Лабараторная Работа № 2
Метод Гаусса

Царик Виталий
3-й курс 2-я группа

n	Процессов	Время выполнения, с
10	2	0.005
10	4	0.006
10	8	0.008
100	2	0.055
100	4	0.042
100	8	0.046
1000	2	6.285
1000	4	4.798
1000	8	6.787

Таблица 1: Результаты для размерности n

Листинг 1: файл main.py

```
import sys

import numpy as np
from numpy import linalg as la
from mpi4py import MPI

from gauss_solver import GaussSolver
from utils import formatting, str_to_row, format_action, Timer,
                                              write_vector, read_input

MASTER = 0
inp = open('input.txt', 'r')
out = open('output.txt', 'w')

if __name__ == '__main__':
    comm = MPI.COMM_WORLD
    rows = []

    if comm.rank == MASTER:
        n, A, b = read_input(inp)

        totalTimer = Timer('master')

        comm.bcast(n, root=MASTER)

        step = n // comm.size
        master_count = step + n % comm.size

        rows = [A[i] + [b[i], i] for i in range(master_count)]

        cur = master_count
        for proc in range(1, comm.size):
            for i in range(step):
                comm.send(A[cur + i] + [b[cur + i], cur + i], dest=proc)
                                                     # send row with index
                                                      as last element
            cur += step

        print(formatting(comm.rank, format_action('receive', rows=rows)))

        g = GaussSolver(np.array(rows), comm, n)
        res = g.calc()

        print(formatting(comm.rank, format_action('result', rows=res)))

        for proc in range(1, comm.size):
            res += comm.recv(source=proc)

        res.sort(key=lambda row: row[-1])

        x = []
        for row in res:
            x.append(row[0])

        totalTimer.finish()

        print(formatting(comm.rank, format_action('result')))

        write_vector(x, out)

        print(formatting(comm.rank, format_action('answer', x=x)))

    else:
        procTimer = Timer('proc{}'.format(comm.rank))
        n = comm.bcast(None, root=MASTER)
        step = n // comm.size

        if not step:
            sys.exit()

        for i in range(step):
            rows.append(comm.recv(source=MASTER))

        print(formatting(comm.rank, format_action('receive', rows=rows)))

        g = GaussSolver(np.array(rows), comm, n)

        inversed = g.calc()

        print(formatting(comm.rank, format_action('result', rows=inversed)
                                                ))

        comm.send(inversed, dest=MASTER)
        procTimer.finish()
```

Листинг 2: файл gauss_solver.py

```
import numpy as np
from mpi4py import MPI


class GaussSolver:
    def __init__(self, data, comm, n):
        self.comm = comm

        self.n = n
        self.indexes = []

        self.A = np.zeros((self.n, self.n))
        self.b = np.zeros(self.n)

        for row in data:
            i = int(row[-1])
            self.indexes.append(i)
            self.A[i] = row[:-2]
            self.b[i] = row[-2]

        self.l = np.zeros(self.n)
        self.r = 0

    def calc(self):
        for i in range(self.n):  # forward elimination
            if i in self.indexes:
                self.b[i] /= self.A[i, i]
                self.A[i] /= self.A[i, i]
                self.__send(i)

            else:
                self._receive(i)

            self.comm.Barrier()

        for i in range(self.n - 1, -1, -1):  # back substitution
            if i in self.indexes:
                self.__send(i)
            else:
                self._receive(i)
```

```python

            self.comm.Barrier()

        return [[self.b[i], i] for i in self.indexes]

    def __eliminate(self, l, r, cur):
        for i in self.indexes:
            if i != cur:
                self.b[i] -= r * self.A[i, cur]
                self.A[i] -= l * self.A[i, cur]

    def __send(self, i):
        self.comm.Bcast([self.A[i], MPI.DOUBLE], root=self.comm.rank)
        self.comm.Bcast([self.b[i], MPI.DOUBLE], root=self.comm.rank)

        self.__eliminate(self.A[i], self.b[i], i)

    def _receive(self, i):
        self.comm.Bcast([self.l, MPI.DOUBLE], root=self.comm.rank)
        self.comm.Bcast([self.r, MPI.DOUBLE], root=self.comm.rank)

        self.__eliminate(self.l, self.r, i)
```

Листинг 3: файл utils.py

```python
import time


def str_to_row(s):
    return [int(x) for x in s.split()][:-1]


class Colors:
    HEADER = '\033[1;95m'
    OKBLUE = '\033[1;94m'
    OKGREEN = '\033[1;92m'
    WARNING = '\033[1;93m'
    FAIL = '\033[1;91m'
    CYAN = '\033[1;96m'
    ENDC = '\033[0m'


def formatting(rank, message):
    template = '${color} {name}{end_color}: {message}'

    return template.format(
        color=(Colors.FAIL if rank == 0 else Colors.OKGREEN),
        name=('master' if rank == 0 else 'proc{}'.format(rank)),
        end_color=Colors.ENDC,
        message=message
    )


def format_action(action, rows=None, x=None):
    template = '\t*{0}*\t'
    args = [action]
    if rows:
        template += '{1} rows'
        args.append(len(rows))
        if len(rows) < 4 and len(rows[0]) < 10:
            template += ': {2}'
            args.append([row[:-1] for row in rows])
    if x and len(x) < 10:
        template += '{1}x = {2}{3}'
        args += [Colors.OKBLUE, x, Colors.ENDC]
    return template.format(*args)


def read_input(inp):
    n = int(inp.readline())
    A = []
    b = []

    for line in inp:
        A.append(str_to_row(line))
        b.append(int(line[-2:-1]))

    return n, A, b

def write_vector(vec, out):
        out.write(('{:.3f}\t' * len(vec)).format(*vec))


class Timer:
    def __init__(self, message):
        self.message = message
        self.start = time.time()

    def finish(self):
        print("-" * 20 + "| {0}: {1:.3f} s |".format(self.message, (time.
                                                              time() - self.start)) + "-" *
                                                               20)
```