2 завдання

Синхронні методи:

Send(buffer, count, datatype, dest, comm)

Надсилає повідомлення з `buffer` (буфер) з `count` елементів типу `datatype` процесу з рангом `dest` в комунікаторі `comm`.

Recv(buffer, count, datatype, source, comm, status)

Приймає повідомлення в `buffer` з `count` елементів типу `datatype` від процесу з рангом `source` в комунікаторі `comm`.

Асинхронні методи:

ISend(buffer, count, datatype, dest, comm, request)

Асинхронне надсилання повідомлення з `buffer` з `count` елементів типу `datatype` процесу з рангом `dest` в комунікаторі `comm`. Завершення надсилання можна відстежувати за допомогою запиту `request`.

IRecv(buffer, count, datatype, source, comm, request)

Асинхронний прийом повідомлення в `buffer` з `count` елементів типу `datatype` від процесу з рангом `source` в комунікаторі `comm`. Завершення прийому можна відстежувати за допомогою запиту `request`.

Прикоал

```
#include <mpi.h>

#include <stdio.h>


int main(int argc, char* argv[]) {

  int rank, size;


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

```
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  int buffer[10];

  if (rank == 0) {
      // Процес 0 надсилає повідомлення процесу 1
      for (int i = 0; i < 10; i++) {
      buffer[i] = i;
      }
      MPI_Send(buffer, 10, MPI_INT, 1, 0, MPI_COMM_WORLD);
      printf("Процес 0: Повідомлення надіслано\n");
  } else if (rank == 1) {
      // Процес 1 отримує повідомлення від процесу 0
      MPI_Recv(buffer, 10, MPI_INT, 0, 0, MPI_COMM_WORLD,
MPI_STATUS_IGNORE);
      printf("Процес 1: Повідомлення отримано\n");

      // Виводить отримані дані
      for (int i = 0; i < 10; i++) {
      printf("buffer[%d] = %d\n", i, buffer[i]);
      }
  }

  MPI_Finalize();
```

```
  return 0;

}
```

1 завдання

1. Ініціалізація:

- Розсилаються блоки Aii та Bii до всіх процесів Rij.
- Знаходиться їх добуток і присвоюється результуючій матриці процесу Cij = Aii * Bii.

2. На кожній ітерації:

- Виконується циклічна пересилка блоків A (по другому індексу) і B (по першому індексу):
    - Якщо i + 1 == q, то i + 1 := 0.
    - Знаходиться добуток блоків Ai(i + 1), B(i + 1)j і додається до результуючої матриці процесу Cij += Ai(i + k)mod q * B(i + k)mod q.j, де k - номер ітерації.

3. Фіналізація:

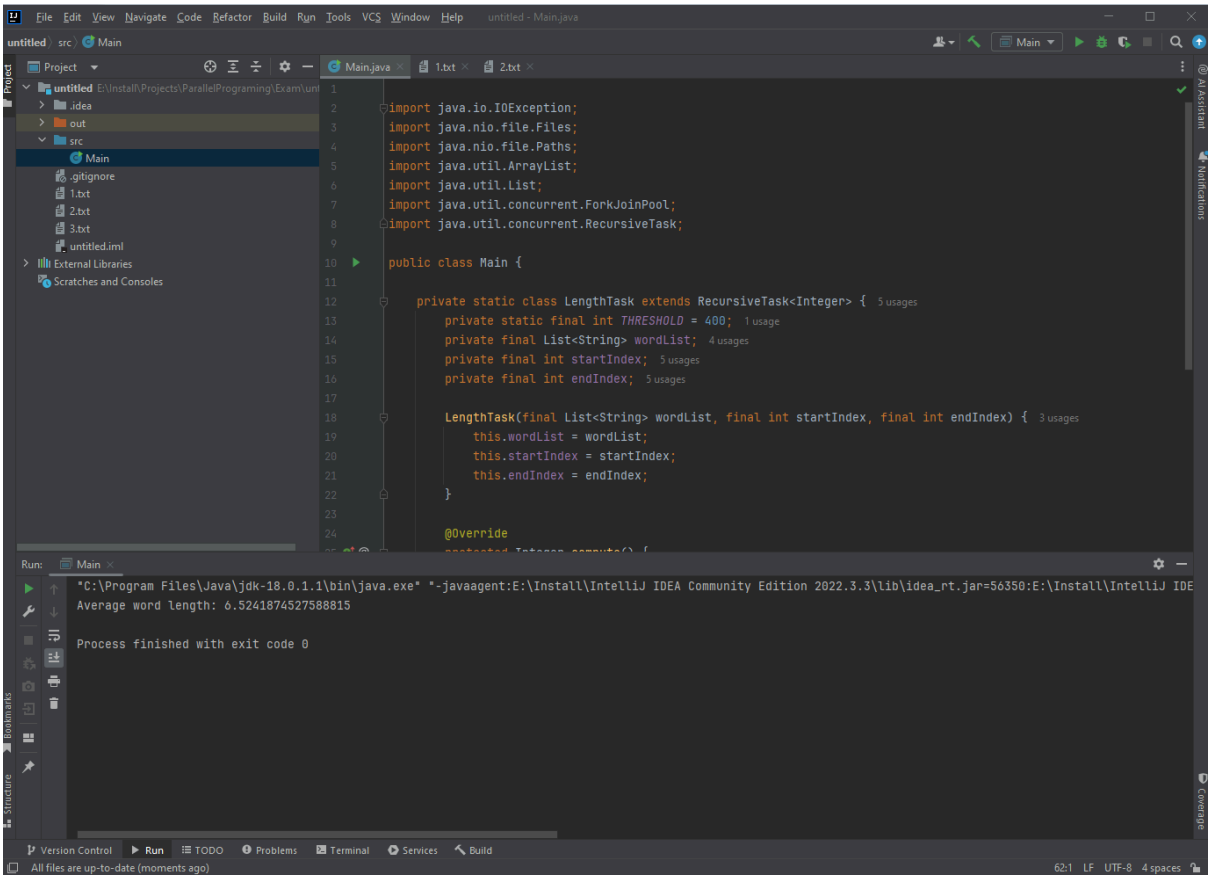
- Результуючі матриці процесів передаються в головний процес, де з них складається результуюча матриця.

Якщо розмір матриць A і B NxN, а кількість блоків по горизонталі і діагоналі d, то алгоритм реалізується на 2 процесах, а розмір блоків матриць t = N/q.

3 завдання

```java
package org.example;

import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Paths;
import java.util.ArrayList;
import java.util.List;
import java.util.concurrent.ForkJoinPool;
import java.util.concurrent.RecursiveTask;

public class Main {

   private static class LengthTask extends RecursiveTask<Integer> {
       private static final int THRESHOLD = 400;
       private final List<String> words;
```

```
        private final int start;
        private final int end;

        LengthTask(final List<String> words, final int start, final int end) {
            this.words = words;
            this.start = start;
            this.end = end;
        }

        @Override
        protected Integer compute() {
            if (end - start <= THRESHOLD) {
                return computeDirectly();
            } else {
                int mid = (start + end) / 2;
                final var rhs = new LengthTask(words, mid, end);
                rhs.fork();
                final var lhs = new LengthTask(words, start, mid);

                final var lhsRes = lhs.compute();
                final var rhsRes = rhs.join();
                return lhsRes + rhsRes;
            }
        }

        private int computeDirectly() {
            var sum = 0;
            for (int i = start; i < end; i++) {
                sum += words.get(i).length();
            }
            return sum;
        }
    }

    public static void main(String[] args) throws IOException {
        String[] fileNames = {"1.txt", "2.txt"};
        List<String> all_words = new ArrayList<>();
        for(var fileName : fileNames) {
            final var words = Files.readAllLines(Paths.get(fileName));
            all_words.addAll(words);
        }
        final var pool = ForkJoinPool.commonPool();
        final var totalLength = pool.invoke(new LengthTask(all_words, 0,
all_words.size()));
        double averageLength = (double) totalLength / (double)
all_words.size();
        System.out.println("Average word length: " + averageLength);
    }

}
```



4 завдання

```cpp
#include <boost/mpi.hpp>
#include <boost/serialization/vector.hpp>

static constexpr int MAIN_RANK = 0;
   static constexpr int MAX_SIZE = 10000;
   static constexpr int FROM_MAIN_THREAD_TAG = 1;
   static constexpr int FROM_CHILD_THREAD_TAG = 2;


   using Matrix = std::vector<std::vector<int>>;

   template<typename T>
   void print(const std::vector<T>& vec) {
      for(const auto v : vec) {
         std::cout << v << ", ";
      }
   }

   void print(const Matrix& mat) {
      for(const auto& row : mat) {
         print(row); std::printf("\n");
```

```
    }
  }

  Matrix generate_matrix(const size_t side_size) {
    auto mat = Matrix(side_size, std::vector<int>(side_size));
    for(auto& v : mat) {
      std::generate(std::begin(v), std::end(v),
      [] { return std::rand() % MAX_SIZE; } );
    }
    return mat;
  }

  namespace calc_average_vector {

    int calc_child_ranks_num(const boost::mpi::communicator& world) {
      return world.size() - 1;
    }

    static void main_rank_send_side_size(const
boost::mpi::communicator& world, const Matrix& mat) {
      for(auto i = 1; i < world.size(); ++i) {
        world.send(i, FROM_MAIN_THREAD_TAG, mat.size());
      }
    }

    static size_t child_rank_recv_side_size(const
boost::mpi::communicator& world) {
      size_t size = 0;
      world.recv(MAIN_RANK, FROM_MAIN_THREAD_TAG, size);
      return size;
    }

    template<typename T>
    static void index_rank_iterator(const boost::mpi::communicator&
world, const size_t side_size, T&& callable) {
      const auto child_ranks_num = calc_child_ranks_num(world);
      for(size_t i = 0; i < side_size; ++i) {
        const auto child_index = static_cast<int>((i %
child_ranks_num) + 1);
        callable(child_index, i);
      }
    }

    static double calc_average(const std::vector<int>& v) {
      return static_cast<double>(std::accumulate(std::begin(v),
std::end(v), 0, std::plus<>()))
        / static_cast<double>(v.size());
    }

    void child_rank(const boost::mpi::communicator& world) {
      const auto side_size = child_rank_recv_side_size(world);
```

```
        index_rank_iterator(world, side_size,
           [&world](const int child_index, const int mat_index) {
              if(child_index == world.rank()) {
                 auto row = std::vector<int>();
                 world.recv(MAIN_RANK, FROM_MAIN_THREAD_TAG, row);
                 world.send(MAIN_RANK, FROM_CHILD_THREAD_TAG,
calc_average(row));
              }
           }
        );
     }

     std::vector<double> main_rank(const boost::mpi::communicator&
world, const Matrix& mat) {

        main_rank_send_side_size(world, mat);

        auto send_requests = std::vector<boost::mpi::request>();
        auto recv_requests = std::vector<boost::mpi::request>();
        auto average_values = std::vector<double>();

        send_requests.reserve(mat.size());
        recv_requests.reserve(mat.size());
        average_values.resize(mat.size());

        index_rank_iterator(world, mat.size(),
           [&](const int child_index, const int mat_index) {
           send_requests.emplace_back(world.isend(child_index,
FROM_MAIN_THREAD_TAG, mat[mat_index]));
           recv_requests.emplace_back(world.irecv(child_index,
FROM_CHILD_THREAD_TAG, average_values[mat_index]));
        });

        boost::mpi::wait_all(std::begin(send_requests),
std::end(send_requests));
        boost::mpi::wait_all(std::begin(recv_requests),
std::end(recv_requests));

        return average_values;
     }

  }

  void calc_c_array(const boost::mpi::communicator& world, const
size_t side_size) {
     if(world.rank() == 0) {
        const auto first_mat = generate_matrix(side_size);
        const auto second_mat = generate_matrix(side_size);
```

```
         std::cout << "Matrices:\n";
         print(first_mat); std::printf("\n");
         print(second_mat); std::printf("\n");

         auto first_averages = calc_average_vector::main_rank(world,
first_mat);
         auto second_averages = calc_average_vector::main_rank(world,
second_mat);

         std::cout << "Averages:\n";
         print(first_averages); std::printf("\n");
         print(second_averages); std::printf("\n");

         for(size_t i = 0; i < first_averages.size(); ++i) {
            first_averages[i] *= second_averages[i];
         }

         std::cout << "Result:\n";
         print(first_averages);
         std::printf("\n");
      } else {

         calc_average_vector::child_rank(world);
         calc_average_vector::child_rank(world);

      }
   }

   void main_logic(const size_t side_size) {

      const auto env = boost::mpi::environment();
      const auto world = boost::mpi::communicator();

      const auto needed_procs_num =
std::min(static_cast<size_t>(world.size()), side_size);
      const auto is_needed = world.rank() < needed_procs_num;
      const auto local_world = world.split(is_needed);

      if(not is_needed) {
         return;
      }

      calc_c_array(local_world, side_size);
   }

int main(int argc, char *argv[]) {
   main_logic(std::stoi(argv[1]));
   return 0;
}
```

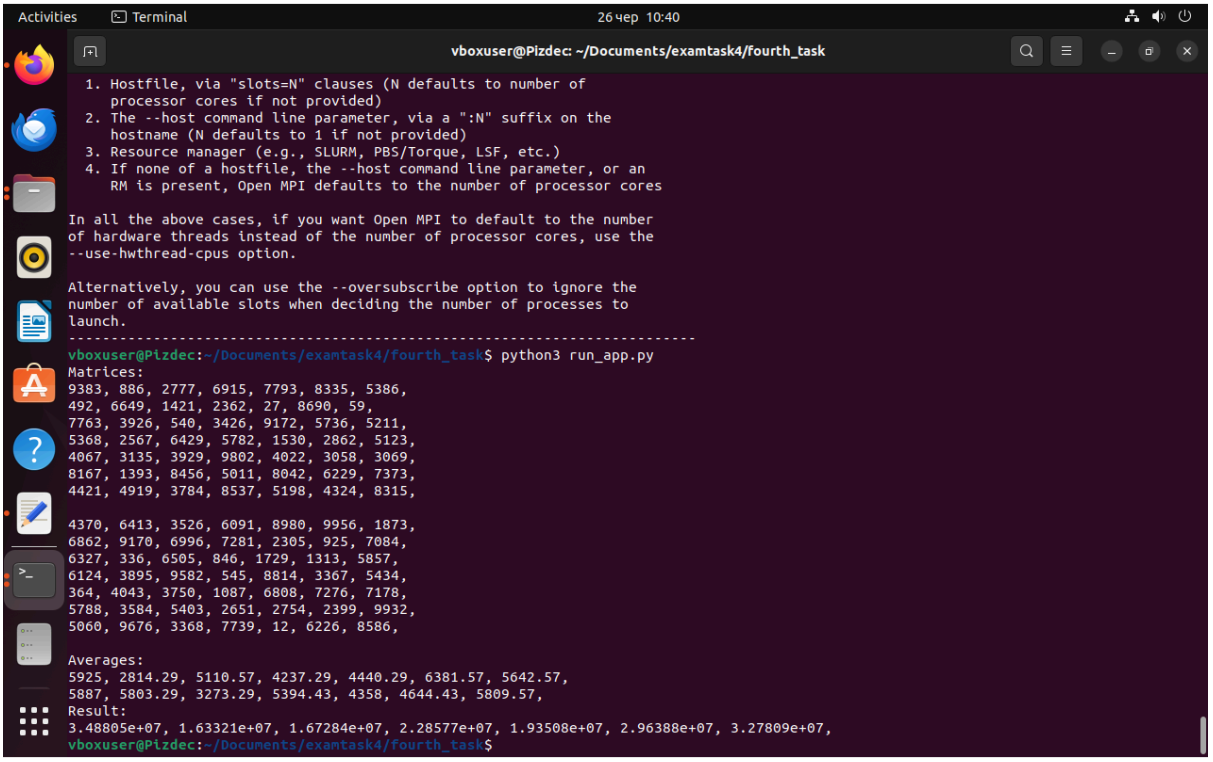
```
import os
```

```
procs_num = 5
size = 7

os.system(f'mpiexec -np {procs_num} ./exam_mpi {size}')
```

Matrices:
9383, 886, 2777, 6915, 7793, 8335, 5386,
492, 6649, 1421, 2362, 27, 8690, 59,
7763, 3926, 540, 3426, 9172, 5736, 5211,
5368, 2567, 6429, 5782, 1530, 2862, 5123,
4067, 3135, 3929, 9802, 4022, 3058, 3069,
8167, 1393, 8456, 5011, 8042, 6229, 7373,
4421, 4919, 3784, 8537, 5198, 4324, 8315,

4370, 6413, 3526, 6091, 8980, 9956, 1873,
6862, 9170, 6996, 7281, 2305, 925, 7084,
6327, 336, 6505, 846, 1729, 1313, 5857,
6124, 3895, 9582, 545, 8814, 3367, 5434,
364, 4043, 3750, 1087, 6808, 7276, 7178,
5788, 3584, 5403, 2651, 2754, 2399, 9932,
5060, 9676, 3368, 7739, 12, 6226, 8586,

Averages:
5925, 2814.29, 5110.57, 4237.29, 4440.29, 6381.57, 5642.57,
5887, 5803.29, 3273.29, 5394.43, 4358, 4644.43, 5809.57,
Result:
3.48805e+07, 1.63321e+07, 1.67284e+07, 2.28577e+07, 1.93508e+07, 2.96388e+07,
3.27809e+07,


Activities
Terminal
26 чер 10:40
vboxuser@Pizdec: ~/Documents/examtask4/fourth_task
1. Hostfile, via "slots=N" clauses (N defaults to number of
processor cores if not provided)
2. The --host command line parameter, via a ":N" suffix on the
hostname (N defaults to 1 if not provided)
3. Resource manager (e.g., SLURM, PBS/Torque, LSF, etc.)
4. If none of a hostfile, the --host command line parameter, or an
RM is present, Open MPI defaults to the number of processor cores
In all the above cases, if you want Open MPI to default to the number
of hardware threads instead of the number of processor cores, use the
--use-hwthread-cpus option.
Alternatively, you can use the --oversubscribe option to ignore the
number of available slots when deciding the number of processes to
launch.
vboxuser@Pizdec:~/Documents/examtask4/fourth_task$ python3 run_app.py
Matrices:
9383, 886, 2777, 6915, 7793, 8335, 5386,
492, 6649, 1421, 2362, 27, 8690, 59,
7763, 3926, 540, 3426, 9172, 5736, 5211,
5368, 2567, 6429, 5782, 1530, 2862, 5123,
4067, 3135, 3929, 9802, 4022, 3058, 3069,
8167, 1393, 8456, 5011, 8042, 6229, 7373,
4421, 4919, 3784, 8537, 5198, 4324, 8315,
4370, 6413, 3526, 6091, 8980, 9956, 1873,
6862, 9170, 6996, 7281, 2305, 925, 7084,
6327, 336, 6505, 846, 1729, 1313, 5857,
6124, 3895, 9582, 545, 8814, 3367, 5434,
364, 4043, 3750, 1087, 6808, 7276, 7178,
5788, 3584, 5403, 2651, 2754, 2399, 9932,
5060, 9676, 3368, 7739, 12, 6226, 8586,
Averages:
5925, 2814.29, 5110.57, 4237.29, 4440.29, 6381.57, 5642.57,
5887, 5803.29, 3273.29, 5394.43, 4358, 4644.43, 5809.57,
Result:
3.48805e+07, 1.63321e+07, 1.67284e+07, 2.28577e+07, 1.93508e+07, 2.96388e+07, 3.27809e+07,
vboxuser@Pizdec:~/Documents/examtask4/fourth_task$