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

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

Множество D имеет описывается ограничениями и имеет вид на рисунке 1

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

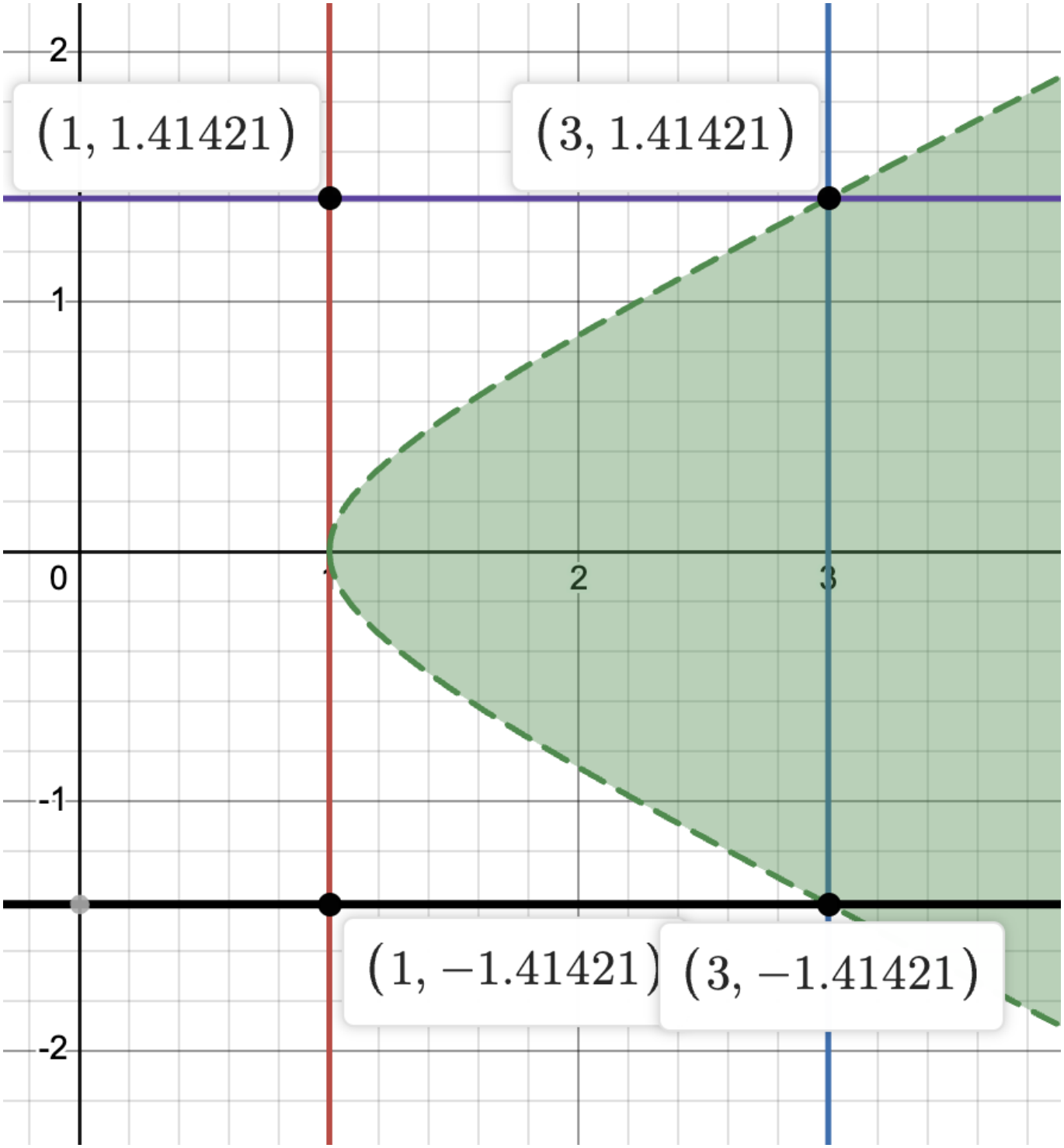


Рисунок 1: Область D

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 $\delta = 26e^{-9}$

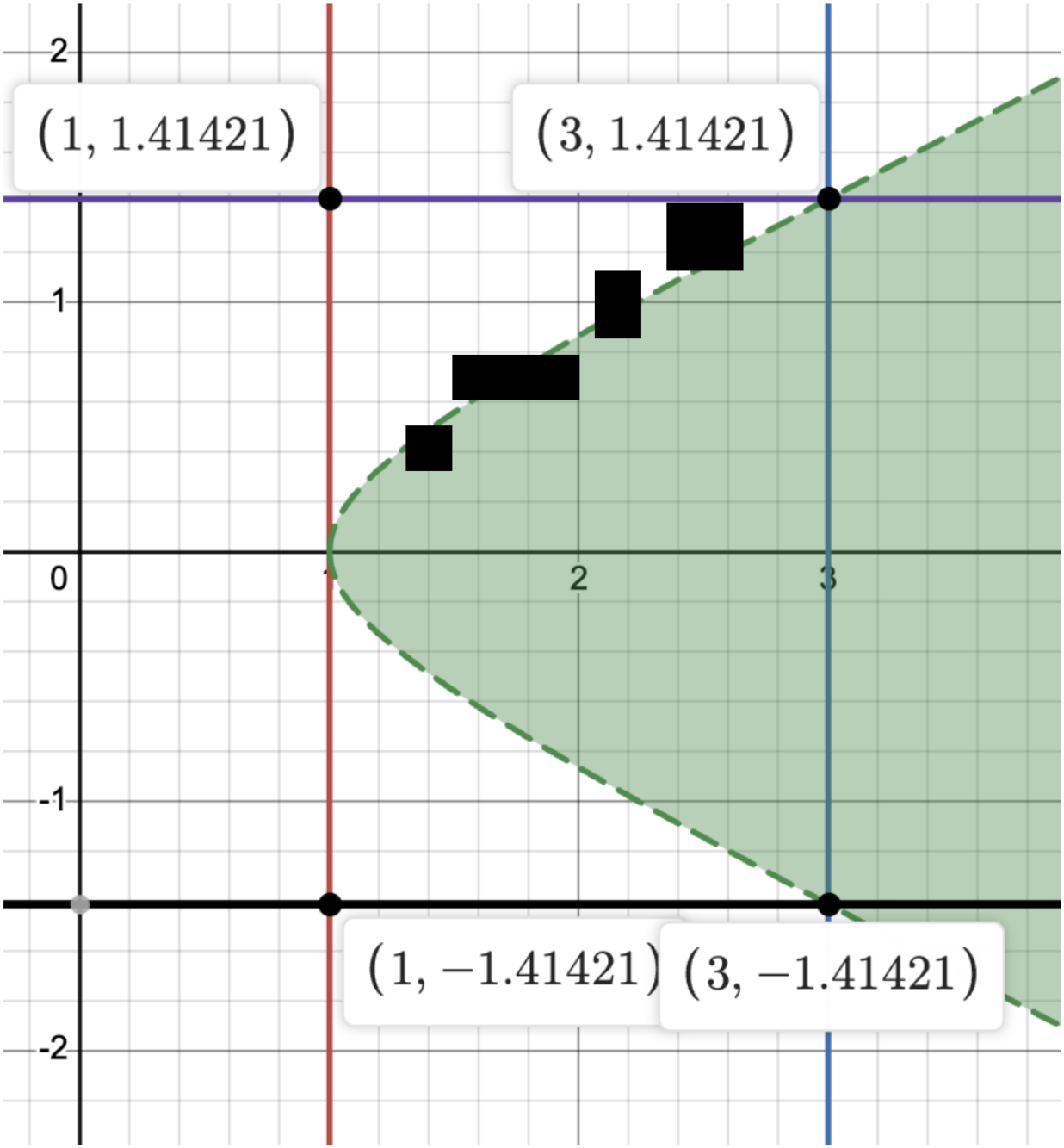


Рисунок 2: Варианты пересечений

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.

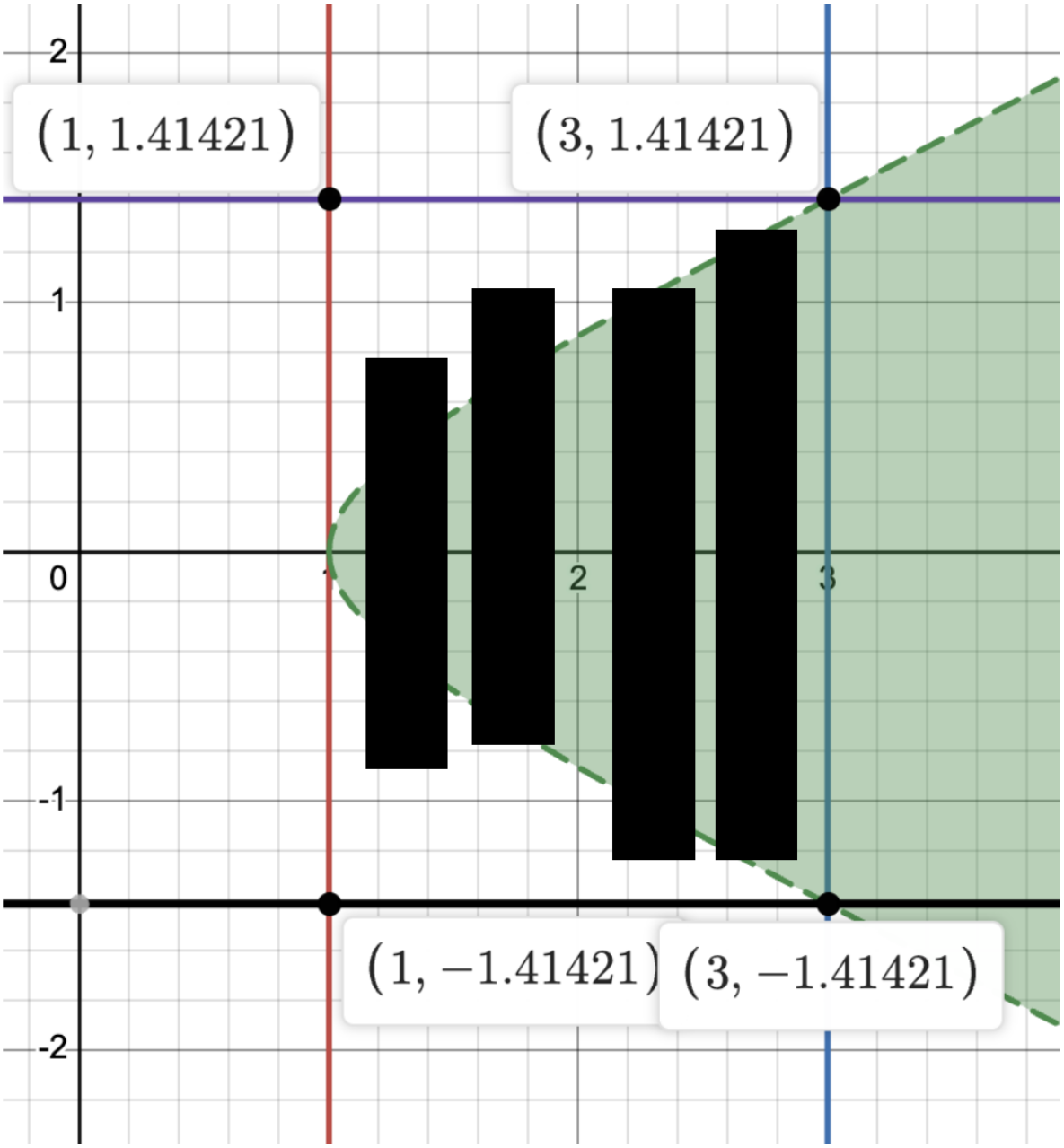


Рисунок 3: Варианты пересечений

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.

На рисунке 4 продемонстрирован финальный ответ метода.

Далее представлены результаты экспериментов и графики ускорений.

Таблица 1: Таблица с результатами расчетов на ПВС IBM Polus последовательной программы.

Число точек сетки ($M \times N$)	**Число итераций**	**Время решения(мс**
10×10	927	18
20×20	11773	861
40×40	109441	31613

Анализ работы gpu-частей программы представлен в таблице 6

Таблица 2: Таблица с результатами сравнения последовательной программы и OpenMP код

OpenMP-нити	**Число точек сетки**	**Число итераций**	**Время**	**Ускорение**
1	40×40	109441	31613	1
2	40×40	109441	17135	1.84
4	40×40	109441	14589	2.16
8	40×40	109441	12060	2.62

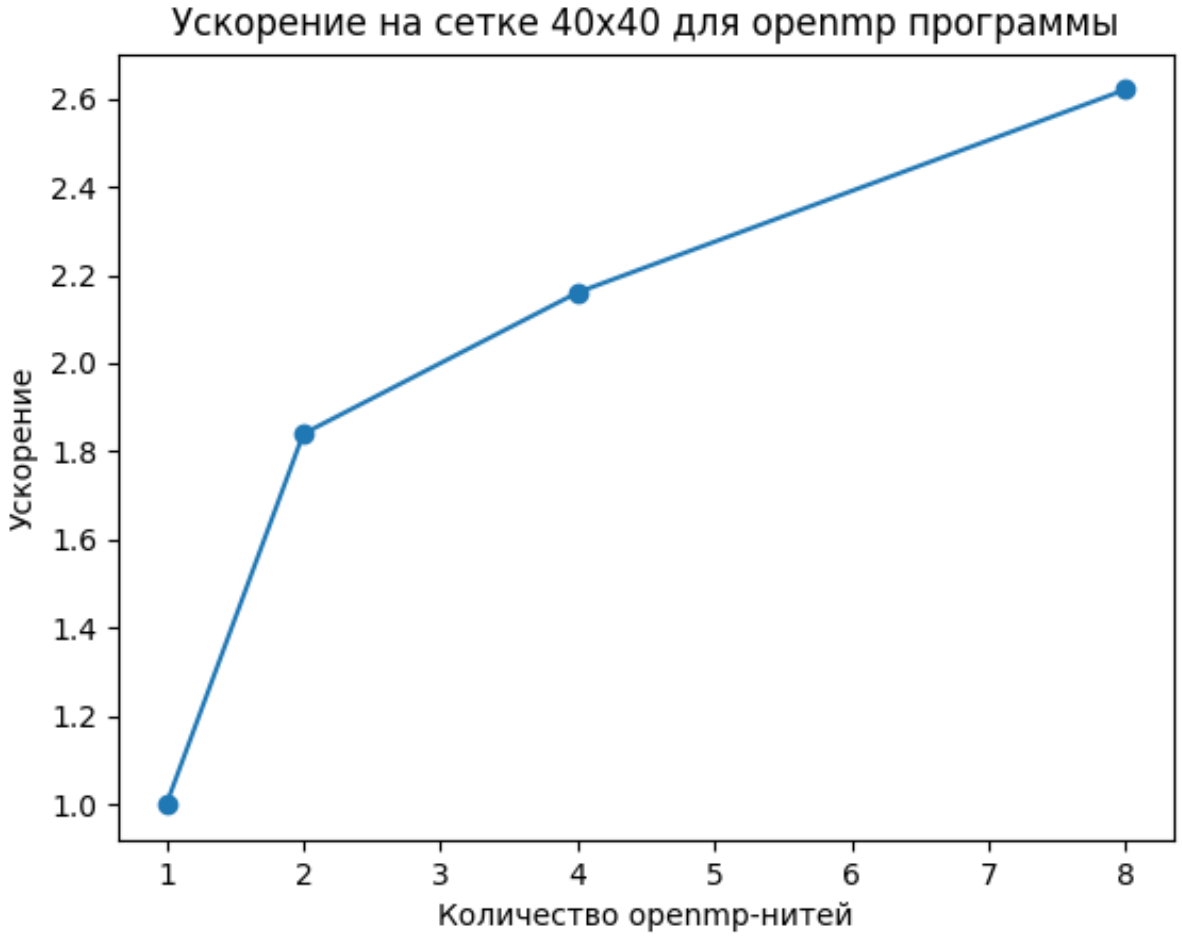
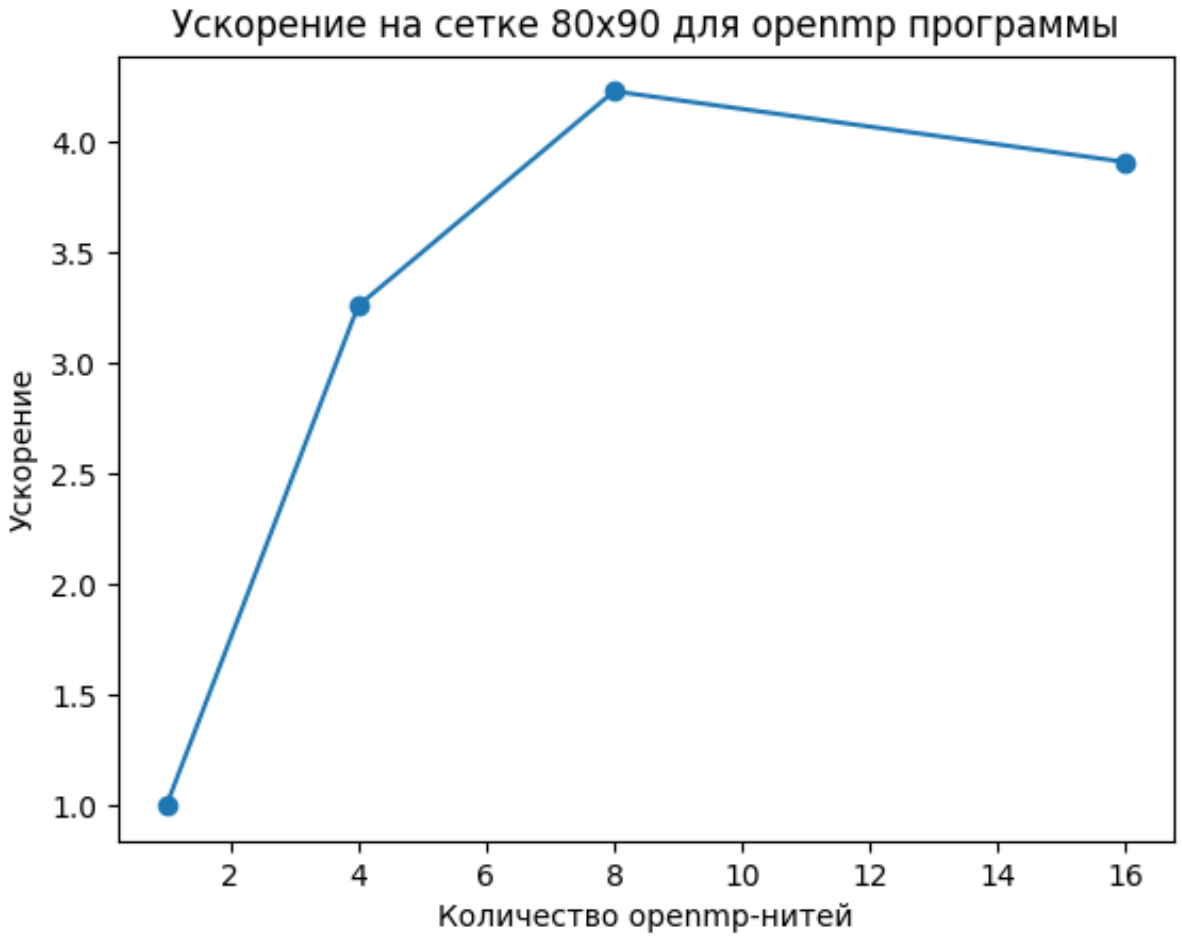


Таблица 3: Таблица с результатами расчетов на ПВС IBM Polus (OpenMP код).

OpenMP-нити	**Число точек сетки**	**Число итераций**	**Время**	**Ускорение**
1	80×90	216073	77764	1
4	80×90	216073	23841	3.26
8	80×90	216073	18364	4.23
16	80×90	216073	19883	3.91
1	160×180	436649	627027	1
4	160×180	436649	165407	3.79
8	160×180	436649	96835	6.47
16	160×180	436649	97567	6.42
32	160×180	436649	106537	5.88


Ускорение на сетке 80x90 для openmp программы
Ускорение
Количество openmp-нитей
1.0
1.5
2.0
2.5
3.0
3.5
4.0
2
4
6
8
10
12
14
16

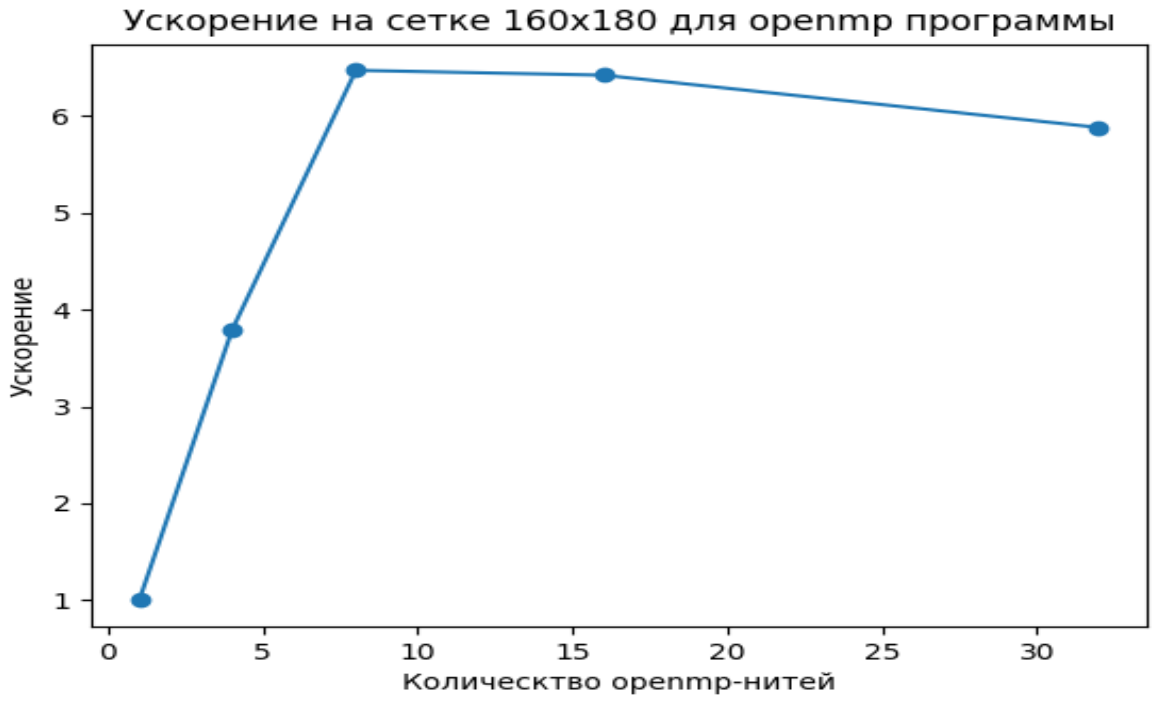
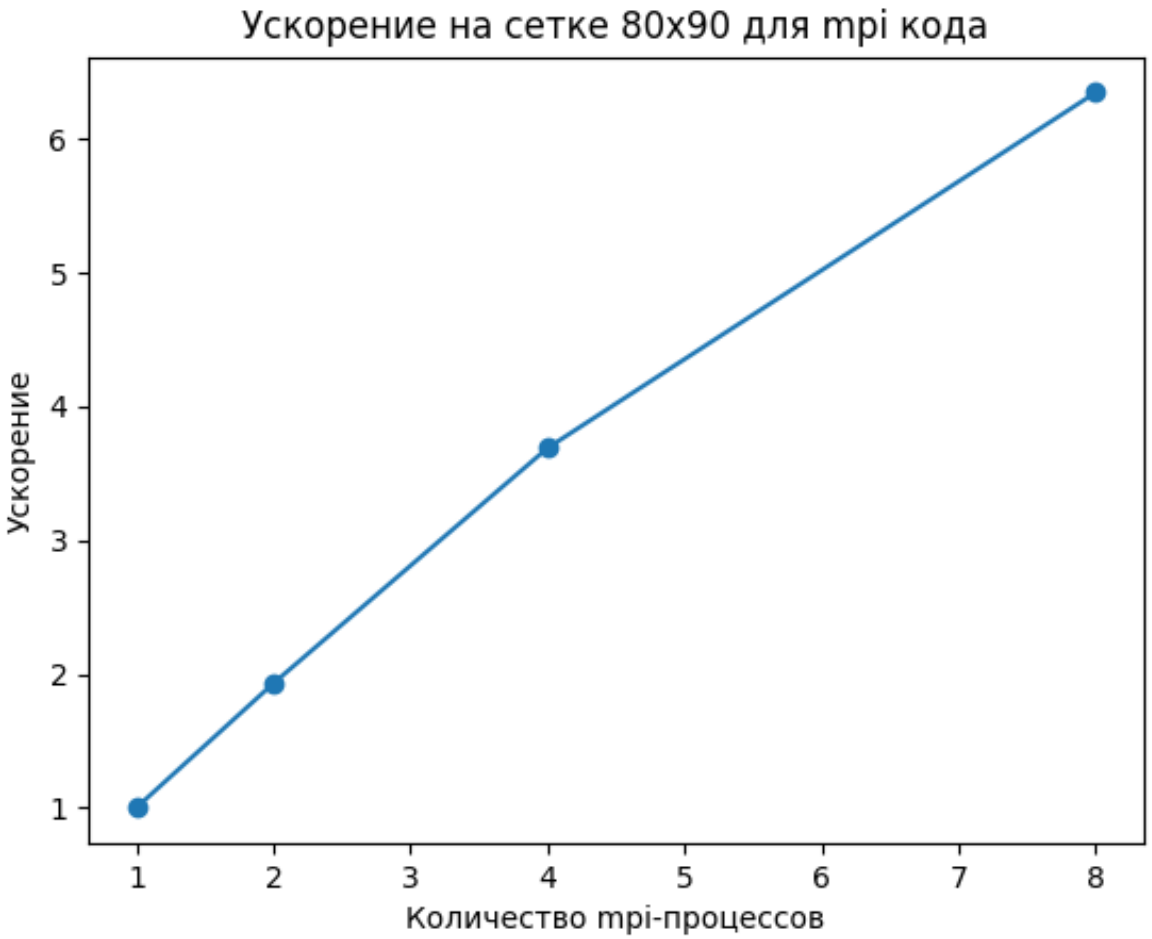

Ускорение на сетке 160x180 для openmp программы
Ускорение
Колическтво openmp-нитей
1
2
3
4
5
6
0
5
10
15
20
25
30

Таблица 4: Таблица с результатами расчетов на ПВС IBM Polus (MPI код).

MPI-процессы	**Число точек сетки**	**Число итераций**	**Время**	**Ускорение**
1	80×90	216073	123074	1
2	80×90	216073	63709	1.93
4	80×90	216073	33294	3.69
8	80×90	216073	19368	6.35
1	160×180	436649	1000830	1
2	160×180	436649	509306	1.96
4	160×180	436649	258794	3.86
8	160×180	436649	134026	7.46
16	160×180	436649	72921	13.72



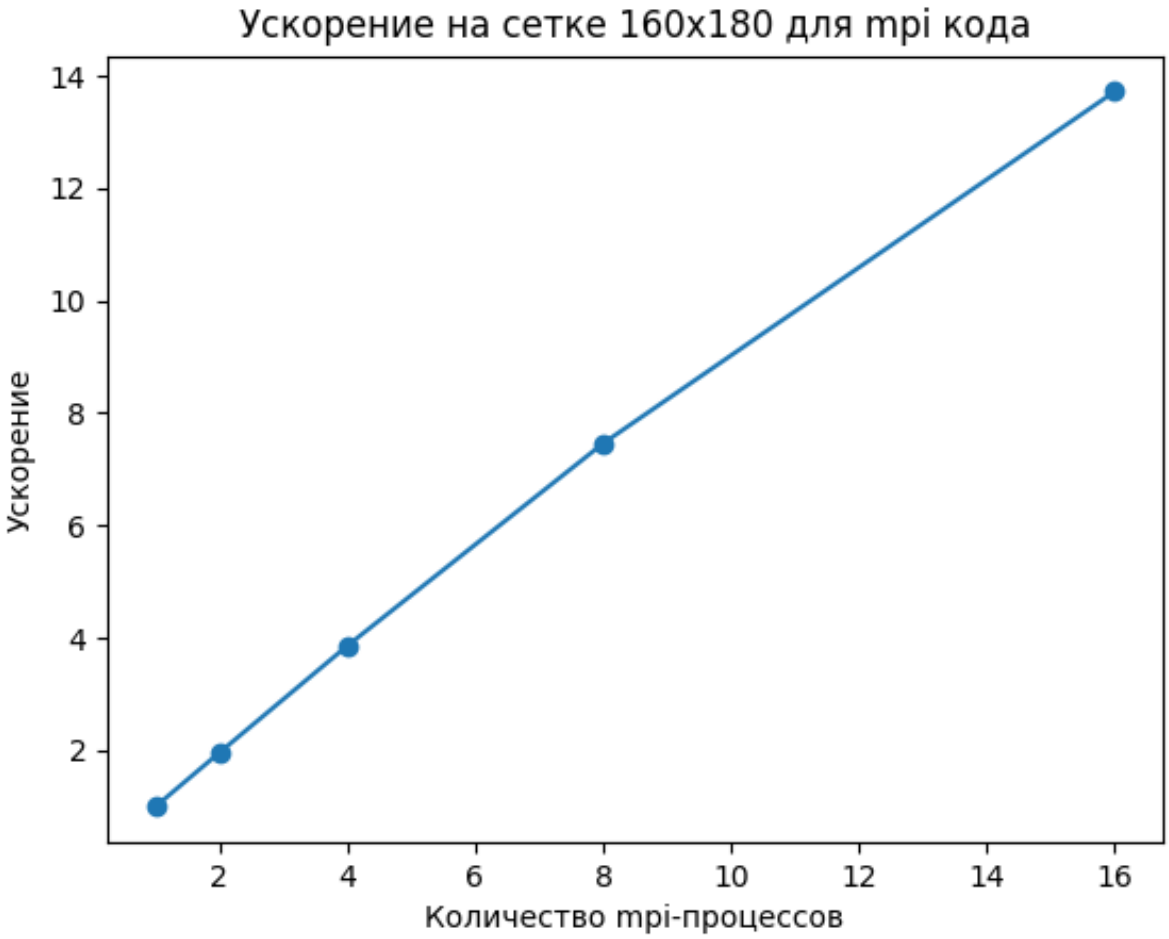
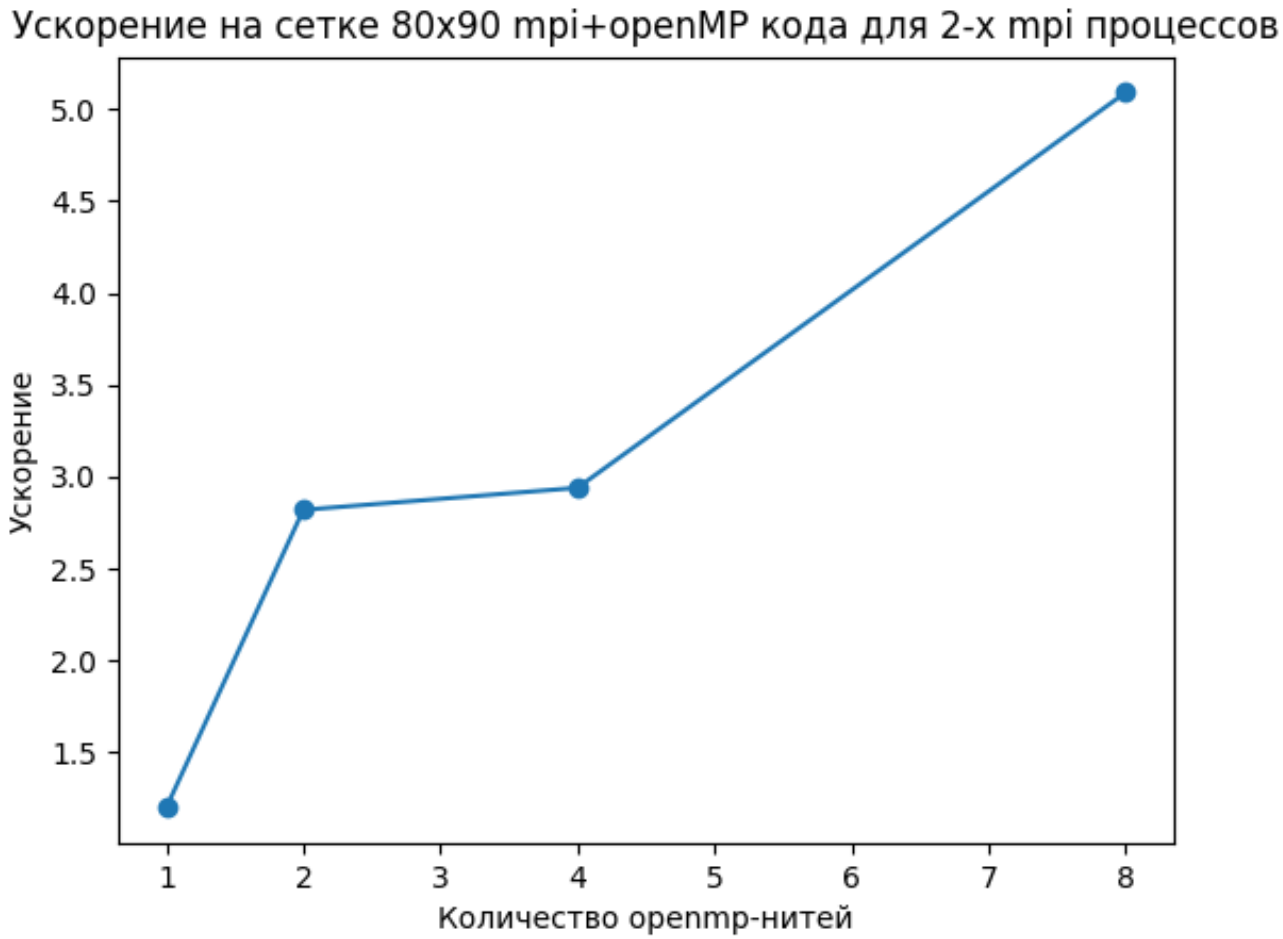
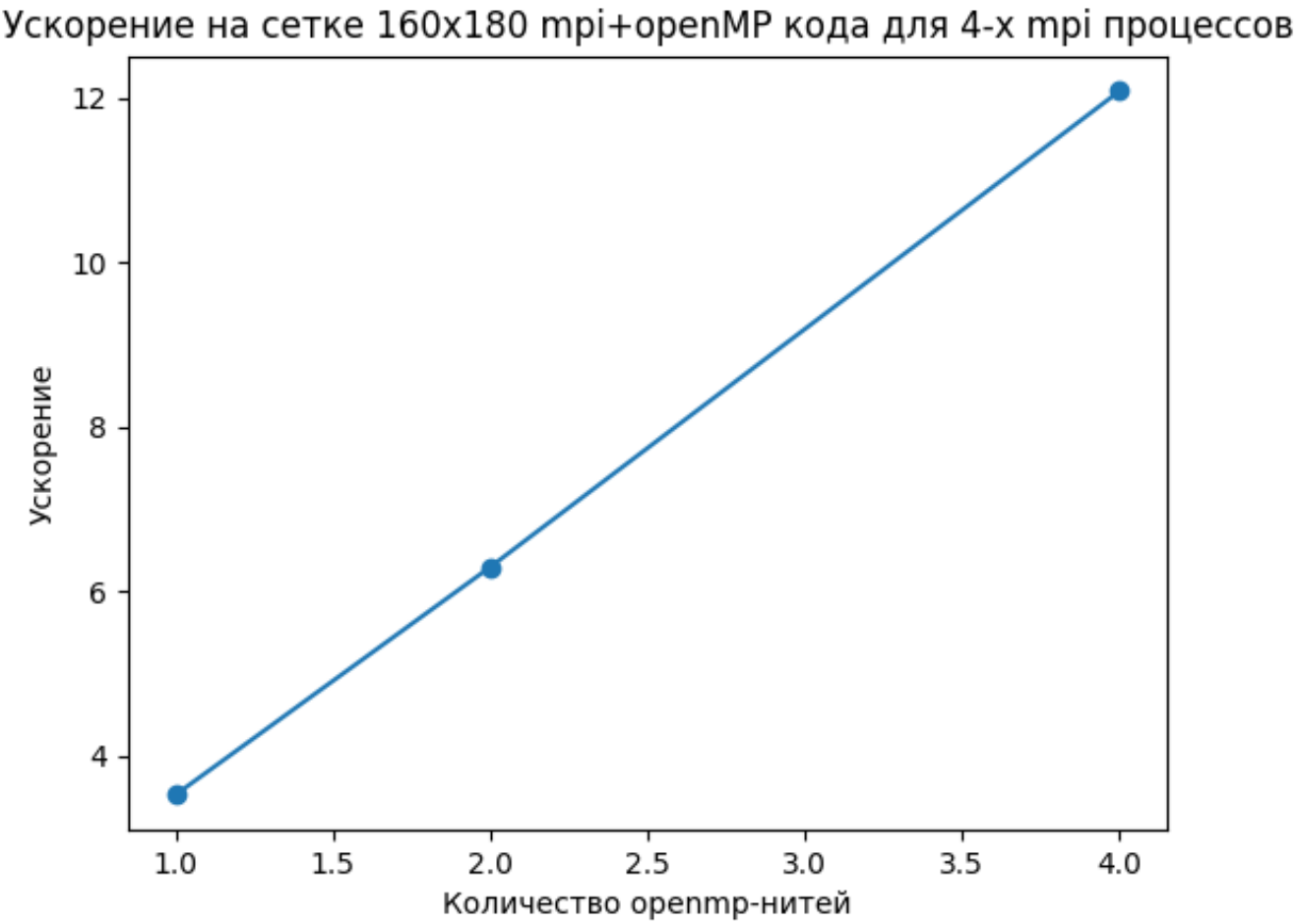


Таблица 5: Таблица с результатами расчетов на ПВС IBM Polus.

MPI-процессы	OpenMP-нити	Число точек сетки	Число итераций	Время	Ускорение
1	1	80×90	216073	125169	1
2	1	80×90	216073	104224	1.20
2	2	80×90	216073	44343	2.82
2	4	80×90	216073	42522	2.94
2	8	80×90	216073	24565	5.09
1	1	160×180	436649	1036870	1
4	1	160×180	436649	293674	3.53
4	2	160×180	436649	164411	6.30
4	4	160×180	436649	85736	12.09
4	8	160×180	436649	???	???

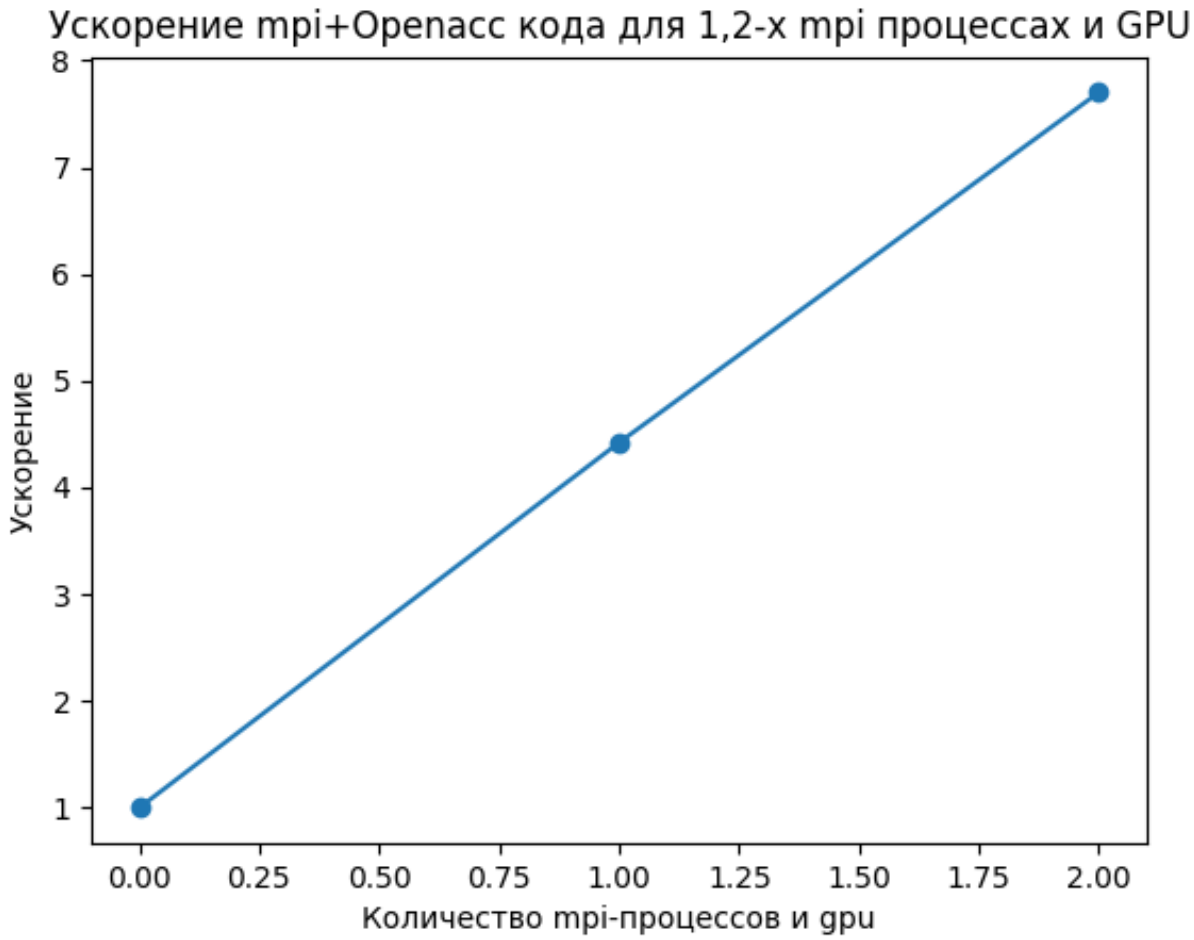

Ускорение на сетке 80x90 mpi+openMP кода для 2-х mpi процессов
Ускорение
5.0
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1
2
3
4
5
6
7
8
Количество openmp-нитей



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 результаты: 1. Openmp программа позволяет ускорить выполение по сравнению с последовательной версией, ускорение ожидаемо примерно линейно по количеству потоков. На графиках присутствует точка максимума - вероятная причина - недостаточный размер сеток для таких ресурсов. 2. Mpi позволяет ускорить программу, ускорение линейно по количеству mpi-процессов. По сравнению с openmp версией точка пика ускорения находится при большем количестве mpi-процессов. При этом ускорение увеличивается с размером сетки, так как уменьшаются затраты на коммуникацию между процессами. 3. Mpi+GPU показывает очень хорошее ускорение непосредственно циклов работы по сравнению с посоедовательной программой. Некоторые циклы ускоряются в 40 раз. Но при этом тратится больше времени на пересылку данных между хостом и устройством. Разницs между двумя mpi-процессами с GPU и одним процессом почти нет. В этом случае матрица уменьшается в два раза, ускорения нет предположительно



из-за того, что параметры задачи из условия не используют все ресурсы gpu.

Таблица 6: Таблица с результатами времени выполнения частей программы

Операция	**Последовательная**	**1GPU, 1 mpi**	**2GPU, 2 mpi**
Вычисление AW	586.42	13.66	13.26
Перенос r с gpu	0.02	10.75	11.04
Вычисление (r,r)	10.03	40.18	41.19
Асинхронная отправка r	0.03	0.03	1.10
Перенос r на gpu	0.02	8.67	9.25
Вычисление Ar	464.2	13.74	13.42
Вычисление (Ar,r)	10.03	40.11	40.85
Вычисление tau	0.36	2.45	5.04
Вычисление new_w	20.05	10.86	11.43
Общее время решения	1091.40	140.83	148.73

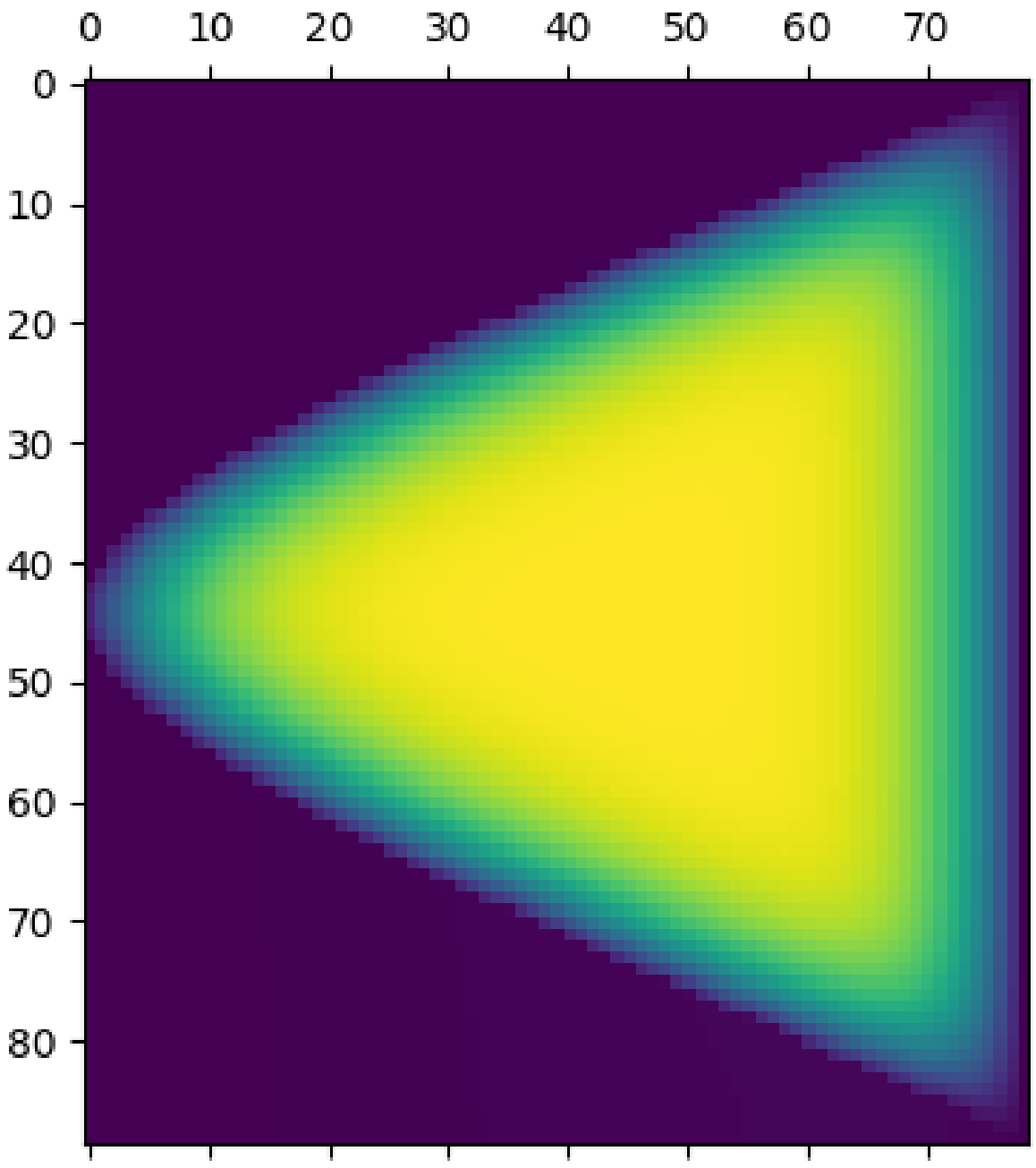


Рисунок 4: Финальный ответ метода