# Homework 4

Xinyuan Miao
020033910009
mxinyuan@sjtu.edu.cn

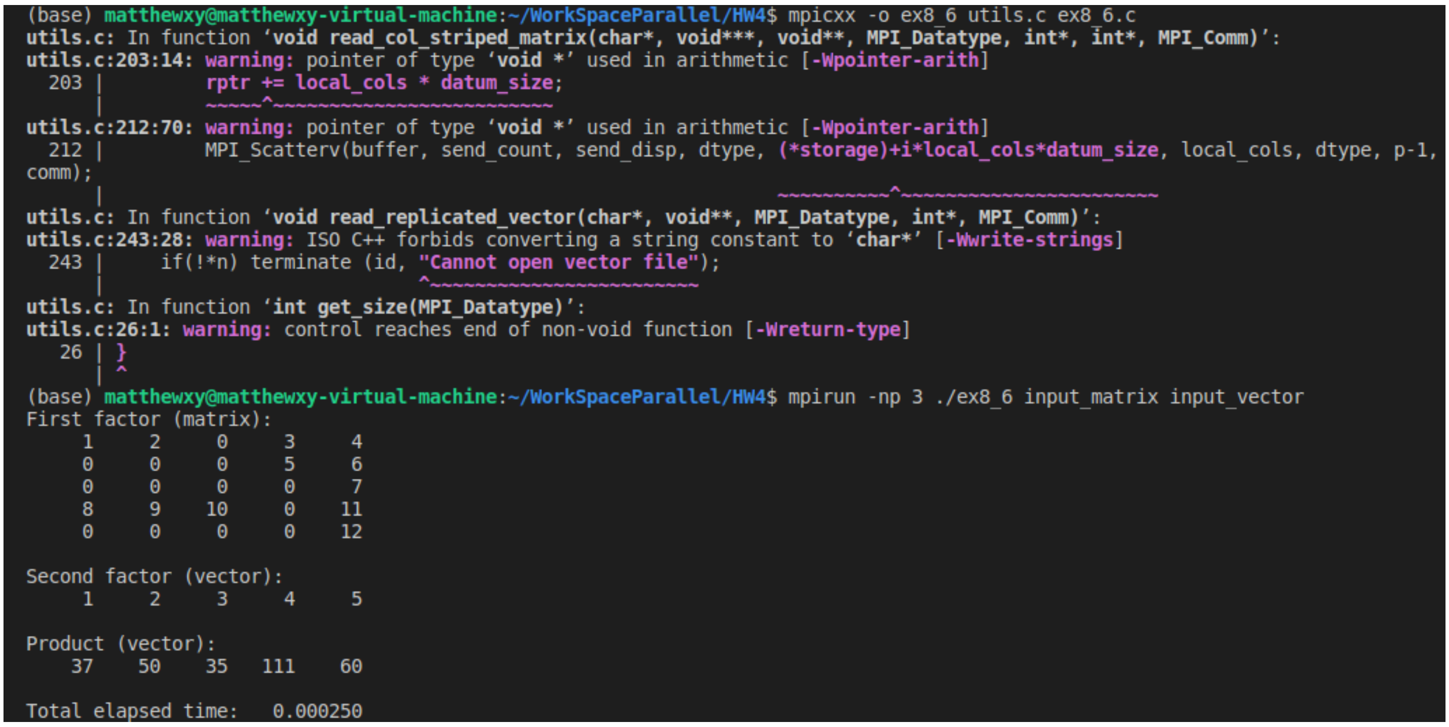
## Exercise 8.6

### Main Idea

The idea for the block-column split is simple, each processor gets some continuous lines and a copy of the second factor, say the vector-b. After the multiplication, each processor provides its partial product, which is a vector of length same as the vector-b. We call the API **MPI_Allreduce** to sum up all the partial products to get the final result, which is kept by every processor.

### Execution Results

We provide programs to generate the binary file containing the matrix and the vector that will be used as the input. The figure shows that the output of the program is correct.



```
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpicxx -o ex8_6 utils.c ex8_6.c
utils.c: In function ‘void read_col_striped_matrix(char*, void***, void**, MPI_Datatype, int*, int*, MPI_Comm)’:
utils.c:203:14: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  203 |         rptr += local_cols * datum_size;
      |         ~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~
utils.c:212:70: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  212 |         MPI_Scatterv(buffer, send_count, send_disp, dtype, (*storage)+i*local_cols*datum_size, local_cols, dtype, p-1,
comm);
      |                                                            ~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~
utils.c: In function ‘void read_replicated_vector(char*, void**, MPI_Datatype, int*, MPI_Comm)’:
utils.c:243:28: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
  243 |     if(!*n) terminate (id, "Cannot open vector file");
      |                            ^~~~~~~~~~~~~~~~~~~~~~~~~
utils.c: In function ‘int get_size(MPI_Datatype)’:
utils.c:26:1: warning: control reaches end of non-void function [-Wreturn-type]
   26 | }
      | ^
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 3 ./ex8_6 input_matrix input_vector
First factor (matrix):
     1     2     0     3     4
     0     0     0     5     6
     0     0     0     0     7
     8     9    10     0    11
     0     0     0     0    12

Second factor (vector):
     1     2     3     4     5

Product (vector):
    37    50    35   111    60

Total elapsed time:   0.000250
```

Figure 1: Execution result

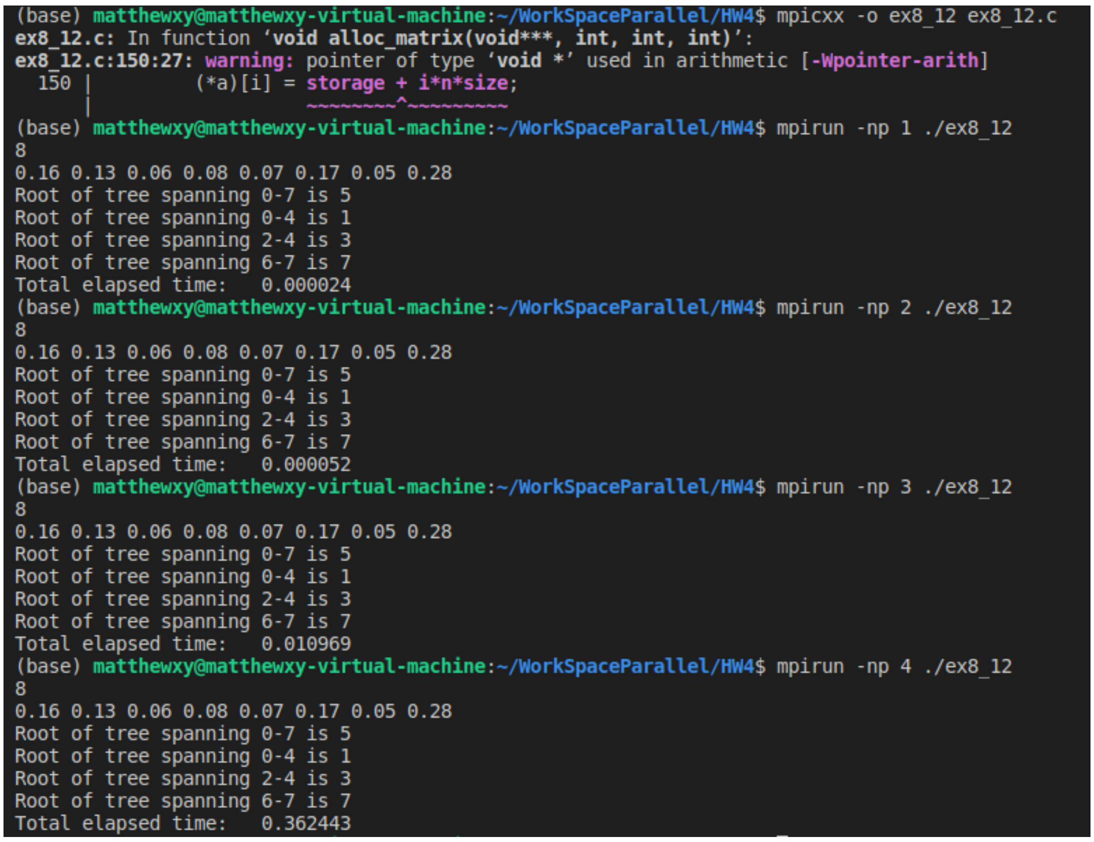
## Exercise 8.12

### Main Idea

According to the dynamic programming algorithm, we need to construct two matrices, say the root matrix and cost matrix and update the value of the upper triangular region in an iterative way. We can also find that the update of one position in the matrix requires the value of the left and below. Based on the observation, we update the matrix diagonal by diagonal, from the main diagonal to the minor ones. In each iterations, we allocate primitive tasks containing some continuous elements in the diagonal to multiple processors.
One key points is to map the coordinate values to the new coordinate system (because we have reorganized the elements). For the details, please refer to the code.

### Execution Results

The results of the execution proves the correctness of out program. However, we can easily find that parallelism does not help, at least on such a problem with relative small scale. One possible reason is that, as the iteration increases, the diagonal waiting to be updated is shorter, we actually do not have too many elements to allocate, therefore, the parallel efficiency decreases.



```
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpicxx -o ex8_12 ex8_12.c
ex8_12.c: In function ‘void alloc_matrix(void***, int, int, int)’:
ex8_12.c:150:27: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  150 |         (*a)[i] = storage + i*n*size;
      |                   ~~~~~~~~^~~~~~~~~
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 1 ./ex8_12
8
0.16 0.13 0.06 0.08 0.07 0.17 0.05 0.28
Root of tree spanning 0-7 is 5
Root of tree spanning 0-4 is 1
Root of tree spanning 2-4 is 3
Root of tree spanning 6-7 is 7
Total elapsed time:   0.000024
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 2 ./ex8_12
8
0.16 0.13 0.06 0.08 0.07 0.17 0.05 0.28
Root of tree spanning 0-7 is 5
Root of tree spanning 0-4 is 1
Root of tree spanning 2-4 is 3
Root of tree spanning 6-7 is 7
Total elapsed time:   0.000052
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 3 ./ex8_12
8
0.16 0.13 0.06 0.08 0.07 0.17 0.05 0.28
Root of tree spanning 0-7 is 5
Root of tree spanning 0-4 is 1
Root of tree spanning 2-4 is 3
Root of tree spanning 6-7 is 7
Total elapsed time:   0.010969
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 4 ./ex8_12
8
0.16 0.13 0.06 0.08 0.07 0.17 0.05 0.28
Root of tree spanning 0-7 is 5
Root of tree spanning 0-4 is 1
Root of tree spanning 2-4 is 3
Root of tree spanning 6-7 is 7
Total elapsed time:   0.362443
```

Figure 2: Execution result

## Exercise 9.7

### Main Idea

To perform matrix-vector multiplication in a manager/worker-style, we define the duties of the manager and workers.

For the manager:

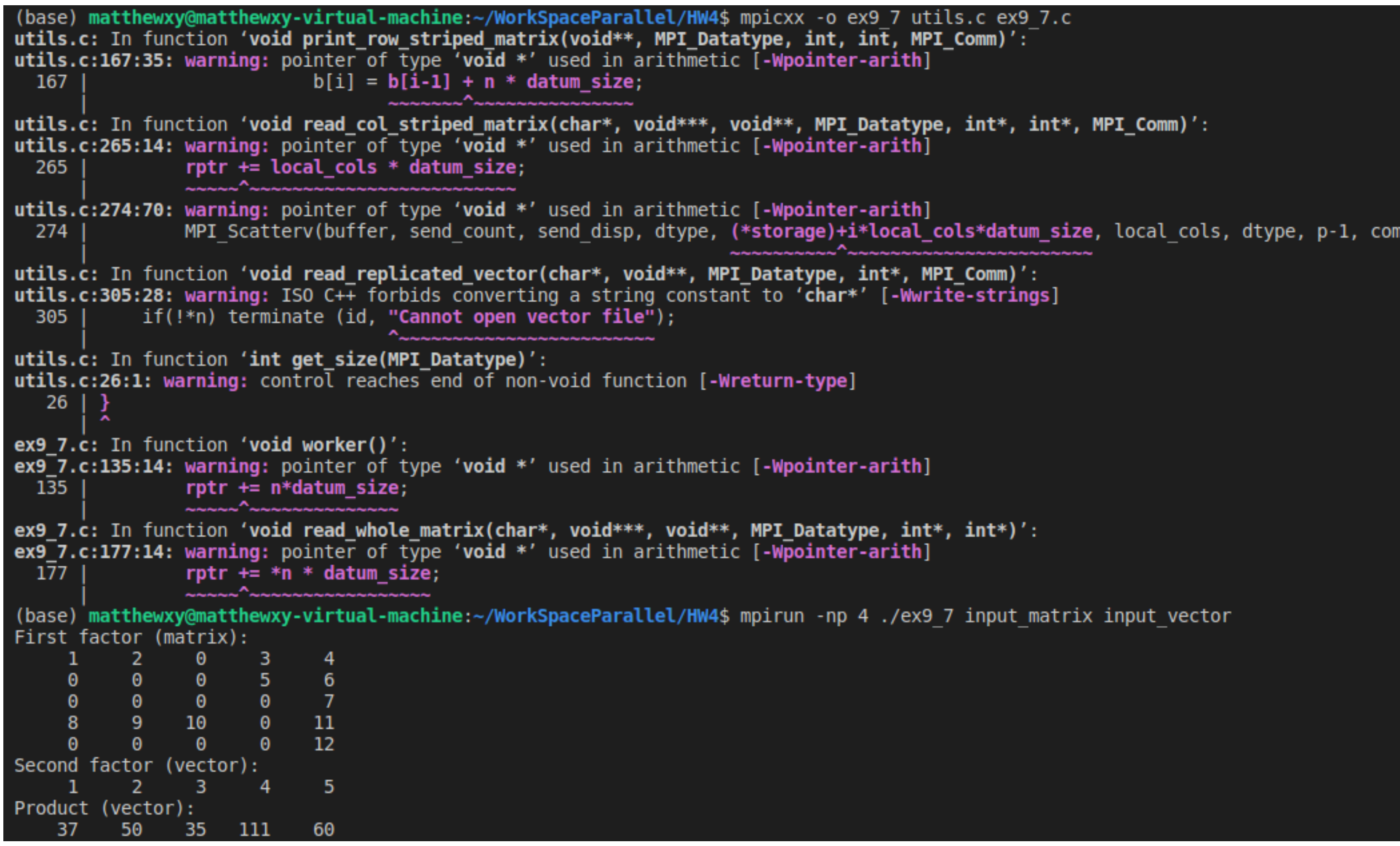
1. Calculate the number of local rows that will be assigned to each worker and notify the workers.
2. Read the vector-b and send the replicated vector to each worker.
3. Read the while matrix, split it into parts according to the number of worker, send the sub matrices to the workers.
4. Receive partial products from the workers, combine them to form the final product vector, print it to standard output.

For each worker:

1. Get the number of rows and columns it will be assigned from the manager.
2. Receive the replicated vector from the manager.
3. Receive the sub matrix from the manager.
4. Perform matrix multiplication and get the partial product.
5. Send the partial product to the manager.

## Execution Results

We take the same matrix and vector used in the ex_8.6 as the input, and we get the same correct product.



```
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpicxx -o ex9_7 utils.c ex9_7.c
utils.c: In function ‘void print_row_striped_matrix(void**, MPI_Datatype, int, int, MPI_Comm)’:
utils.c:167:35: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  167 |                      b[i] = b[i-1] + n * datum_size;
      |                             ~~~~~~~^~~~~~~~~~~~~~~~
utils.c: In function ‘void read_col_striped_matrix(char*, void***, void**, MPI_Datatype, int*, int*, MPI_Comm)’:
utils.c:265:14: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  265 |         rptr += local_cols * datum_size;
      |         ~~~~~^~~~~~~~~~~~~~~~~~~~~~~~~~
utils.c:274:70: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  274 |         MPI_Scatterv(buffer, send_count, send_disp, dtype, (*storage)+i*local_cols*datum_size, local_cols, dtype, p-1, com
      |                                                            ~~~~~~~~~~^~~~~~~~~~~~~~~~~~~~~~~~
utils.c: In function ‘void read_replicated_vector(char*, void**, MPI_Datatype, int*, MPI_Comm)’:
utils.c:305:28: warning: ISO C++ forbids converting a string constant to ‘char*’ [-Wwrite-strings]
  305 |     if(!*n) terminate (id, "Cannot open vector file");
      |                            ^~~~~~~~~~~~~~~~~~~~~~~~~
utils.c: In function ‘int get_size(MPI_Datatype)’:
utils.c:26:1: warning: control reaches end of non-void function [-Wreturn-type]
   26 | }
      | ^
ex9_7.c: In function ‘void worker()’:
ex9_7.c:135:14: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  135 |         rptr += n*datum_size;
      |         ~~~~~^~~~~~~~~~~~~~~
ex9_7.c: In function ‘void read_whole_matrix(char*, void***, void**, MPI_Datatype, int*, int*)’:
ex9_7.c:177:14: warning: pointer of type ‘void *’ used in arithmetic [-Wpointer-arith]
  177 |         rptr += *n * datum_size;
      |         ~~~~~^~~~~~~~~~~~~~~~~~
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 4 ./ex9_7 input_matrix input_vector
First factor (matrix):
     1     2     0     3     4
     0     0     0     5     6
     0     0     0     0     7
     8     9    10     0    11
     0     0     0     0    12
Second factor (vector):
     1     2     3     4     5
Product (vector):
    37    50    35   111    60
```

Figure 3: Execution result

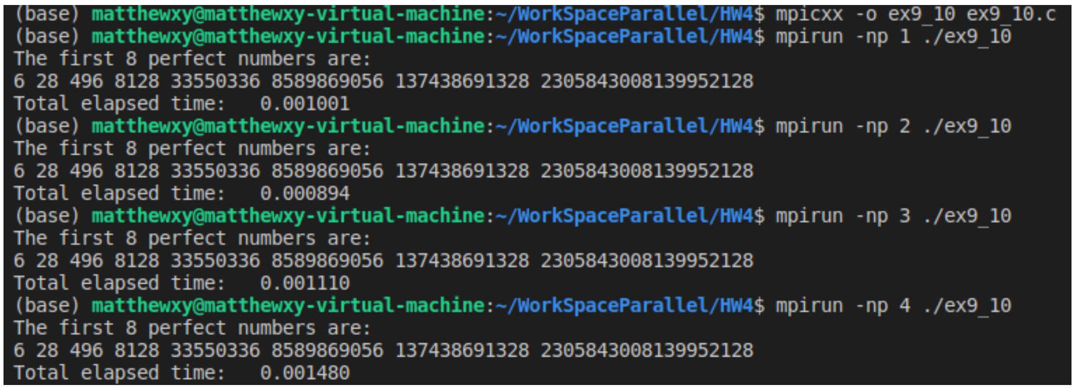
# Exercise 9.10

## Main Idea

We provide a simple sequential solution to search for the perfect number based on the methods found by Euclid.
If we have more processors to perform the search, we can do it in a manager/worker fashion. The process contains multiple iterations. In each iteration, workers fetch the task in an interleaved fashion. Suppose we have three worker and begin from the number 4. In the first iteration, the first worker fetches 4, the second fetches 4*2, the third fetches 4*4. Each worker checks whether the fetched number minus 1 is a prime, if so, return the (number-1)*number/2, which is a perfect number to the manager otherwise, return 0. In the second iteration, the number begins from 4*8, and the process is similar to the previous.
After the manager receive 8 perfect numbers, the manager seed termination message to each worker.

## Execution Results

The result shows that parallelism seems not help to this problem. One possible reason is the overhead brought by the synchronization between every two iterations. Meanwhile, even in some iteration, one worker find the eighth perfect number, the manager has to wait for the results from all the workers in this iteration.



```
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpicxx -o ex9_10 ex9_10.c
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 1 ./ex9_10
The first 8 perfect numbers are:
6 28 496 8128 33550336 8589869056 137438691328 2305843008139952128
Total elapsed time:   0.001001
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 2 ./ex9_10
The first 8 perfect numbers are:
6 28 496 8128 33550336 8589869056 137438691328 2305843008139952128
Total elapsed time:   0.000894
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 3 ./ex9_10
The first 8 perfect numbers are:
6 28 496 8128 33550336 8589869056 137438691328 2305843008139952128
Total elapsed time:   0.001110
(base) matthewxy@matthewxy-virtual-machine:~/WorkSpaceParallel/HW4$ mpirun -np 4 ./ex9_10
The first 8 perfect numbers are:
6 28 496 8128 33550336 8589869056 137438691328 2305843008139952128
Total elapsed time:   0.001480
```

Figure 4: Execution result