Московский государственный университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Николайчук Артём Константинович

Отчёт о практической работе по курсу суперкомпьютерное моделирование и технологии

Москва, 2024

Содержание

1 Постановка задачи **3**

2 Решение задачи **5**

3 Распараллеливание при помощи OpenMp-нитей **7**

4 Распараллеливание при помощи MPI **9**

5 Распараллеливание при помощи совмещения подходов(OpenMP + MPI) **9**

6 Распараллеливание при помощи MPI и GPU **10**

7 Результаты **11**

8 Анализ результатов **12**

1 Постановка задачи

Необходимо найти численное решение уравнения $-\triangle u = 1, (x, y) \in D$

Множество D имеет описывается ограничениями и имеет вид на рисунке 1

$$\begin{cases} 1 < x < 3, \\ x^2 - 4 \cdot y^2 > 1 \end{cases}$$

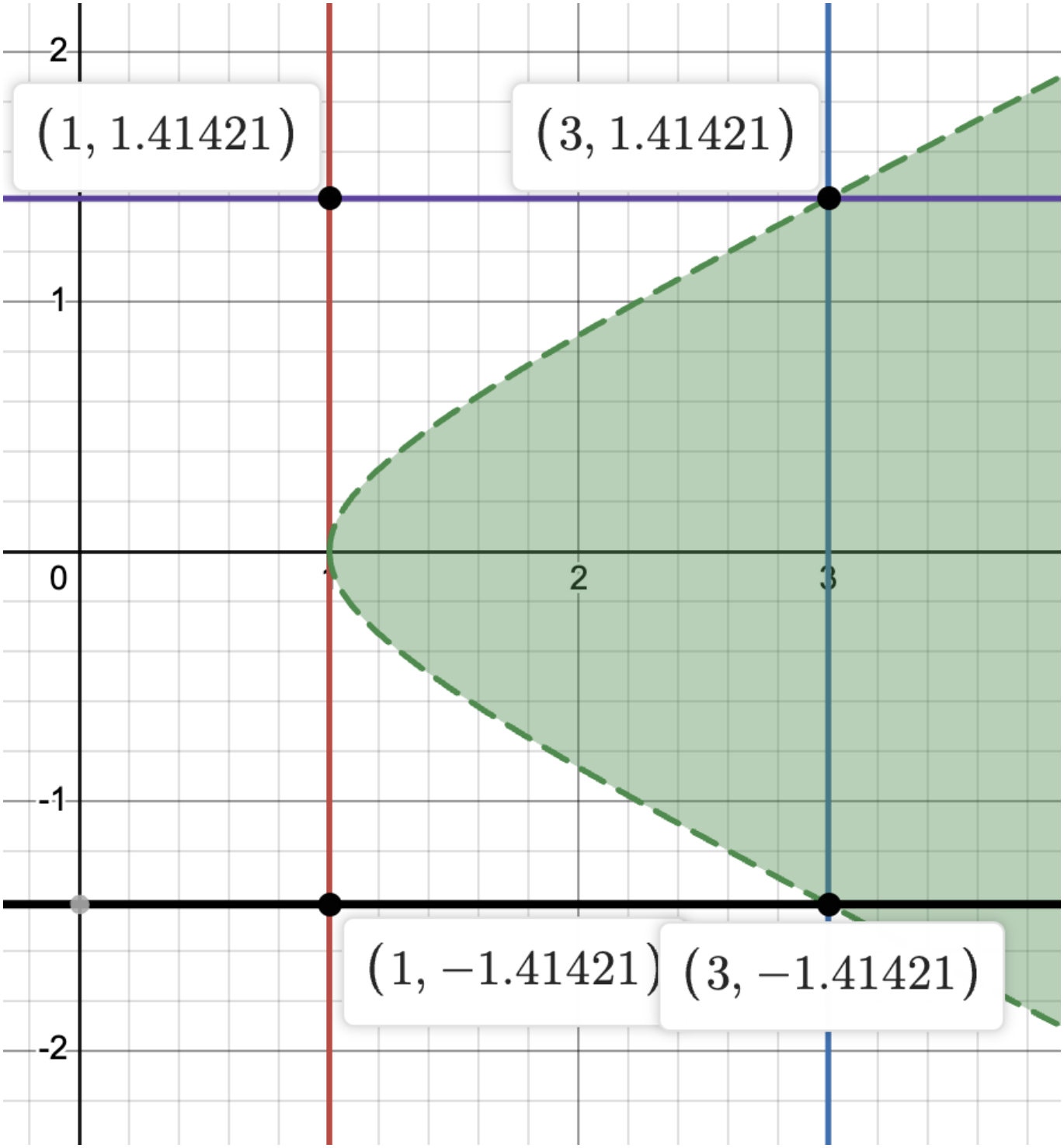

Рисунок 1: Область D

Область D является подобластью прямоугольника с вершинами

$A(1, -\sqrt{2}), B(1, \sqrt{2}), C(3, \sqrt{2}), D(3, -\sqrt{2})$. Этот прямоугольник будет использован для построения сетки и поиска решения.

2 Решение задачи

Первым шагом решения является вычисление на сетке вспомогательных матриц A, B, F. Значения матриц определяются формулами

$$a_{ij} = \frac{1}{h_2} \int_{y_{j-1/2}}^{y_{j+1/2}} k(x_{i-1/2}, t)\, dt$$

$$b_{ij} = \frac{1}{h_1} \int_{x_{i-1/2}}^{x_{i+1/2}} k(t, y_{j-1/2})\, dt$$

$$F_{ij} = \frac{1}{h_1 h_2} \iint_{\Pi_{ij}} F(x, y)\, dx\, dy, \quad \Pi_{ij} = \{(x, y) : x_{i-1/2} \le x \le x_{i+1/2},\, y_{j-1/2} \le y \le y_{j+1/2}\}$$

Для a и b интегралы можно вычислить аналитически по формуле

$$a_{ij} = h_2^{-1} l_{ij} + \left(1 - h_2^{-1} l_{ij}\right) / \varepsilon$$

Где l_{ij} это длина части отрезка $[x_{i-1/2}, x_{i+1/2}]$, которая попадает в область D.

Рассмотрим случаи для вычисления коэффициента a_{ij}.

- Вертикальный отрезок полностью в D.
- Вертикальный отрезок полностью не в D.
- Вертикальный отрезок имеет одно пересечение с гиперболой.
- Вертикальный отрезок имеет два пересечения с гиперболой.

Во всех случаях можно воспользоваться формулой выше.

Рассмотрим случаи для вычисления коэффициента b_{ij}.

- Горизонтальный отрезок полностью в D.
- Горизонтальный отрезок полностью не в D.
- Горизонтальный отрезок имеет одно пересечение с гиперболой и не пересекает границы $x = 1$ и $x = 3$.
- Горизонтальный отрезок имеет одно пересечение с гиперболой и пересекает границу $x = 1$.

- Горизонтальный отрезок имеет одно пересечение с гиперболой и пересекает границу $x = 3$.

Во всех случаях пользуемся формулой выше.
Для расчёта f_{ij} используется формула

$$(h_1 h_2)^{-1} S_{ij} f(x_i^*, y_j^*),$$

В этой формуле S_{ij} это площадь пересечения прямоугольника Π_{ij} и области D, $f(x_i^*, y_j^*)$ значение функции в произвольной точке из этого пересечения. При этом можно отметить, что все Π_{ij} можно обрезать по x до отрезка $[1, 3]$, потому что части вне этого отрезка имеют нулевую площадь пересечения с областью D. Все варианты рассмотрены на рисунках 2 и 3. При вычислении полезно разбить прямоугольник на два $y > 0$ и $y < 0$. В некоторых случаях удобно счиатать площадь, как сумму площади трапеции и прямоугольника.

Далее переходим к методу скорейшего спуска. Метод является одношаговым. Итерация $w^{(k+1)}$ вычисляется по итерации $w^{(k)}$ согласно равенствам:

$$w_{ij}^{(k+1)} = w_{ij}^{(k)} - \tau_{k+1} r_{ij}^{(k)},$$

где невязка $r^{(k)} = Aw^{(k)} - B$, итерационный параметр

$$\tau_{k+1} = \frac{(r^{(k)}, r^{(k)})}{(Ar^{(k)}, r^{(k)})}.$$

Начальное приближение - нулевая матрица. Критерий остановки δ выбирается в зависимости от сетки и времени, которое тратится на сходимость. $\delta = 3e^{-7}$ для всех матриц, кроме $N = 180, M = 160$, для них $\delta = 26e^{-9}$

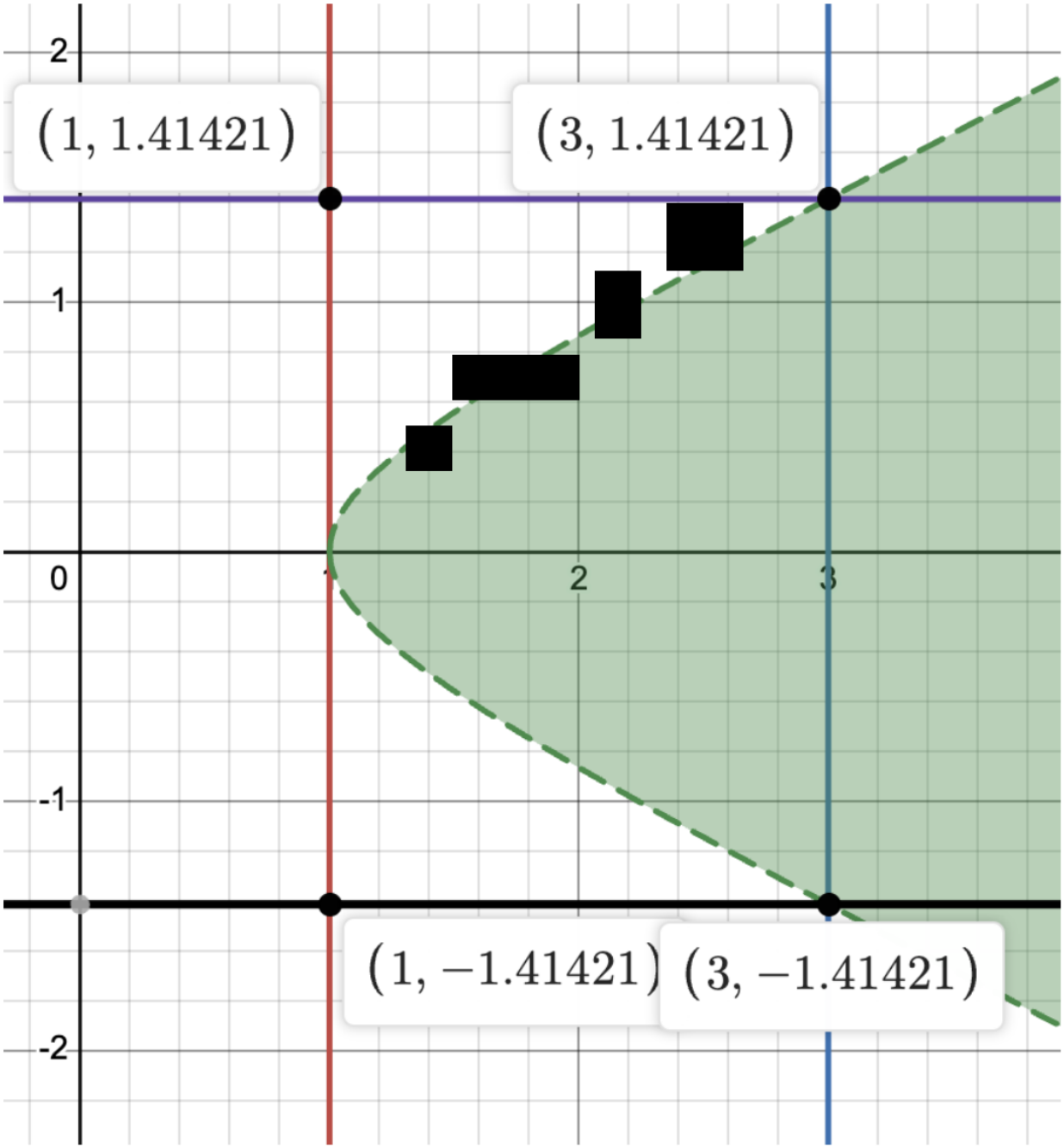

Рисунок 2: Варианты пересечений

3 Распараллеливание при помощи OpenMp-нитей

Основная часть программы, которая занимает большую часть времени - поиск решения матричного уравнения. В нём используются тяжёлые операции трёх типов - применение оператора, вычитание матрицы и вычисление скалярного произведения. Первые две можно ускорить, применив директиву

```
#pragma omp parallel for collapse(2)
```

Они вычисляются при помощи двумерного цикла, все операции внутри независимы.

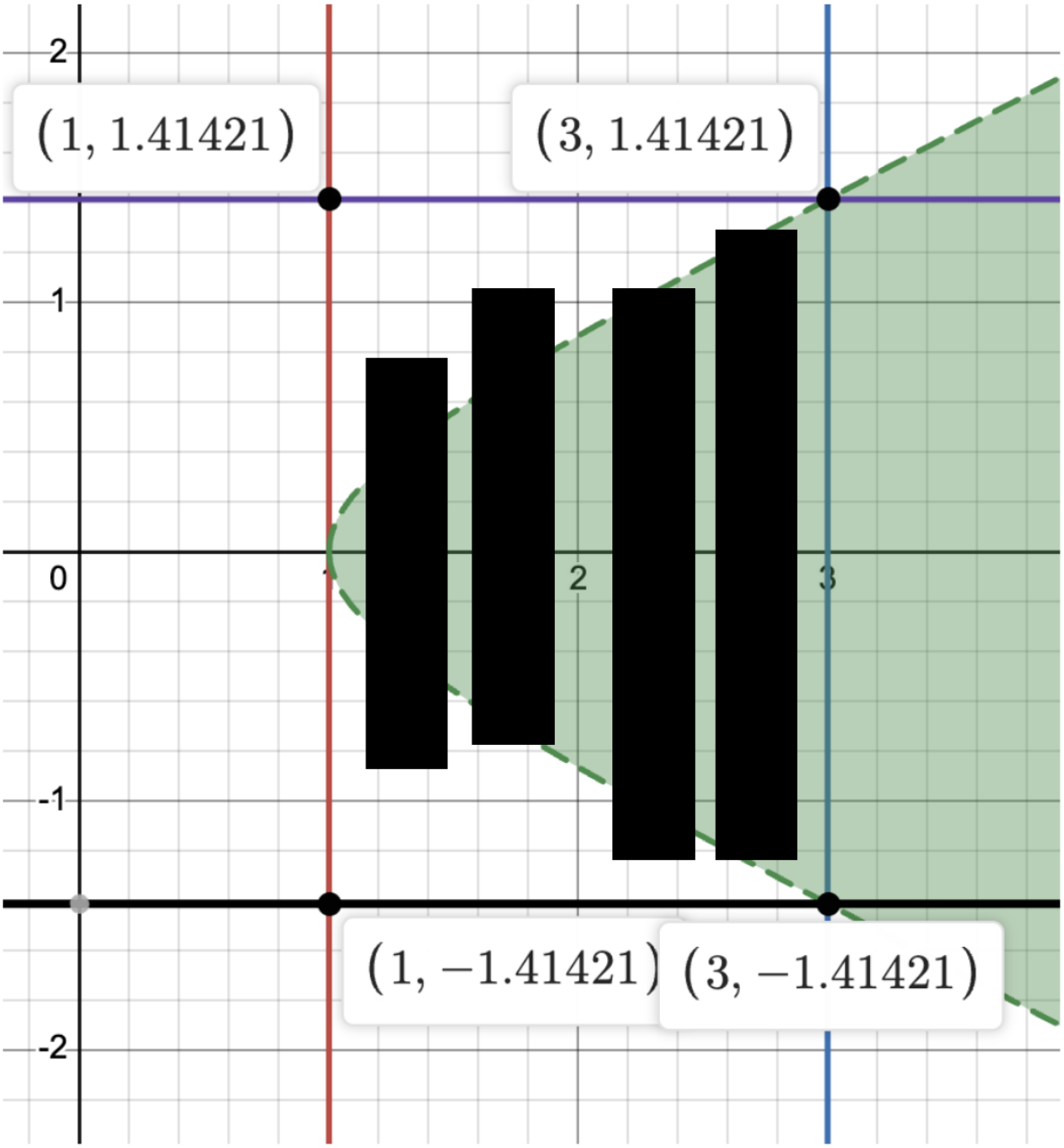

Рисунок 3: Варианты пересечений

Для ускорения вычисления скалярного произведения необходимо применить директиву, потому что вычисление использует запись в общую переменную *result*

```
#pragma omp parallel for reduction (+: result)
```

4 Распараллеливание при помощи MPI

Каждый из процессов должен работать над своей областью матрицы. Для этого необходимо равномерно разбить матрицу между процессами. Для получения разбиения матрицы на прямоугольники воспользуемся функцией

`MPI_Dims_create`

Она вернёт размерность сетки, на которую надо разбить матрицу, например для 4 процессов будет сетка 2x2, для шести 3x2.

Даллее необходимо разделить матрицу размера (N+1)x(M+1) на сетку между процессами.

Алгоритм на примере стороны - dim[0] - нужное количество элементов сетки. Тогда в каждом элементе будет хотя бы $N/dim[0]$ элементов матрицы. При этом среди первых прямоугольников необходимо распределить $N\%dim[0]$ элементов.

Формула - $a * (N/dim[0] + 1) + b * (N/dim[0]) = N$, где a - количество прямоугольников со стороной $N/dim[0] + 1$ и b - со строной $N/dim[0]$.

Таким образом можно разбить матрицу на равномерную сетку между процессами.

При вычислении действия оператора A на матрицу необходимо каждому из процессов использовать части матрицы, которые принадлежат другому процессу. Для этого эти границы подматриц передаются между процессами на каждой итерации.

Для этого используются функции MPI_Isend - для отправки и MPI_Irecv для чтения результата.

Для подсчёта скалярного произведения используется функция, в которую передаются предварительно подсчитанные скалярные произведения на подматрицах $MPI\_Allreduce$

5 Распараллеливание при помощи совмещения подходов(OpenMP + MPI)

Комбинируем оба метода - добавляем прагмы в код программы для MPI.

6 Распараллеливание при помощи MPI и GPU

Для ускорения на GPU был выбран подход использования openacc директив. Описание особенностей работы с openacc-директивами и способ применения их для решения текущей задачи:

- Изначально инициализируются массивы с матрицами из условия, матрицей-решением, матрица для ошибки и набо вспомогательных массивов для пересылки данных. Они копируются на устройство директивой #pragma acc data copy. Далее начинается основной цикл.
- Вычисление матрицы r - действия оператора A на текущее решение w. Это вычисление двумерной матрицы производится полностью на устройстве при помощи директивы #pragma acc parallel loop.
- Теперь необходимо отправить копию данных на границе r другим mpi процессам. Для этого границы загружаются из устройства на хост директивой #pragma acc update self.
- Асинхронно граница отправляется другим процессам.
- В этот момент на устройстве запускается вычисление скалярного произведения (r, r) при помощи директивы #pragma acc parallel loop reduction(+:result)
- Теперь необходимо передать на устройство границы матрицы r от других процессов. - #pragma acc update device
- На устройстве вычисляется действие A на r. #pragma acc parallel loop
- Вычисляется скалярное произведение Ar на r - pragma acc parallel loop reduction(+:result)
- На хосте при помощи обмена с другими процессами вычисляется ошибка tau
- На устройстве обновляется текущее решение обновляется директивой #pragma acc parallel loop

Таким образом в алгоритме используется две пересылки данных между устройством и хостом - размер данных линеен. Используется две пересылки между mpi процессами - одна линейна по данным, вторая для вычисления скалярного произведения - константного размера. Причём mpi пересылка выполняется асинхронно.

7 Результаты

Файл *Readme.md* содержит описание действия для повторения экспериментов. В папке results можно найти артефакты запусков на полюсе.

Методы оценивались на сетке 5000×5000, что эквивалетно примерно 200мб памяти.

Далее представлены результаты экспериментов в таблицах 1 и 2

В таблице 3 собраны вместе все времена работы для наглядности.

Таблица 1: Таблица с результатами времени выполнения частей программы(1)

Операция	**Последовательная**	**1mpi, 1GPU**	**2mpi, 2GPU**
Вычисление AW	329.64	1.66	0.799
Перенос r с gpu	0.000015	0.011649	0.027
Вычисление (r,r)	5.87	0.164762	0.115
Асинхронная отправка r	0.000285	0.000037	0.074
Перенос r на gpu	0.000013	0.0102	0.0089
Вычисление Ar	263.74	1.52	0.758
Вычисление (Ar,r)	6.13	0.2589	0.1537
Вычисление tau	0.002	0.0019	0.038
Вычисление new_w	11.34	0.3201	0.170
Общее время решения	616.750	4.96	2.629

Таблица 2: Таблица с результатами времени выполнения частей программы(2)

Операция	**10 mpi**	**20 mpi**	**160openmp**	**2mpi,80openmp**
Вычисление AW	33.36	18.11	13.30	9.947
Перенос r с gpu	-	-	-	-
Вычисление (r,r)	0.59	0.332	1.92	0.413
Асинхронная отправка r	0.85	1.59	-	3.111
Перенос r на gpu	-	-	-	-
Вычисление Ar	26.611	14.53	10.11	8.65
Вычисление (Ar,r)	0.68	0.47	2.50	1.15
Вычисление tau	19.77	5.080	0.0044	2.683
Вычисление new_w	1.176	0.898	1.796	0.931
Общее время решения	83.05	41.034	29.659	26.91

Таблица 3: Таблица с результатами сравнения методов ускорения

тип програмы	**Время работы метода**
Последовательная	616.750
10 mpi	83.05
20 mpi	41.034
160omp	29.659
2mpi,80openmp	26.91
1mpi, 1GPU	4.96
2mpi, 2GPU	2.629

8 Анализ результатов

Корректность программ, использующих технологии распараллеливания, определялась по сравнению с последовательной программой - при одинаковых начальных данных совпадало количество итерация алгоритма и финальная ошибка.

На основании запусков можно сделать следующие выводы:

- Mpi+GPU работает гораздо быстрее остальных методов. Основное ускорение получается при сложных вычислениях в больших матрицах, в этом случае наиболее эффективно используются ресурсы GPU. Программа с двумя mpi-процессами и GPU ожидаемо в два раза быстрее, чем с одним. Время на общение между процессами незначительно.