MPI_Type_commit | 91e-6 s | | 91e-6 s | 40 | 2.275e-6 s |
| MPI_Cart_coords | 80e-6 s | | 80e-6 s | 8 | 10e-6 s |
| MPI_Wtime | 18e-6 s | | 18e-6 s | 2 | 9e-6 s |
| MPI_Type_vector | 69e-6 s | | 69e-6 s | 16 | 4.3125e-6 s |
| MPI_Type_create_resized | 466e-6 s | | 466e-6 s | 16 | 29.125e-6 s |
| MPI_Gather | 45.136e-3 s | | 45.136e-3 s | 8 | 5.642e-3 s |
| MPI_Gatherv | 6.55186 s | | 6.55186 s | 8 | 818.982e-3 s |

4x2






| Name | TSelf | TSelf | TTotal | #Calls | TSelf /Call |
|---|---|---|---|---|---|
| Group All_Processes | | | | | |
| Group Application | 806.023 s | | 832.283 s | 8 | 100.753 s |
| MPI_Cart_sub | 370e-6 s | | 370e-6 s | 16 | 23.125e-6 s |
| MPI_Comm_size | 6e-6 s | | 6e-6 s | 8 | 750e-9 s |
| MPI_Comm_rank | 0 s | | 0 s | 8 | 0 s |
| MPI_Type_contiguous | 8e-6 s | | 8e-6 s | 8 | 1e-6 s |
| MPI_Finalize | 2.735e-3 s | | 2.735e-3 s | 8 | 341.875e-6 s |
| MPI_Scatter | 8.73369 s | | 8.73369 s | 6 | 1.45562 s |
| MPI_Bcast | 13.3876 s | | 13.3876 s | 16 | 836.726e-3 s |
| MPI_Cart_create | 660e-6 s | | 660e-6 s | 8 | 82.5e-6 s |
| MPI_Type_free | 129e-6 s | | 129e-6 s | 40 | 3.225e-6 s |
| MPI_Type_commit | 100e-6 s | | 100e-6 s | 40 | 2.5e-6 s |
| MPI_Cart_coords | 59e-6 s | | 59e-6 s | 8 | 7.375e-6 s |
| MPI_Wtime | 20e-6 s | | 20e-6 s | 2 | 10e-6 s |
| MPI_Type_vector | 67e-6 s | | 67e-6 s | 16 | 4.1875e-6 s |
| MPI_Type_create_resized | 468e-6 s | | 468e-6 s | 16 | 29.25e-6 s |
| MPI_Gather | 63.677e-3 s | | 63.677e-3 s | 8 | 7.95962e-3 s |
| MPI_Gatherv | 4.07072 s | | 4.07072 s | 8 | 508.84e-3 s |

# ЗАКЛЮЧЕНИЕ

В ходе практической работы удалось ознакомиться с концепцией MPI-коммуникаторов и декартовых топологий. Были приобретены навыки работы с произвольными типами данных при помощи MPI-функций.

В результате исследования наибольшее быстродействие показала программа при использовании решетки 6x4 (время работы такой программы составило ≈44,46 секунд).

## **Приложение 1.** *Описание алгоритма*

1. Создание решетки процессов p1 x p2.
2. Генерация матриц A[n1 x n2] и B[n2 x n3] на процессе с координатами (0;0) как одномерных массивов.
3. Раздача матрицы A по горизонтальным полосам на вертикальную линейку процессов (0;0), (1;0), (2;0), …, (p1 - 1; 0)
4. Определение нового производного типа данных для выбора из матрицы B вертикальных полос.
5. Раздача матрицы B по вертикальным полосам на горизонтальную линейку процессов (0;0), (0;1), (0;2), …, (0; p2 – 1) таким образом, что каждому процессу высылается только 1 элемент производного типа.
6. Каждый из процессов в левой вертикальной колонке ( (1;0), (2;0), …, (p1 - 1; 0) ) при помощи MPI_Bcast раздает свою полосу матрицы A всем процессам своей горизонтали. Т.е. процесс (1;0) раздает свою полосу процессам (1;1), (1;2),…
7. То же с полосами матрицы B, которые процессы первой горизонтали раздают по своим вертикальным столбцам решетки процессов (MPI_Bcast).
8. Теперь на каждом процессе есть по полосе A и по столбцу B, перемножаем, получаем миноры C.
9. Собираем всю C на процессе (0;0).

**Приложение 2.** *Листинг параллельной программы на языке Си*

*Код компиляции:* `mpiicpc main.c -o exec`

*Листинг 1 — Исходный код программы*

```
#include <mpi.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define N1    10200
#define N2    3000
#define N3    12000
#define NDIMS 2

double *gen_matrix(size_t matrix_rows, size_t matrix_cols) {
    double *A = (double *) malloc(matrix_rows * matrix_cols *
                                  sizeof(double));
    srand(0);

    for (size_t i = 0; i < matrix_rows; ++i) {
        for (size_t j = 0; j < matrix_cols; ++j) {
            A[i * matrix_cols + j] = (double) rand() / RAND_MAX *
                                     (double) rand();
        }
    }
    return A;
}

int main(int argc, char *argv[]) {
    if (argc < 3) {
        fprintf(stderr, "Not enough arguments\n");
        return EXIT_FAILURE;
    }

    const int P1 = atoi(argv[1]);
    const int P2 = atoi(argv[2]);

    int size      = 0;
    int grid_rank = 0;
    int row_width = N1 / P1;
    int col_width = N3 / P2;
    double time_start;
    double time_end;

    MPI_Init(&argc, &argv);
```

```
MPI_Comm_size(MPI_COMM_WORLD, &size);

if (size != P1 * P2) {
    fprintf(stderr, "Expected %d, instead of %d processes\n",
            P1 * P2, size);
    MPI_Abort(MPI_COMM_WORLD, EXIT_FAILURE);
}

MPI_Comm COMM_GRID;
int dims[NDIMS] = {P1, P2};
int periods[NDIMS] = {false, false};
MPI_Cart_create(MPI_COMM_WORLD, NDIMS, dims,
                periods, false, &COMM_GRID);

MPI_Comm COMM_ROW;
int row_dims[NDIMS] = {false, true};
MPI_Cart_sub(COMM_GRID, row_dims, &COMM_ROW);

MPI_Comm COMM_COL;
int col_dims[NDIMS] = {true, false};
MPI_Cart_sub(COMM_GRID, col_dims, &COMM_COL);

int coords[NDIMS] = {0, 0};
MPI_Comm_rank(COMM_GRID, &grid_rank);
MPI_Cart_coords(COMM_GRID, grid_rank, NDIMS, coords);

MPI_Datatype TYPE_ROW;
MPI_Type_contiguous(N2 * row_width, MPI_DOUBLE, &TYPE_ROW);
MPI_Type_commit(&TYPE_ROW);

MPI_Datatype TYPE_COL, TYPE_COL_RESIZED;
MPI_Type_vector(N2,
                col_width, N3, MPI_DOUBLE, &TYPE_COL);
MPI_Type_commit(&TYPE_COL);
MPI_Type_create_resized(TYPE_COL,
                        0,
                        col_width * sizeof(double),
                        &TYPE_COL_RESIZED);
MPI_Type_commit(&TYPE_COL_RESIZED);

MPI_Datatype TYPE_MINOR, TYPE_MINOR_RESIZED;
MPI_Type_vector(row_width,
                col_width, N3, MPI_DOUBLE, &TYPE_MINOR);
MPI_Type_commit(&TYPE_MINOR);
MPI_Type_create_resized(TYPE_MINOR,
                        0,
```

```
                        col_width * sizeof(double),
                        &TYPE_MINOR_RESIZED);
MPI_Type_commit(&TYPE_MINOR_RESIZED);

double *A, *B, *C;
if (coords[0] == 0 && coords[1] == 0) {
    A = gen_matrix(N1, N2);
    B = gen_matrix(N2, N3);
    C = (double *) calloc(N1 * N3, sizeof(double));
    time_start = MPI_Wtime();
}

double *part_A = (double *) malloc(row_width * N2 * sizeof(double));
double *part_B = (double *) malloc(col_width * N2 * sizeof(double));
double *part_C = (double *) calloc(row_width * col_width,
                                   sizeof(double));

if (coords[1] == 0)
    MPI_Scatter(A, 1, TYPE_ROW,
                part_A, 1, TYPE_ROW, 0,
                COMM_COL);
MPI_Bcast(part_A, 1, TYPE_ROW, 0,
          COMM_ROW);

if (coords[0] == 0)
    MPI_Scatter(B, 1, TYPE_COL_RESIZED,
                part_B, col_width * N2, MPI_DOUBLE, 0,
                COMM_ROW);
MPI_Bcast(part_B, col_width * N2, MPI_DOUBLE, 0,
          COMM_COL);

for (int i = 0; i < row_width; ++i) {
    for (int k = 0; k < N2; ++k) {
        for (int j = 0; j < col_width; ++j)
            part_C[i * col_width + j] += part_A[i * N2 + k] *
                                         part_B[k * col_width + j];
    }
}

int *recvcounts = (int *) malloc(size * sizeof(int));
int *displs     = (int *) malloc(size * sizeof(int));
int displ = coords[0] * row_width * P2 + coords[1];
MPI_Gather(&displ, 1, MPI_INT,
           displs, 1, MPI_INT, 0, COMM_GRID);

for (int i = 0; i < size; ++i)  recvcounts[i] = 1;
```

```
    MPI_Gatherv(part_C, row_width * col_width, MPI_DOUBLE,
                C, recvcounts, displs, TYPE_MINOR_RESIZED,
                0, COMM_GRID);

    if (coords[0] == 0 && coords[1] == 0) {
        time_end = MPI_Wtime();
        printf("Time taken: %f sec.\n", time_end - time_start);
        free(A);
        free(B);
        free(C);
    }
    MPI_Type_free(&TYPE_ROW);
    MPI_Type_free(&TYPE_COL);
    MPI_Type_free(&TYPE_COL_RESIZED);
    MPI_Type_free(&TYPE_MINOR);
    MPI_Type_free(&TYPE_MINOR_RESIZED);
    free(recvcounts);
    free(displs);
    free(part_A);
    free(part_B);
    free(part_C);

    MPI_Finalize();
    return EXIT_SUCCESS;
}
```

*Листинг 2 — Скрипт для запуска на кластере НГУ*

```
#!/bin/bash

#PBS -l walltime=00:08:00
#PBS -l select=2:ncpus=12:mpiprocs=12:mem=8000m,place=scatter
#PBS -m n

cd $PBS_O_WORKDIR

MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
echo "Number of MPI process: $MPI_NP"

P1=2
P2=12
```

```
echo "P1 = $P1; P2 = $P2"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P1 $P2

echo "P1 = $P2; P2 = $P1"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P2 $P1

P1=3
P2=8

echo "P1 = $P1; P2 = $P2"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P1 $P2

echo "P1 = $P2; P2 = $P1"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P2 $P1

P1=4
P2=6

echo "P1 = $P1; P2 = $P2"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P1 $P2

echo "P1 = $P2; P2 = $P1"
mpirun -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P2 $P1
```

*Листинг 3 — Скрипт для запуска профилировщика на кластере НГУ*

```
#!/bin/bash

#PBS -l walltime=00:02:00
#PBS -l select=1:ncpus=8:mpiprocs=8:mem=4000m,place=pack
#PBS -m n

cd $PBS_O_WORKDIR
```

```
MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
echo "Number of MPI process: $MPI_NP"

P1=2
P2=4

mpirun -trace -machinefile $PBS_NODEFILE -np $MPI_NP ./exec $P1 $P2
```