



Отчёт по практическому заданию по курсу  
«Суперкомпьютерное моделирование и технологии программирования»  
**«Метод конечных разностей приближенного решения задачи  
Дирихле для уравнения Пуассона в прямоугольной области»**  
Вариант 35

Аят Оспанов  
617 группа, ММП, ВМК МГУ, Москва  
28 ноября 2017 г.

## Содержание

<b>1 Формулировка задания</b>	<b>2</b>
1.1 Математическая постановка дифференциальной задачи . . . . .	2
1.2 Разностная схема решения задачи . . . . .	2
1.3 Метод решения системы линейных алгебраических уравнений . . . . .	3
<b>2 Условия задания</b>	<b>4</b>
<b>3 Аналитическое решение</b>	<b>4</b>
<b>4 Краткое описание проделанной работы</b>	<b>5</b>
<b>5 Результаты для «IBM BlueGene/P»</b>	<b>6</b>
5.1 Выводы . . . . .	7
<b>6 Результаты для «Ломоносов»</b>	<b>8</b>
6.1 Выводы . . . . .	8
<b>7 Результаты численного решения задачи</b>	<b>8</b>
<b>8 Реализация на CUDA</b>	<b>10</b>
8.1 Выводы . . . . .	11
<b>9 Приложение</b>	<b>12</b>

# 1 Формулировка задания

Требуется методом конечных разностей приближенно решить задачу Дирихле для уравнения Пуассона в прямоугольной области. Задание необходимо выполнить на следующих ПВС Московского университета:

- IBM Blue Gene/P
- «Ломоносов»

## 1.1 Математическая постановка дифференциальной задачи

В прямоугольной области

$$\Pi = [A_1, A_2] \times [B_1, B_2]$$

требуется найти дважды гладкую функцию  $u = u(x, y)$ , удовлетворяющую дифференциальному уравнению

$$-\Delta u = F(x, y), \quad A_1 < x < A_2, \quad B_1 < y < B_2 \quad (1)$$

и дополнительному условию

$$u(x, y) = \varphi(x, y) \quad (2)$$

во всех граничных точках  $(x, y)$  прямоугольника. Оператор Лапласа  $\Delta$  определен равенством:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$$

Функции  $F(x, y), \varphi(x, y)$  указаны в разделе 2.

## 1.2 Разностная схема решения задачи

В расчетной области  $\Pi$  определяется прямоугольная сетка

$$\bar{\omega}_h = \{(x_i, y_j), \quad i = 0, 1, \dots, N_1, \quad j = 0, 1, \dots, N_2\},$$

где  $A_1 = x_0 < x_1 < \dots < x_{N_1} = A_2$  – разбиение отрезка  $[A_1, A_2]$  оси  $(Ox)$ ,  
 $B_1 = y_0 < y_1 < \dots < y_{N_2} = B_2$  – разбиение отрезка  $[B_1, B_2]$  оси  $(Oy)$ .

Через  $\omega_h$  обозначим множество внутренних, а через  $\gamma_h$  – множество граничных узлов сетки  $\bar{\omega}_h$ . Пусть  $h_i^{(1)} = x_{i+1} - x_i, \quad i = 0, 1, \dots, N_1 - 1, \quad h_j^{(2)} = y_{j+1} - y_j, \quad j = 0, 1, \dots, N_2 - 1$ , – переменный шаг сетки по оси абсцисс и ординат соответственно. Средние шаги сетки определяются равенствами:

$$\bar{h}_i^{(1)} = 0.5(h_i^{(1)} + h_{i-1}^{(1)}), \quad \bar{h}_j^{(2)} = 0.5(h_j^{(2)} + h_{j-1}^{(2)}).$$

Рассмотрим линейное пространство  $H$  функций, заданных на сетке  $\omega_h$ . Будем считать, что в пространстве  $H$  задано скалярное произведение и евклидова норма

$$(u, v) = \sum_{i=1}^{N_1-1} \sum_{j=1}^{N_2-1} \bar{h}_i^{(1)} \bar{h}_j^{(2)} u_{ij} v_{ij}, \quad \|u\| = \sqrt{(u, u)}, \quad (3)$$

где  $u_{ij} = u(x_i, y_j)$ ,  $v_{ij} = v(x_i, y_j)$  – любые функции из пространства  $H$ .

Для аппроксимации уравнения Пуасона (1) воспользуемся пятиточечным разностным оператором Лапласа, который во внутренних узлах сетки определяется равенством:

$$-\Delta_h p_{ij} = \frac{1}{\bar{h}_i^{(1)}} \left( \frac{p_{ij} - p_{i-1j}}{\bar{h}_{i-1}^{(1)}} - \frac{p_{i+1j} - p_{ij}}{\bar{h}_i^{(1)}} \right) + \frac{1}{\bar{h}_j^{(2)}} \left( \frac{p_{ij} - p_{ij-1}}{\bar{h}_{j-1}^{(2)}} - \frac{p_{ij+1} - p_{ij}}{\bar{h}_j^{(2)}} \right)$$

Здесь предполагается, что функция  $p = p(x_i, y_j)$  определена во всех узлах сетки  $\bar{\omega}_h$ .

Приближенным решением задачи (1), (2) называется функция  $p = p(x_i, y_j)$ , удовлетворяющая уравнениям

$$\begin{cases} -\Delta_h p_{ij} = F(x_i, y_j), & (x_i, y_j) \in \omega_h, \\ p_{ij} = \varphi(x_i, y_j), & (x_i, y_j) \in \gamma_h. \end{cases} \quad (4)$$

Эти соотношения представляют собой систему линейных алгебраических уравнений с числом уравнений равным числу неизвестных и определяют единственным образом неизвестные значения  $p_{ij}$ . Совокупность уравнений (4) называется разностной схемой для задачи (1), (2).

### 1.3 Метод решения системы линейных алгебраических уравнений

Приближенное решение системы уравнений (4) может быть получено итерационным методом сопряженных градиентов. В этом методе начальное приближение

$$\begin{cases} p_{ij}^{(0)} = \varphi(x_i, y_j), & (x_i, y_j) \in \gamma_h, \\ p_{ij}^{(0)} = \text{любые числа}, & (x_i, y_j) \in \omega_h, \end{cases}$$

Последующие итерации осуществляются по формулам:

$$p_{ij}^{(k+1)} = p_{ij}^{(k)} - \tau_{k+1} g_{ij}^{(k)}, \quad k = 0, 1, 2, \dots$$

Здесь

$$\tau_{k+1} = \frac{(r^{(k)}, g^{(k)})}{(-\Delta_h g^{(k)}, g^{(k)})},$$

градиент

$$\begin{cases} g_{ij}^{(k)} = r_{ij}^{(k)} - \alpha_k g_{ij}^{(k-1)}, & k = 1, 2, \dots, \\ g_{ij}^{(0)} = r_{ij}^{(0)}, \end{cases}$$

коэффициент

$$\alpha_k = \frac{(-\Delta_h r^{(k)}, g^{(k-1)})}{(-\Delta_h g^{(k-1)}, g^{(k-1)})},$$

невязка

$$\begin{cases} r_{ij}^{(k)} &= -\Delta_h p_{ij}^{(k)} - F(x_i, y_j), \quad (x_i, y_j) \in \omega_h \\ r_{ij}^{(k)} &= 0, \quad (x_i, y_j) \in \gamma_h \end{cases}$$

Итерационный процесс останавливается, как только

$$||p^{(n)} - p^{(n-1)}|| < \varepsilon,$$

где  $\varepsilon$  – заранее выбранное положительное число.

## 2 Условия задания

В данном варианте (вариант 35) правая часть уравнения (1) и граничное условие (2) определяются следующим образом:

$$F(x, y) = \frac{8(1 - x^2 - y^2)}{(1 + x^2 + y^2)^3} \quad (5)$$

$$\varphi(x, y) = \frac{2}{1 + x^2 + y^2} \quad (6)$$

Дифференциальную задачу надо решить для данных функций на прямоугольнике

$$\Pi = [0, 2] \times [0, 2]$$

на неравномерной сетке, определенной равенствами:

$$\begin{aligned} x_i &= A_2 f(i/N_1) + A_1(1 - f(i/N_1)), \quad i = 0, 1, \dots, N_1, \\ y_j &= B_2 f(j/N_2) + B_1(1 - f(j/N_2)), \quad j = 0, 1, \dots, N_2. \end{aligned} \quad (7)$$

где

$$f(t) = \frac{(1+t)^q - 1}{2^q - 1}, \quad 0 \leq t \leq 1, \quad q = 3/2$$

## 3 Аналитическое решение

Для решения задачи (1) с граничными условиями (2), предположим, что  $u(x, y) = \varphi(x, y)$  для всех точек прямоугольника. Тогда должно выполняться

$$-\Delta \varphi(x, y) = F(x, y)$$

Проверим это: т.к.  $\varphi(x, y)$  симметрична относительно  $x$  и  $y$ , то просто найдем  $\frac{\partial^2 \varphi(x, y)}{\partial x^2}$ :

$$\begin{aligned}\frac{\partial^2 \varphi(x, y)}{\partial x^2} &= \frac{\partial}{\partial x} \frac{\partial \varphi(x, y)}{\partial x} = \left\{ \frac{\partial}{\partial x} \varphi(x, y) = \frac{\partial}{\partial x} \frac{2}{1 + x^2 + y^2} = \frac{-4x}{(1 + x^2 + y^2)^2} \right\} = \\ &= \frac{\partial}{\partial x} \frac{-4x}{(1 + x^2 + y^2)^2} = \frac{16x^2 - 4(1 + x^2 + y^2)}{(1 + x^2 + y^2)^3} = \frac{12x^2 - 4y^2 - 4}{(1 + x^2 + y^2)^3}\end{aligned}$$

Теперь посчитаем  $-\Delta \varphi(x, y)$ :

$$\begin{aligned}-\Delta \varphi(x, y) &= -\frac{12x^2 - 4y^2 - 4}{(1 + x^2 + y^2)^3} - \frac{12y^2 - 4x^2 - 4}{(1 + x^2 + y^2)^3} = \\ &= -\frac{12x^2 - 4y^2 - 4 + 12y^2 - 4x^2 - 4}{(1 + x^2 + y^2)^3} = -\frac{8x^2 + 8y^2 - 8}{(1 + x^2 + y^2)^3} = \\ &= \frac{8(1 - x^2 - y^2)}{(1 + x^2 + y^2)^3} = F(x, y)\end{aligned}$$

Т.к. решение задачи Дирихле единственное, то получаем, что наше предположение верно и

$$u(x, y) = \frac{2}{1 + x^2 + y^2}$$

на всем прямоугольнике.

## 4 Краткое описание проделанной работы

Для создания параллельной программы, решающей дифференциальное уравнение, были использованы язык C++ и библиотеки MPI и OpenMP. Параллелизация MPI выполнялась следующим образом:

- Сетка разделялась на  $2^p$  заданных процессов так, чтобы по оси  $Ox$  было  $\left[\frac{p}{2}\right]$  блоков, по оси  $Oy - (1 - \left[\frac{p}{2}\right])$ . Каждому процессу соответствует один блок.
- Каждый блок расширялся на 1 строку/столбец влево/вправо для хранения данных соседних строк/столбцов (если таковые есть)
- Каждый процесс, после определенных вычислений, обменивался данными с соседними.

Параллелизация OpenMP проводилась в каждом процессе для распараллеливания циклов при подсчетах.

Для удобства была использована парадигма ООП:

- Каждая двумерная функция представлялась как класс Func2D
- В классе Func2D были перегружены следующие операторы (приведены основные):
  - умножения на класс Func2D – скалярное произведение
  - умножения на число

- оператор  $\sim$  – оператор Лапласа
- Реализован класс-синглтон MPIHelper – обертка над MPI
- Реализован класс ConjugateGradientMethod – собственно сам метод сопряженных градиентов

Также предусмотрена возможность включать директивы OpenMP с помощью задания идентификатора USE\_OMP при компиляции программы (-D"USE\_OMP").

## 5 Результаты для «IBM BlueGene/P»

Написанная программа многократно запускалась с разными параметрами на суперкомпьютере IBM BlueGene/P. Т.к. использовалась неравномерная сетка, то подсчеты проводились относительно долго по сравнению с равномерной сеткой, даже при предварительном подсчете шагов сетки. В связи с этим, на компьютере BlueGene/P на 1 процессоре программа не укладывалась в заданные рамки 2x часов. Поэтому было решено ограничить количество итераций, что не повлияет на подсчет ускорений. По итогу замеров времени работы, получились результаты, представленные в Таблице 1 для версии MPI без использования OpenMP директив и в Таблице 2 для версии MPI с OpenMP директивами (3 нити на процесс).

В дополнение к запускам на 500 итераций, были сделаны замеры времени работы программы до схождения итерационного процесса. Результаты можно посмотреть в Таблицах 8 и 9 в Разделе “Приложение”.

**Таблица 1:** MPI версия программы на 500 итераций на BlueGene/P

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		916.56	1
64		14.70	62.35
128	1000 × 1000	7.87	116.46
256		4.11	223.01
512		2.34	391.70
1		3737.03	1
64		56.71	65.90
128	2000 × 2000	29.12	128.33
256		14.75	253.36
512		7.92	471.85

**Таблица 2:** MPI+OpenMP версия программы на 500 итераций на BlueGene/P

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		321.63	2.85
64		5.71	160.52
128	$1000 \times 1000$	3.38	271.17
256		2.00	458.28
512		1.32	694.36
1		1292.99	2.89
64		20.52	182.12
128	$2000 \times 2000$	10.93	341.91
256		5.73	652.19
512		3.38	1105.63

Также для удобства сравнения MPI и MPI+OpenMP версий программ представлена Таблица 3.

**Таблица 3:** Сравнение MPI версии программы с MPI+OpenMP версией на BlueGene/P ( $S_{MPI}$  – ускорение MPI версии программы,  $S_{MPI+OMP}$  – ускорение MPI+OpenMP версии программы)

Число процессоров	Число точек сетки	$S_{MPI}$	$S_{MPI+OMP}$	$S_{MPI+OMP}/S_{MPI}$
1		1	2.85	2.85
64		62.35	160.52	2.57
128	$1000 \times 1000$	116.46	271.17	2.33
256		223.01	458.28	2.05
512		391.7	694.36	1.77
1		1	2.89	2.89
64		65.9	182.12	2.76
128	$2000 \times 2000$	128.33	341.91	2.66
256		253.36	652.19	2.57
512		471.85	1105.63	2.34

## 5.1 Выводы

Как видно из Таблицы 1, ускорение программы линейно относительно количества процессоров. При этом на сетке  $1000 \times 1000$  ускорение чуть меньше, чем количество процессоров, тогда как на сетке  $2000 \times 2000$  ускорение примерно равно количеству процессоров. Связано это с нагрузженностью процессоров. При большой сетке нагрузженность больше, что увеличивает производительность. Также из Таблицы 3 видно, что ускорение MPI+OpenMP версии примерно в 3 раза больше, чем у MPI версии, что подтверждает теоретическое ускорение при использовании трех нитей OpenMP.

## 6 Результаты для «Ломоносов»

Программа также тестировалась на суперкомпьютере «Ломоносов». Но по сравнению с BlueGene/P, на Ломоносов использовалась только MPI версия. На данном компьютере также ограничивалось количество итераций. В результате запусков получилась Таблица 4

Таблица 4: MPI версия программы на 500 итераций на Ломоносов

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		139.68	1
8		19.48	7.17
16		8.98	15.55
32	$1000 \times 1000$	4.53	30.83
64		2.79	50.06
128		1.83	76.33
1		559.45	1
8		80.42	6.96
16		39.05	14.33
32	$2000 \times 2000$	19.64	28.49
64		9.58	58.4
128		4.85	115.35

Также, как и в случае с BlueGene/P, для Ломоносов можно посмотреть результаты работы программы до схождения в Таблице 10 в Разделе “Приложение”.

### 6.1 Выводы

По таблице 4 можно сделать следующие выводы: для сетки  $1000 \times 1000$  ускорение больше для меньших количеств процессоров, а именно для 8, 16 и 32, в то время как для сетки  $2000 \times 2000$  ускорение больше для 64 и 128 процессоров. Объяснить это можно нагрузкой на процессоры и объемом обмениваемых процессами данных. Если в случае малых количеств процессоров и малой сетки процессоры нагружаются достаточно, то при большой сетке процессоры делают больше вычислений и процессы обмениваются большим количеством данных (т.к. данные делятся на меньшее количество процессов), что увеличивает время работы программы. Данного явления на суперкомпьютере BlueGene/P не наблюдалось, т.к. там используются от 64 процессоров. В случае от 64 процессоров, на Ломоносов та же ситуация, что и с BlueGene/P: ускорение больше для большей сетки, т.к. объем обмениваемых данных меньше.

## 7 Результаты численного решения задачи

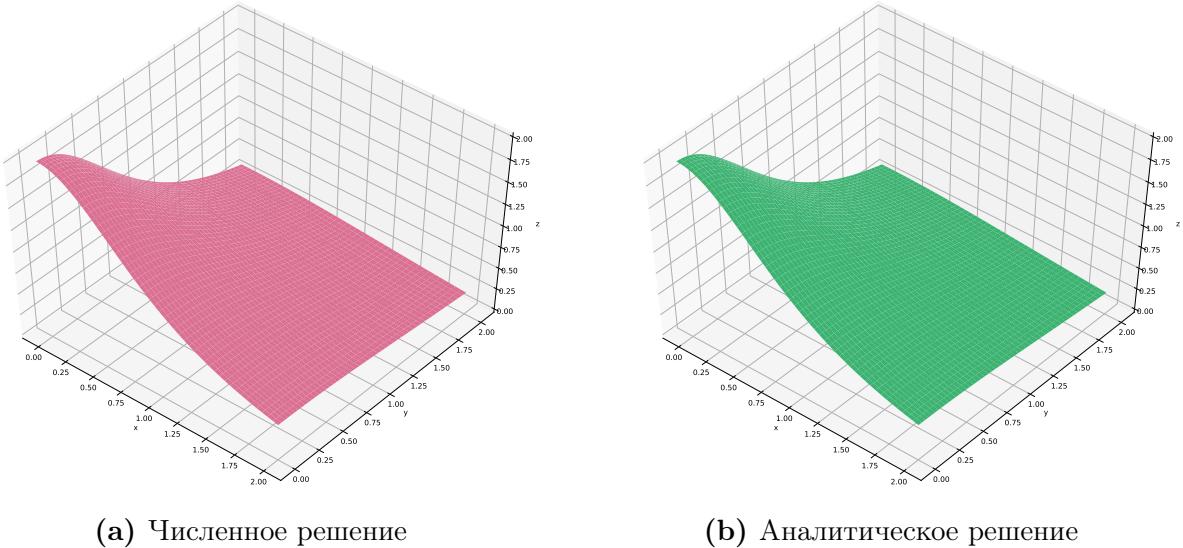
В итоге работы программы, были получены решения с указанными в Таблице 5 невязками между численным и аналитическим решениями и количеством итераций до схожде-

ния.

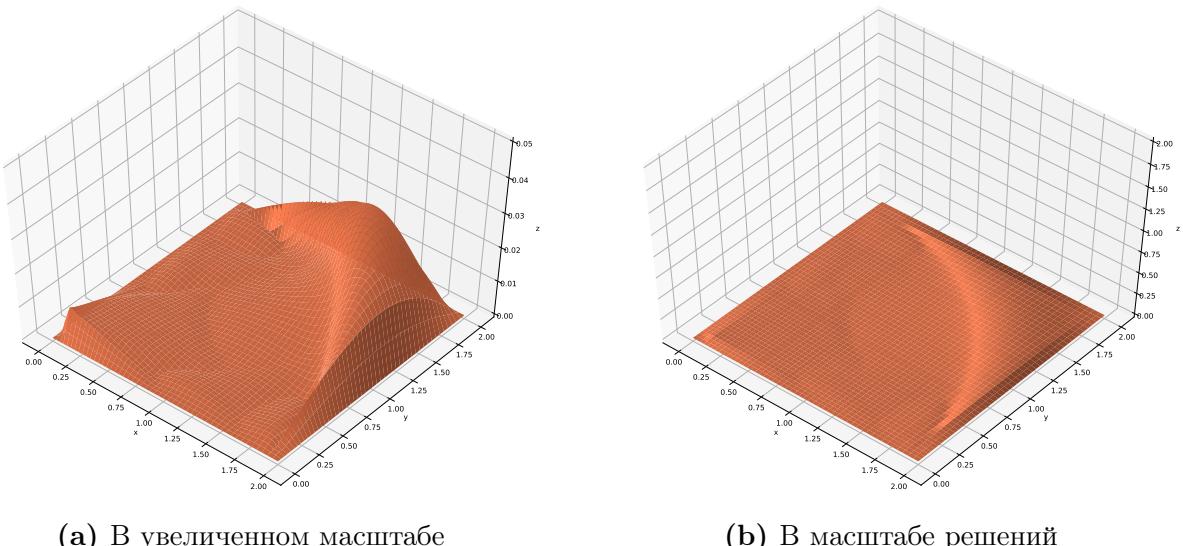
**Таблица 5:** Замеры для численного метода

Размер сетки	Невязка	Количество итераций
$1000 \times 1000$	0.010209	1693
$2000 \times 2000$	0.020420	3251

Ниже представлены графики решений дифференциальной задачи на сетке  $2000 \times 2000$ . По графикам разницы между численным решением (Рис. 1a) и аналитическим решением (Рис. 1b) не видно, т.к. погрешность в каждой точке относительно мала, что видно из Рис. 2b. На Рис. 2a можно увидеть график погрешности в увеличенном масштабе.



**Рис. 1:** Графики решения дифференциальной задачи



**Рис. 2:** Графики разности численного и аналитического решений дифференциальной задачи

## 8 Реализация на CUDA

CUDA версия программы тестировалась на компьютере «Ломоносов» в очереди gputest. Т.к. данная очередь ограничивается максимум 8 узлами, данные были получены только до 16 процессов/видеокарт.

Ускорением программы называют отношение времени выполнения последовательной версии программы к времени выполнения параллельной:

$$S = \frac{T_1}{T_p},$$

где  $T_p$  – время выполнения программы на  $p$  процессах.

Для оценки масштабируемости параллельной программы используется понятие эффективности:

$$E = \frac{S}{p}$$

В результате запусков программы для разных количеств процессоров и видеокарт, были получены Таблицы 6 и 7 времени работы программ и их ускорений.

Ускорение для CUDA версии программы считалось относительно последовательной версии программы, реализованной без использования видеокарт.

Таблица 6: MPI версия программы

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		167.95	1
2		86.12	1.95
4	1000 × 1000	42.66	3.94
8		22.09	7.60
16		10.72	15.67
1		676.53	1
2		346.07	1.95
4	2000 × 2000	173.75	3.89
8		95.30	7.1
16		45.49	14.87

Таблица 7: CUDA версия программы

Число видеокарт	Число точек сетки	Время решения, сек	Ускорение
1	$1000 \times 1000$	101.23	1.66
2		59.05	2.84
4		31.23	5.38
8		18.11	9.27
16		9.99	16.81
1	$2000 \times 2000$	341.68	1.98
2		188.97	3.58
4		104.55	6.47
8		55.39	12.21
16		31.70	21.34

## 8.1 Выводы

Для более наглядного анализа таблиц, нарисуем графики ускорений и эффективностей (Рис. 3 и 4). Эффективности програм считались относительно последовательной версии каждого варианта. Таким образом, для CUDA версии эффективность считалась относительно CUDA версии на одной видеокарте, а для MPI версии – на последовательной версии программы на одном процессоре.

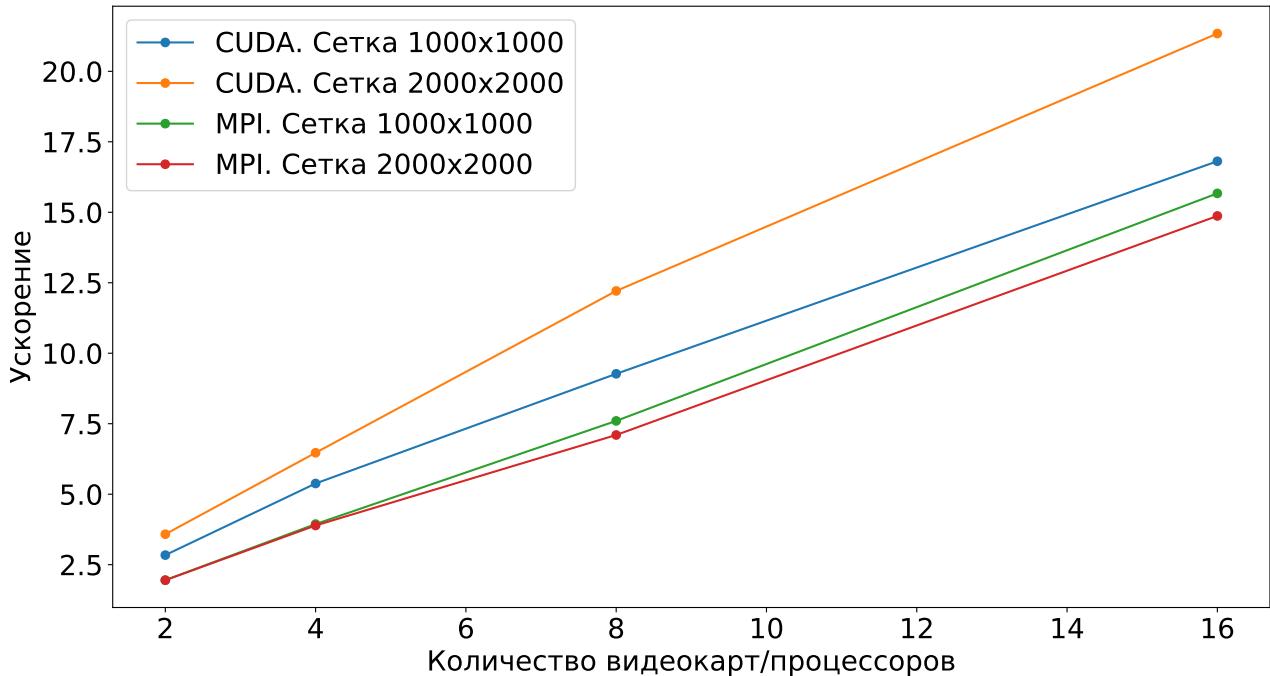
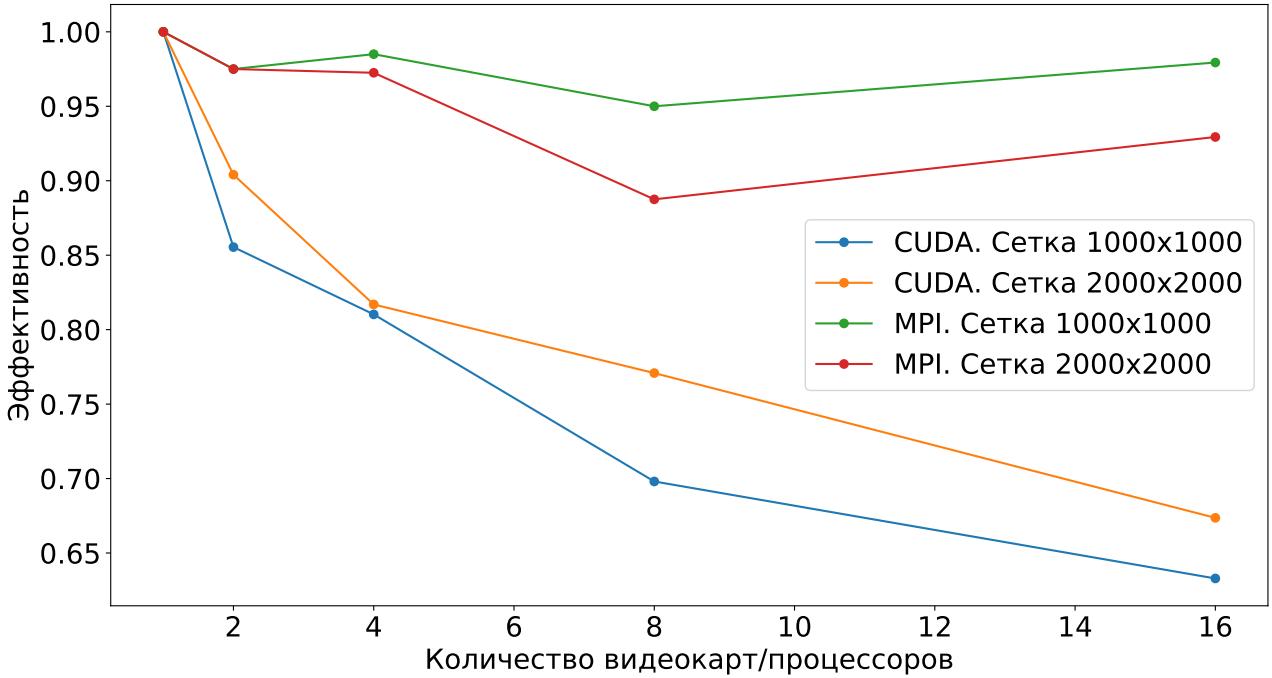


Рис. 3: Ускорение CUDA и MPI версий

Ускорение CUDA версии (Рис. 3) по сравнению с MPI версией на малых размерах сетки не велико, но с ростом размера сетки увеличивается ускорение программы. Данный факт характеризуется особенностью CUDA программ, а именно их “бутылочным горлышком” в

виде загрузки-выгрузки данных. Чем больше данных и меньше вычислений, тем менее эффективна программа. К тому же, в нашем случае, вычисления на карте растут не линейно с ростом данных, а в 5 раз больше, т.к. идет подсчет оператора Лапласа. Таким образом, чем больше сетка, тем больше данных на видеокарту и тем больше она нагружена, что приводит к большему ускорению.



**Рис. 4:** Эффективность CUDA и MPI версий

Также по Рис. 4 эффективности программы можно заметить, что эффективность падает для CUDA программы, но для MPI программы она примерно постоянная. Объясняется это тем, что с увеличением количества видеокарт, данные для обработки одной картой уменьшаются. Это приводит к неполной нагруженности видеокарты. Также при обмене данными между процессами, данные с видеокарты полностью выгружаются, а затем загружаются обратно, что очень затратно при больших количествах данных и процессов. Таким образом, в данной реализации, масштабируемость программы маленькая. Чтобы масштабируемость была хорошей, данные и вычисления должны быть такими, чтобы полностью нагружать видеокарты.

В дополнение стоит отметить, что корректность программы проверялась также, как и в случае с MPI версией – невязкой численного решения и аналитического. Также количество итераций и ошибка каждого шага были одинаковы без учета машинной точности, что подтверждает правильность CUDA программы.

## 9 Приложение

Для всех приведенных ниже таблиц время выполнения на одном процессоре для сетки  $2000 \times 2000$  было выведено примерными расчетами. Они указаны через знак  $\sim$ .

**Таблица 8:** MPI версия программы до сходжения на BlueGene/P

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		3107.29	1
64		49.79	62.41
128	1000 × 1000	26.67	116.51
256		13.91	223.39
512		7.94	391.35
1		~23615.39	1
64		368.69	64.05
128	2000 × 2000	189.29	124.76
256		95.87	246.33
512		51.45	459.00

**Таблица 9:** MPI+OpenMP версия программы до сходжения на BlueGene/P

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		1085.00	2.86
64		19.31	160.92
128	1000 × 1000	11.38	273.05
256		6.68	465.16
512		4.47	695.14
1		~8137.5	2.90
64		133.18	177.32
128	2000 × 2000	71.09	332.19
256		37.26	633.80
512		21.94	1076.36

**Таблица 10:** MPI версия программы до сходжения на Ломоносов

Число процессоров	Число точек сетки	Время решения, сек	Ускорение
1		473.59	1
8		66.18	7.16
16		30.34	15.61
32	1000 × 1000	14.89	31.81
64		7.52	62.98
128		4.54	104.31
1		~3820.60	1
8		523.37	7.3
16		254.14	15.03
32	2000 × 2000	126.12	30.29
64		58.61	65.19
128		29.92	127.69