Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное
учреждение высшего образования
Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчет по лабораторной работе

«Многошаговая схема решения двумерных задач глобальной оптимизации. Распараллеливание по характеристикам»

Выполнил:
студент группы 381706-1
Резанцев С.А.

Проверил:
Доцент кафедры МОСТ,
кандидат технических наук
Сысоев А.В.

Нижний Новгород
2019

Содержание

1. Введение

Оптимизация — в математике, информатике и исследовании операций задача нахождения экстремума (минимума или максимума) целевой функции в некоторой области конечномерного векторного пространства, ограниченной набором линейных и/или нелинейных равенств и/или неравенств. Глобальные методы имеют дело с многоэкстремальными целевыми функциями. При глобальном поиске основной задачей является выявление тенденций глобального поведения целевой функции.

Стандартная математическая задача оптимизации формулируется таким образом. Среди элементов x, образующих множества X, найти такой элемент x^*, который доставляет минимальное значение $f(x^*)$ заданной функции f(x). Для того, чтобы корректно поставить задачу оптимизации, необходимо задать:

1. Допустимое множество — множество $X = \{\bar{x} | g_i(\bar{x}) \le 0,\ i = 1,\ ...,\ m\} \subset R^n$;
2. Целевую функцию — отображение $f : X \to R$;
3. Критерий поиска (max или min).

Тогда решить задачу означает одно из:

1. Показать, что $X = \varnothing$.
2. Показать, что целевая функция не ограничена снизу.
3. Найти $\bar{x}^* \in X : f(\bar{x}^*) = min\, f(\bar{x})$.
4. Если $\nexists \bar{x}^*$, то найти $inf\, f(\bar{x}^*)$.

Если минимизируемая функция не является выпуклой, то часто ограничиваются поиском локальных минимумов и максимумов.

2. Постановка задачи

В рамках данной лабораторной работы нужно реализовать параллельный алгоритм решения задачи глобальной оптимизации.

Необходимо реализовать следующие компоненты:

- Последовательный алгоритм решения задачи глобальной оптимизации
- Параллельный алгоритм метода решения задачи глобальной оптимизации
- Вспомогательные функции для проверки работоспособности и эффективности

Программное решение будет выглядеть следующим образом:

1. Модуль, содержащий реализацию библиотеки.
2. Набор автоматических тестов с использованием Google C++ Testing Framework.

3. Метод решения

Рассмотрим алгоритм решения задачи глобальной оптимизации.

Для решения задачи сперва необходимо задать функцию, область поиска, точность поиска ε и критерий надежности r.

Упорядочиваем поисковую информацию в порядке возрастания значений точек ранее выполненных итераций(точки первой итерации получаем исходя из условий поиска):

$$x_1 \le x_2 \le \ \dots \ \le x_k$$

Для каждого интервала (x_{i-1}, x_i), $1 \le i \le k$ вычислить характеристику R(i), найти среди них максимальную R(t) и в интервале с максимальной характеристикой вычислить следующую точку x_{k+1} в соответствии с некоторым правилом

$$x_{k+1} = s(t) \in (x_{t-1}, x_t).$$

В качестве текущей оценки данного решения принимаем наименьшее вычисленное значение минимизируемой функции

$$z_k = min\{z_i : 1 \le i \le k\}.$$

Если минимальное расстояние между двумя соседними точками меньше ε, останавливаемся, иначе продолжаем вычисление новых точек в области поиска.

Характеристика вычисляется по следующей формуле:

$$R(i) = m(x_i - x_{i-1}) + \frac{(z_i - z_{i-1})^2}{m(x_i - x_{i-1})} - 2 * (z_i - z_{i-1}),$$

где m - текущая оценка константы Липшица.

Правило вычисления точки:

$$x_{k+1} = \frac{(x_t + x_{t-1})}{2} - \frac{(z_t - z_{t-1})}{2m}.$$

4. Схема распараллеливания

Выполнение итераций метода осуществляется последовательно, следовательно, наиболее целесообразный подход состоит в распараллеливании вычислений, реализуемых в ходе выполнения отдельных итераций.

Самой ресурсоемкой операцией в алгоритме решения задачи глобальной оптимизации является вычисление текущей оценки данного решения z_i . Поэтому было принято решение разделить процессы на главный и вспомогательные. Главный вычисляет характеристики и находит x_i, вспомогательные процессы принимают x_i, находят z_i и отправляют z_i главному процессу.

При необходимости процессы обмениваются информацией, посредством выполнения операций MPI_Send и MPI_Recv.

5. Описание программной реализации

Программа включает в себя пять классов и одну структуру:

- class Point - класс, служащий для описания точек на плоскости, имеет конструктор и перегруженный оператор сравнения
- class GlobalPoint - наследник класса Point, служит для описания точек в трехмерном пространстве, имеет конструктор и перегруженный оператор сравнения
- class RPoint - служит для удобного хранения интервалов и характеристик, имеет конструктор и перегруженный оператор сравнения
- struct result - хранит результат вычислений алгоритма
- class OneDimensionalOpt - служит для решения задачи глобальной оптимизации на плоскости. Имеет следующие функции:
    - OneDimensionalOpt - конструктор
    - calculateR - вычисление характеристики
    - calculateM - вычисляет отношение интервалов по формуле
    $$M = \frac{z_i - z_{i-1}}{x_i - x_{i-1}}.$$
    - findMmax - находит максимальное отношение
    - findRmax - находит максимальную характеристику
    - calculatem - находит оценку константы Липшица
    - OrdinaryPrepearing - выполнение первой итерации алгоритма
    - newFunction - на вход принимает область поиска, функцию, точность поиска и критерий надежности, работа с новой задачей глобальной оптимизации
    - calculateNewPoint - вычисляет новую точку
    - calculate - решает задачу глобальной оптимизации
- class GlobalOptimazer - служит для решения задачи глобальной оптимизации в трехмерном пространстве. Имеет следующие функции:
    - GlobalOptimazer - конструктор
    - calculateR - вычисление характеристики
    - calculateM - вычисляет отношение интервалов по формуле
    $$M = \frac{z_i - z_{i-1}}{x_i - x_{i-1}}.$$
    - findMmax - находит максимальное отношение
    - calculatem - находит оценку константы Липшица
    - OrdinaryPrepearing - выполнение первой итерации алгоритма
    - OrdinaryCalculation - решает задачу глобальной оптимизации последовательно
    - calculateNewPoint - вычисляет новую точку

- globalCalculation - решает задачу глобальной оптимизации параллельно

6. Подтверждение корректности

Для подтверждения корректности в программе представлен набор тестов, разработанных с помощью использования Google C++ Testing Framework.

Тесты проверяют корректную работу последовательного и параллельного алгоритмов решения задач глобальной оптимизации заданных функции и сравнивают с известными точками минимума этих функций.

7. Результаты экспериментов

Вычислительные эксперименты проводились на ПК со следующей аппаратной конфигурацией:

- Операционная система: Windows 10 Home, версия 1903, сборка 18362.535
- Оперативная память: 8 ГБ
- Процессор: Intel Core i5-8250U 1.6GHz
- Версия Visual Studio: 2017

Функция, с которой проводим испытания:

$$f(x,y) = x^2 * y^2 + x^2 \div 2 + y^2 \div 2 + x * y + 1$$

Точность поиска $\varepsilon = 0,04$.

Таблица 1. Результаты экспериментов

Количество процессов	Время работы последовательного алгоритма, сек	Время работы параллельного алгоритма, сек	Ускорение
1	0,533	0,538	0,99
2	0,535	0,540	0,99
3	0,534	0,29	1,84
4	0,534	0,244	2.19

По данным экспериментов видно, что параллельный алгоритм работает быстрее, чем последовательный, если число процессов больше двух. Причем при увеличении числа процессов заметно, что ускорение возрастает. Также можно заметить, если процессов всего два, то параллельный алгоритм работает медленнее последовательного. Это объясняется тем, то вычислениями экстремума функции занимается только один процесс и, к тому же, тратится время на обмен данными между процессами.

8. Заключение

В результате лабораторной работы была разработана библиотека, реализующая алгоритм решения задачи глобальной оптимизации.

Реализация параллельной версии алгоритма была успешно достигнута, о чем говорят результаты экспериментов, проведенных в ходе работы. Они показывают, что параллельный случай работает быстрее, чем последовательный.

Для подтверждения корректности работы программы были разработаны и доведены до успешного выполнения тесты, созданные с использованием Google C++ Testing Framework.

9. Список литературы

1. В. П. Гергель, В. А. Гришин, А. В. Гергель “МНОГОМЕРНАЯ МНОГОЭКСТРЕМАЛЬНАЯ ОПТИМИЗАЦИЯ НА ОСНОВЕ АДАПТИВНОЙ МНОГОШАГОВОЙ РЕДУКЦИИ РАЗМЕРНОСТИ” Н. Новгород: Изд-во Нижегородского госуниверситета им. Н.И. Лобачевского, 2010.

2. Васильев А.Н. Самоучитель C++ с примерами и задачами. -СПб.: Наука и Техника, 2016. -480с.

3. Т. А. Павловская C/C++ Программирование на языке высокого уровня. - СПб.:Питер, 2011. - 461 с.

10. Приложение

global_opt.h

```
// Copyright 2019 Rezantsev Sergey
#ifndef MODULES_TASK_3_REZANTSEV_S_GLOBAL_OPT_GLOBAL_OPT_H_
#define MODULES_TASK_3_REZANTSEV_S_GLOBAL_OPT_GLOBAL_OPT_H_
#include <mpi.h>
#include <algorithm>
#include <cmath>
#include <iostream>
#include <set>

class Point {
 public:
  double x;
  double y;
  Point(double _x, double _y);
  friend bool operator<(const Point& l, const Point& r) { return l.x < r.x; }
};

class GlobalPoint : public Point {
 public:
  double z;
  GlobalPoint(double _x, double _y, double _z = 0) : Point(_x, _y), z(_z) {}
  friend bool operator<(const GlobalPoint& l, const GlobalPoint& r) {
    return l.x < r.x;
  }
};

class RPoint {
 public:
  double R;
  double x;
  double z;
  double prevX;
  double prevZ;
  RPoint(double _R, double _x, double _z, double _prevX, double _prevZ)
      : R(_R), x(_x), z(_z), prevX(_prevX), prevZ(_prevZ) {}
  friend bool operator<(const RPoint& l, const RPoint& r) { return l.R > r.R; }
};
```

```
struct result {
  double x;
  double y;
  double z;
};

class OneDimensionalOpt {
 public:
  result res;
  double a;
  double b;
  double currX;
  double (*function)(double _x, double _y);
  double eps;
  int Nmax;
  std::set<Point> set;
  std::set<Point>::iterator lPoint;
  std::set<Point>::iterator rPoint;
  std::set<Point>::iterator rPointRmax;
  std::set<Point>::iterator lPointRmax;
  double r_p;
  double Rmax;
  double smallestInterval;
  double m;
  double Mmax;
  OneDimensionalOpt(double _a, double _b, double _currx,
                    double (*f)(double _x, double _y), double _eps = 0.001,
                    int _Nmax = 500, double _r_p = 2.0);
  double calculateR();
  double calculateM();
  double calculateNewPoint();
  void newFunction(double _a, double _b, double _currx,
                   double (*f)(double _x, double _y));
  void calculatem();
  void OrdinaryPrepearing();
  void calculate();
  void findMmax();
  void findRmax();
};

class GlobalOptimazer {
 public:
```

```
  result res;
  double a;
  double b;
  double c;
  double d;
  double (*function)(double _x, double _y);
  double eps;
  int Nmax;
  double r_p;
  double Rmax;
  double smallestInterval;
  double m;
  double Mmax;
  int size;
  int rank;
  std::set<GlobalPoint> set;
  std::set<GlobalPoint>::iterator lP;
  std::set<GlobalPoint>::iterator rP;
  std::set<RPoint> setR;
  std::set<RPoint>::iterator RP;
  result temp;
  GlobalOptimazer(double _a, double _b, double _c, double _d,
                  double (*f)(double _x, double _y), double _eps = 0.05,
                  int _Nmax = 500, double _r_p = 2.0)
      : a(_a),
        b(_b),
        c(_c),
        d(_d),
        function(f),
        eps(_eps),
        Nmax(_Nmax),
        r_p(_r_p) {}
  void globalCalculation();
  double calculateR();
  double calculateM();
  double calculateNewPoint();
  void calculatem();
  void findMmax();
  void OrdinaryCalculation();
  void setRPoints();
};
```

```
#endif  // MODULES_TASK_3_REZANTSEV_S_GLOBAL_OPT_GLOBAL_OPT_H_
```

global_opt.cpp

```
// Copyright 2019 Rezantsev Sergey

#include "../../../modules/task_3/rezantsev_s_global_opt/global_opt.h"

Point::Point(double _x, double _y) : x(_x), y(_y) {}

OneDimensionalOpt::OneDimensionalOpt(double _a, double _b, double _currX,
                                     double (*f)(double _x, double _y),
                                     double _eps, int _Nmax, double _r_p)
    : a(_a),
      b(_b),
      currX(_currX),
      function(f),
      eps(_eps),
      Nmax(_Nmax),
      r_p(_r_p) {}

double OneDimensionalOpt::calculateM() {
  return std::abs((rPoint->y - lPoint->y) / (rPoint->x - lPoint->x));
}

double OneDimensionalOpt::calculateR() {
  return m * (rPoint->x - lPoint->x) +
         (std::pow(rPoint->y - lPoint->y, 2) / (m * (rPoint->x - lPoint->x))) -
         2 * (rPoint->y - lPoint->y);
}

void OneDimensionalOpt::OrdinaryPrepearing() {
  res.x = currX;
  res.y = a;
  res.z = function(currX, a);
  set.insert(Point(a, res.z));
  double tmp = function(currX, b);
  set.insert(Point(b, tmp));
  if (res.z > tmp) {
    res.y = b;
    res.z = tmp;
  }
  lPoint = set.begin();
  rPoint = set.begin();
  rPoint++;
  Rmax = calculateR();
```

```
  smallestInterval = std::abs(a - b);
}

void OneDimensionalOpt::findMmax() {
  Mmax = -1;
  lPoint = set.begin();
  rPoint = set.begin();
  rPoint++;
  while (rPoint != set.end()) {
    double M = calculateM();
    if (M > Mmax) {
      Mmax = M;
    }
    rPoint++;
    lPoint++;
  }
}

void OneDimensionalOpt::calculatem() {
  if (Mmax > 0) {
    m = Mmax * r_p;
  } else if (Mmax == 0) {
    m = 1.0;
  }
}

void OneDimensionalOpt::newFunction(double _a, double _b, double _currx,
                                    double (*f)(double _x, double _y)) {
  a = _a;
  b = _b;
  currX = _currx;
  function = f;
}

void OneDimensionalOpt::findRmax() {
  Rmax = -200000000;
  lPoint = set.begin();
  rPoint = set.begin();
  rPointRmax = set.begin();
  lPointRmax = set.begin();
  rPoint++;
  while (rPoint != set.end()) {
    double R = calculateR();
    double s = rPoint->x - lPoint->x;
    if (R > Rmax) {
      Rmax = R;
```

```
      rPointRmax = rPoint;
      lPointRmax = lPoint;
    }
    if (s < smallestInterval) {
      smallestInterval = s;
    }
    rPoint++;
    lPoint++;
  }
}

double OneDimensionalOpt::calculateNewPoint() {
  return (0.5) * (rPointRmax->x + lPointRmax->x) -
         ((rPointRmax->y - lPointRmax->y) / (2.0 * m));
}

void OneDimensionalOpt::calculate() {
  OrdinaryPrepearing();
  int n = 2;
  while (smallestInterval > eps) {
    findMmax();
    calculatem();
    findRmax();
    n++;
    double newY = calculateNewPoint();
    double tmp = function(currX, newY);
    set.insert(Point(newY, tmp));
    if (res.z > tmp) {
      res.y = newY;
      res.z = tmp;
    }
  }
  set.clear();
}

double GlobalOptimazer::calculateR() {
  return m * (rP->x - lP->x) +
         (std::pow((rP->z - lP->z), 2) / (m * (rP->x - lP->x))) -
         2 * (rP->z - lP->z);
}

double GlobalOptimazer::calculateM() {
  return std::abs((rP->z - lP->z) / (rP->x - lP->x));
}

void GlobalOptimazer::calculatem() {
```

```
  if (Mmax > 0) {
    m = r_p * Mmax;
  } else {
    m = 1;
  }
}

void GlobalOptimazer::findMmax() {
  Mmax = -1;
  rP = set.begin();
  rP++;
  lP = set.begin();
  while (rP != set.end()) {
    double currM = calculateM();
    if (currM > Mmax) Mmax = currM;
    rP++;
    lP++;
  }
}

void GlobalOptimazer::setRPoints() {
  rP = set.begin();
  rP++;
  lP = set.begin();
  while (rP != set.end()) {
    double R = calculateR();
    setR.insert(RPoint(R, rP->x, rP->z, lP->x, lP->z));
    rP++;
    lP++;
  }
}

double GlobalOptimazer::calculateNewPoint() {
  return (0.5) * (RP->x + RP->prevX) - ((RP->z - RP->prevZ) / (2 * m));
}

void GlobalOptimazer::OrdinaryCalculation() {
  res = {0, 0, 0};
  result ress;
  OneDimensionalOpt opt(c, d, a, function, eps, Nmax);
  opt.calculate();
  ress = opt.res;
  set.insert(GlobalPoint(ress.x, ress.y, ress.z));
  res = ress;
  opt.newFunction(c, d, b, function);
  opt.calculate();
```

```
  ress = opt.res;
  set.insert(GlobalPoint(ress.x, ress.y, ress.z));
  if (ress.z < res.z) res = ress;
  do {
    setR.clear();
    findMmax();
    calculatem();
    setRPoints();
    RP = setR.begin();
    double newX = calculateNewPoint();
    opt.newFunction(c, d, newX, function);
    opt.calculate();
    ress = opt.res;
    set.insert(GlobalPoint(ress.x, ress.y, ress.z));
    if (ress.z < res.z) res = ress;
  } while (RP->x - RP->prevX >= eps);
}

void GlobalOptimazer::globalCalculation() {
  MPI_Comm_size(MPI_COMM_WORLD, &size);
  if (size < 2) {
    OrdinaryCalculation();
  } else {
    MPI_Status status;
    res = {0, 0, 0};
    result ress;
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
    if (rank == 0) {
      int k;
      if (std::abs(a - b) <= 0.0001 || std::abs(c - d) <= 0.0001) {
        k = size + 1;
      } else {
        k = size;
      }
      double segmentLen = (b - a) / (k - 1);
      for (int i = 0; i < size - 1; ++i) {
        double Xf = a + i * segmentLen;
        MPI_Send(&Xf, 1, MPI_DOUBLE, i + 1, 1, MPI_COMM_WORLD);
      }
      OneDimensionalOpt tmp(c, d, b, function, eps);
      tmp.calculate();
      ress = tmp.res;
      set.insert(GlobalPoint(ress.x, ress.y, ress.z));
      res = ress;
      if (k != size) {
        tmp.newFunction(c, d, a + segmentLen * size, function);
```

```
      tmp.calculate();
      ress = tmp.res;
      set.insert(GlobalPoint(ress.x, ress.y, ress.z));
      if (ress.z < res.z) res = ress;
    }
    for (int i = 0; i < size - 1; ++i) {
      MPI_Recv(&ress, 3, MPI_DOUBLE, MPI_ANY_SOURCE, 1,
MPI_COMM_WORLD,
          &status);
      if (ress.z < res.z) res = ress;
      set.insert(GlobalPoint(ress.x, ress.y, ress.z));
    }
    bool f = true;
    while (f) {
      setR.clear();
      findMmax();
      calculatem();
      setRPoints();
      RP = setR.begin();
      for (int i = 0; i < size - 1; ++i) {
        double newX = calculateNewPoint();
        MPI_Send(&newX, 1, MPI_DOUBLE, i + 1, 1, MPI_COMM_WORLD);
        if (RP->x - RP->prevX <= eps) {
          f = false;
        }
        RP++;
      }
      for (int i = 0; i < size - 1; ++i) {
        MPI_Recv(&ress, 3, MPI_DOUBLE, MPI_ANY_SOURCE, 1,
MPI_COMM_WORLD,
            &status);
        if (ress.z < res.z) res = ress;
        set.insert(GlobalPoint(ress.x, ress.y, ress.z));
      }
    }
    for (int i = 0; i < size - 1; ++i) {
      double v = eps * 0.001;
      MPI_Send(&v, 1, MPI_DOUBLE, i + 1, 1, MPI_COMM_WORLD);
    }
  } else {
    double npoint = -1;
    do {
      MPI_Recv(&npoint, 1, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD, &status);
      if (npoint == eps * 0.001) {
        break;
      }
```

```
        OneDimensionalOpt tmp(c, d, npoint, function, eps, Nmax);
        tmp.calculate();
        ress = tmp.res;
        MPI_Send(&ress, 3, MPI_DOUBLE, 0, 1, MPI_COMM_WORLD);
      } while (true);
    }
  }
}
```

main.cpp

```
// Copyright 2019 Rezantsev Sergey
#include <gtest/gtest.h>
#include <gtest-mpi-listener.hpp>
#include "../../../modules/task_3/rezantsev_s_global_opt/global_opt.h"

double f1(double x, double y) { return std::pow(x, 2) + std::pow(y - 1, 2); }
double f2(double x, double y) { return exp(2 * x) * (x + y * y - 2 * y); }
double f3(double x, double y) {
  return x * x * y * y + x * x / 2 + y * y / 2 + x * y + 1;
}
double f4(double x, double y) {
  return std::pow(x, 2) + x * y + std::pow(y, 2) - 4 * std::log(x) -
         10 * std::log(y);
}

TEST(global_opt, test_f1) {
  int rank;
  result r;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  double (*fptr)(double, double) = f1;
  GlobalOptimazer glob(-5, 5, -5, 5, fptr);
  glob.globalCalculation();
  if (rank == 0) {
    r = glob.res;
    result res = {std::abs(r.x), std::abs(r.y - 1.0), 0};
    ASSERT_TRUE(res.x <= 0.1);
    ASSERT_TRUE(res.y <= 0.1);
  }
}

TEST(global_opt, test_f1_on_oy) {
  int rank;
  result r;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  double (*fptr)(double, double) = f1;
  GlobalOptimazer glob(0, 5, 0, 5, fptr);
```

```
  glob.globalCalculation();
  if (rank == 0) {
    r = glob.res;
    result res = {std::abs(r.x), std::abs(r.y - 1.0), 0};
    ASSERT_TRUE(res.x <= 0.1);
    ASSERT_TRUE(res.y <= 0.1);
  }
}

TEST(global_opt, test_f1_on_ox) {
  int rank;
  result r;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  double (*fptr)(double, double) = f1;
  GlobalOptimazer glob(-5, 1, 5, 1, fptr);
  glob.globalCalculation();
  if (rank == 0) {
    r = glob.res;
    result res = {std::abs(r.x), std::abs(r.y - 1.0), 0};
    ASSERT_TRUE(res.x <= 0.1);
    ASSERT_TRUE(res.y <= 0.1);
  }
}

TEST(global_opt, test_f2) {
  int rank;
  result r;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  double (*fptr)(double, double) = f2;
  GlobalOptimazer glob(-5, 5, -5, 5, fptr);
  glob.globalCalculation();
  if (rank == 0) {
    r = glob.res;
    result res = {std::abs(r.x - 0.5), std::abs(r.y - 1.0), 0};
    ASSERT_TRUE(res.x <= 0.1);
    ASSERT_TRUE(res.y <= 0.1);
  }
}

TEST(global_opt, test_f3) {
  int rank;
  result r;
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  double (*fptr)(double, double) = f3;
  GlobalOptimazer glob(-5, 5, -5, 5, fptr, 0.04);
  double startPar = MPI_Wtime();
```

```
  glob.globalCalculation();
  double endPar = MPI_Wtime();
  if (rank == 0) {
    r = glob.res;
    GlobalOptimazer g(-5, 5, -5, 5, fptr, 0.04);
     double startSeq = MPI_Wtime();
     g.OrdinaryCalculation();
     double endSeq = MPI_Wtime();
     std::cout << "Time seq: " << endSeq - startSeq << std::endl;
     std::cout << "Time par: " << endPar - startPar << std::endl;
    result res = {std::abs(r.x), std::abs(r.y), 0};
    ASSERT_TRUE(res.x <= 0.1);
    ASSERT_TRUE(res.y <= 0.1);
  }
}

int main(int argc, char** argv) {
  ::testing::InitGoogleTest(&argc, argv);
  MPI_Init(&argc, &argv);

  ::testing::AddGlobalTestEnvironment(new GTestMPIListener::MPIEnvironment);
  ::testing::TestEventListeners& listeners =
      ::testing::UnitTest::GetInstance()->listeners();

  listeners.Release(listeners.default_result_printer());
  listeners.Release(listeners.default_xml_generator());

  listeners.Append(new GTestMPIListener::MPIMinimalistPrinter);
  return RUN_ALL_TESTS();
}
```