

Математическая постановка 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:

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

и дополнительному условию:

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

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

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

Вариант 2_23:

- $\Pi = [0, 2] \times [0, 2]$
- $F(x, y) = 2(x^2 + y^2)(1 - 2x^2y^2) \exp(1 - x^2y^2)$
- $\varphi(x, y) = \exp(1 - x^2y^2)$

Численный метод решения задачи

В расчетной области Π определяется прямоугольная сетка

$$\bar{\omega}_h = \{(x_i, y_j), i = 0, 1, 2, \dots, N_1, j = 0, 1, 2, \dots, N_2\},$$

где $A_1 = x_0 < x_1 < x_2 < \dots < x_{N_1} = A_2$ – разбиение отрезка $[A_1, A_2]$ оси (ox) ,

$B_1 = y_0 < y_1 < y_2 < \dots < y_{N_2} = B_2$ – разбиение отрезка $[B_1, B_2]$ оси (oy) .

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

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

Рассмотрим линейное пространство H функций, заданных на сетке $\bar{\omega}_h$. Будем считать, что в пространстве H задано скалярное произведение и евклидова норма

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

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

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

$$-\Delta_h p_{ij} = \frac{1}{h_i^{(1)}} \left(\frac{p_{ij} - p_{i-1,j}}{h_{i-1}^{(1)}} - \frac{p_{i+1,j} - p_{ij}}{h_i^{(1)}} \right) + \frac{1}{h_j^{(2)}} \left(\frac{p_{ij} - p_{i,j-1}}{h_{j-1}^{(2)}} - \frac{p_{i,j+1} - p_{ij}}{h_j^{(2)}} \right).$$

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

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

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).

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

$$p_{ij}^{(0)} = \varphi(x_i, y_j), (x_i, y_j) \in \gamma_h,$$

во внутренних узлах сетки $p_{ij}^{(0)}$ – любые числа. Метод является одношаговым. Итерация $p^{(k+1)}$ вычисляется по итерации $p^{(k)}$ согласно равенствам:

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

где невязка

$$\begin{aligned} r_{ij}^{(k)} &= -\Delta_h p_{ij} - F(x_i, y_j), (x_i, y_j) \in \omega_h, \\ r_{ij}^{(k)} &= 0, (x_i, y_j) \in \gamma_h. \end{aligned} \quad (5)$$

Итерационный параметр

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

Известно, что с увеличением номера итерации k последовательность сеточных функций $p^{(k)}$ сходится к точному решению p задачи (4) по норме пространства H , то есть

$$\|p - p^{(k)}\|_H \rightarrow 0, k \rightarrow +\infty.$$

Существенно большей скоростью сходимости обладает метод сопряженных градиентов. Начальное приближение $p^{(0)}$ и первая итерация $p^{(1)}$ вычисляются так же, как и в методе скорейшего спуска. Последующие итерации осуществляются по формулам:

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

Здесь

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

вектор

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

коэффициент

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

Вектор невязки $r^{(k)}$ вычисляется согласно равенствам (5). Итерационный процесс останавливается, как только

$$\|p^{(n)} - p^{(n-1)}\| < \varepsilon, \quad (6)$$

где ε – заранее выбранное положительное число. В последнем неравенстве, согласно варианту, используется евклидова сеточная норма.

Для аппроксимации дифференциальной задачи используется равномерная прямоугольная сетка:

$$\begin{aligned} x_i &= A_2 \frac{i}{N_1} + A_1 \left(1 - \frac{i}{N_1}\right), i = 0, 1, 2, \dots, N_1, \quad (8) \\ y_j &= B_2 \frac{j}{N_2} + B_1 \left(1 - \frac{j}{N_2}\right), j = 0, 1, 2, \dots, N_2. \end{aligned}$$

Приближенное решение разностной схемы (4) следует вычислять методом сопряженных градиентов. Для остановки итерационного процесса предлагается использовать условие (6), положив $\varepsilon = 10^{-4}$.

Постановка задачи

Для функций $F(x, y), \varphi(x, y)$:

- подобрать точное решение задачи Дирихле,

- методом сопряженных градиентов построить приближенное решение на сетке с числом узлов $N_1 = N_2 = 1000$, определить погрешность решения $\psi = \|u(x_i, y_j) - p_{ij}\|$,
- методом сопряженных градиентов построить приближенное решение на сетке с числом узлов $N_1 = N_2 = 2000$ и вновь определить погрешность решения.

Расчеты необходимо проводить на многопроцессорных вычислительных комплексах IBM Blue Gene/P и «Ломоносов», используя различное количество вычислительных узлов, указанное в требованиях к отчету. Для каждого расчета определить его продолжительность и ускорение по сравнению с аналогичным расчетом на одном вычислительном узле. При распараллеливании программы необходимо использовать двумерное разбиение области на подобласти прямоугольной формы, в каждой из которых отношение θ количества узлов по ширине и длине должно удовлетворять неравенствам $0.5 \leq \theta \leq 2$.

Проделанная работа по созданию гибридных реализаций MPI/OpenMP и MPI/CUDA

MPI/OpenMP

Процессоры разбиваются на сетку $X \times Y$, где

$$X = \begin{cases} \sqrt{n}, & \text{если } \log_2 n - \text{четное} \\ \sqrt{\frac{n}{2}}, & \text{иначе} \end{cases}, Y = \begin{cases} X, & \text{если } \log_2 n - \text{четное} \\ 2 \times X, & \text{иначе} \end{cases}.$$

То есть если количество процессоров n является квадратом, то сетка квадратная, иначе она делится на две части по горизонтали, каждая из которых разбивается на квадратную. Далее, с помощью функции *MPI_Cart_create* создается виртуальная топология процессоров. Координаты процессора в топологии и номера соседних процессоров вычисляются с помощью функций *MPI_Cart_coords* и *MPI_Cart_shift* соответственно. Например, для $n = 8$ топология будет иметь следующий вид:

0 (0, 0)	1 (0, 1)	2 (0, 2)	3 (0, 3)
4 (1, 0)	5 (1, 1)	6 (1, 2)	7 (1, 3)

Точки распределяются на процессоры, и каждый процессор обрабатывает только свои точки. Так как количество точек по координате сетки может не делиться нацело на количество процессоров по соответствующей координате в топологии, то лишние точки распределяются на последний процессор по этой координате.

Таким образом, все процессоры, кроме нижней и правой границы, будут иметь одинаковое количество обрабатываемых точек.

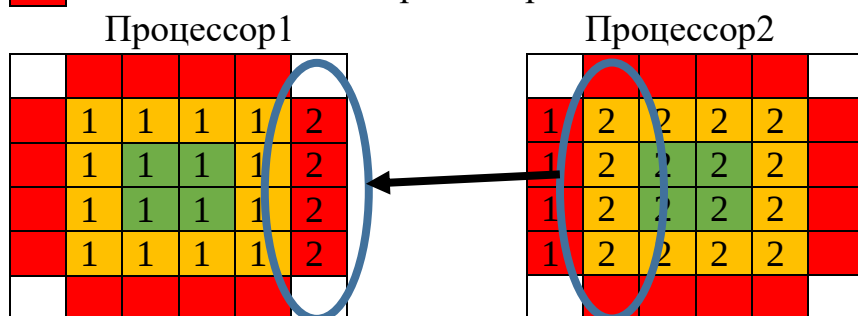
Например, для $n = 8, N_1 = N_2 = N = 8$ топология будет иметь следующий вид (81 точка):

4×2	4×2	4×2	4×3
5×2	5×2	5×2	5×3

После распределения каждый процессор вычисляет значения переменных из метода для своих точек. Итерации циклов в расчетах распределяются на потоки с помощью директивы OpenMP *parallel for*. Для подсчета частичного скалярного произведения используется директива *reduction* для суммы.

Для вычисления оператора Лапласа требуются данные с точек соседних процессоров. Для этого реализуется обмен между процессорами с помощью команд *MPI_Isend* и *MPI_Irecv* (так как все пересылки являются парными, то можно использовать асинхронные версии). Рассылаются только граничные точки:

N - внутренние точки процессора N
 - граничные точки
 - точки соседнего процессора



Последовательность обменов следующая:

1. получить точки у соседа снизу
2. получить точки у соседа справа
3. отправить точки соседу сверху
4. отправить точки соседу слева
5. отправить точки соседу снизу
6. отправить точки соседу справа
7. получить точки у соседа сверху
8. получить точки у соседа слева

Для топологии, указанной выше, получается следующая последовательность шагов, если рассматривать синхронные обмены:

0 (0:0)	1 (0:1)	2 (0:2)	3 (0:3)	4 (1:0)	5 (1:1)	6 (1:2)	7 (1:3)
.	.	.	RD7	.	.	.	SU3
.	.	.	.	.	.	RR7	SL6
.	.	RD6	.	.	.	SU2	.
.	.	RR3	SL2	.	RR6	SL5	.
.	RD5	.	SD7	.	SU1	.	RU3
.	RR2	SL1	RL2	RR5	SL4	SR7	RL6
RD4	.	SD6		SU0	.	RU2	
RR1	SL0	SR3		.	SR6	RL5	
.	SD5	RL1		.	RU1		
.	SR2			SR5	RL4		
SD4	RL0			RU0			
SR1							

XYN – send/receive, направление, номер процессора.

RD7 – получить точки снизу от процессора 7.

Чтобы посчитать итерационный параметр τ^{k+1} , коэффициент α и проверить условие остановки итерационного процесса необходимо вычислить полное скалярное произведение. Для этого, с помощью функции *MPI_Allreduce*, локальные частичные скалярные произведения суммируются и отправляются по всем процессорам.

MPI/CUDA

Для решения задачи на графических ускорителях использовалась библиотека Thrust (<http://docs.nvidia.com/cuda/thrust/index.html>). Так как на одном узле вычислялось два процесса, то ГПУ привязывался к процессу с помощью вызова *cudaSetDevice*: процессы с четным номером устанавливали первую видеокарту, с нечетным – вторую. Все директивы OpenMP были заменены. Во всех случаях, кроме редукции, использовалась функция *transform*. Для некоторых ее вызовов были реализованы соответствующие функторы. Для сокращения количества обходимых точек, где это было возможно, применялся *permutation_iterator*, в остальных случаях применялась комбинация *counting_iterator* для идентификации индекса в функторе и условие выхода из функтора без расчетов. Редукция осуществлялась с помощью вызова *transform_reduce*. Обмен граничными точками между GPU и CPU для передачи с помощью MPI и после их получения производился вызовом функции *copy*.

Результаты расчетов

Погрешность и проверка результатов

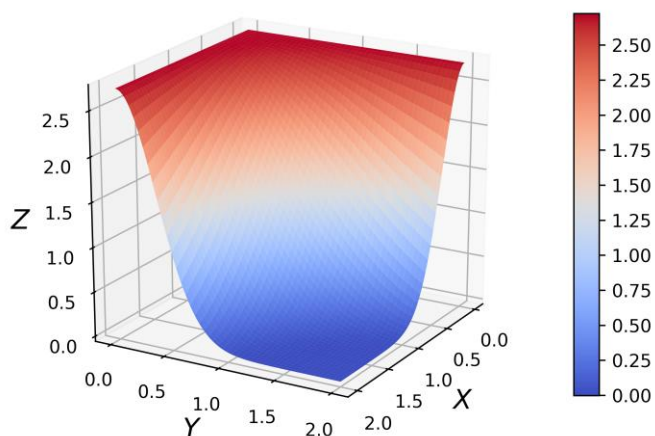
1500 x 1500: 0.011614

2000 x 2000: 0.011669

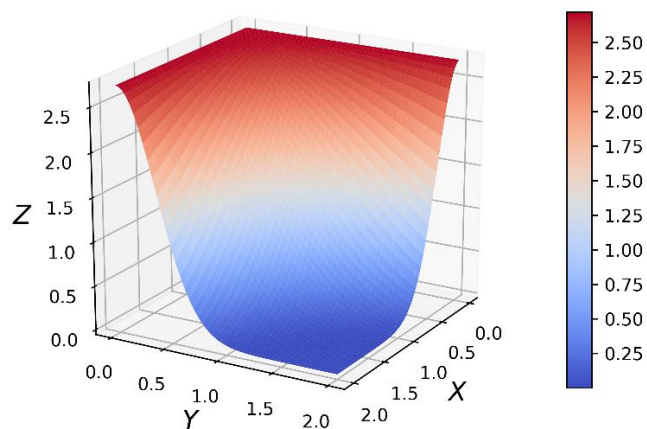
2500 x 2500: 0.011710

Пример проверки решения на сетке 2000 x 2000:

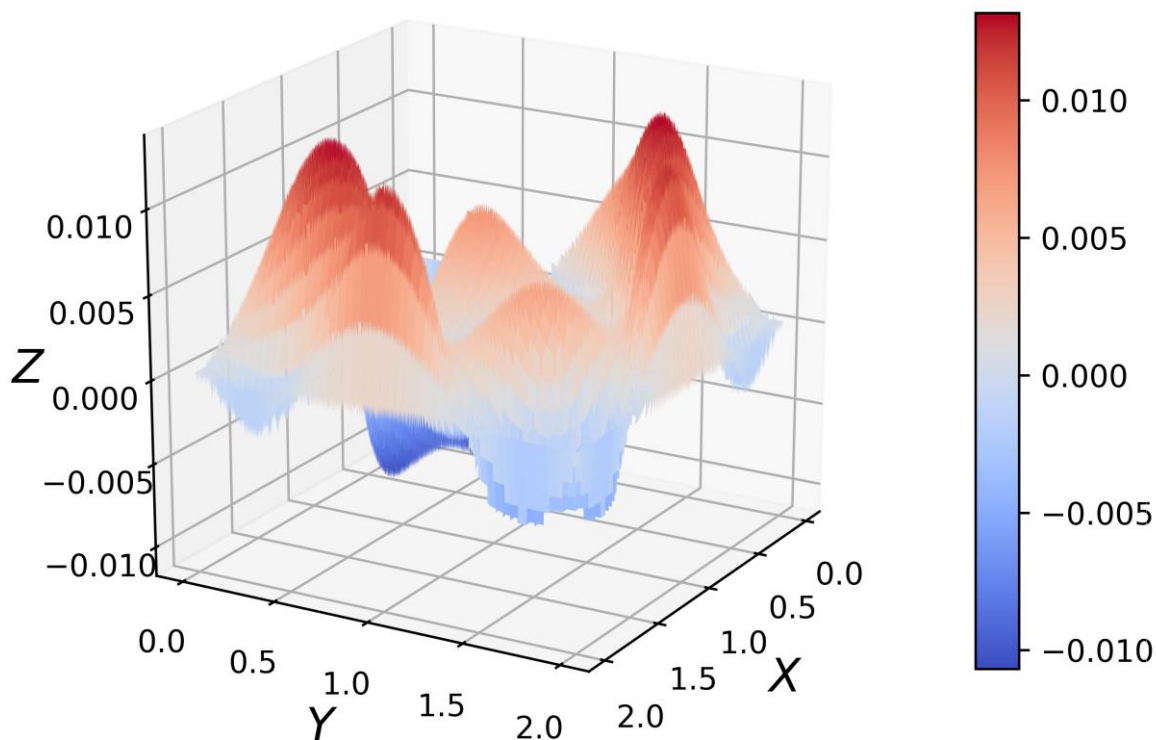
Приближенное решение p_{ij}



Точное решение $u(x_i, y_j)$



Погрешность $\psi = u(x_i, y_j) - p_{ij}$



Время и ускорение

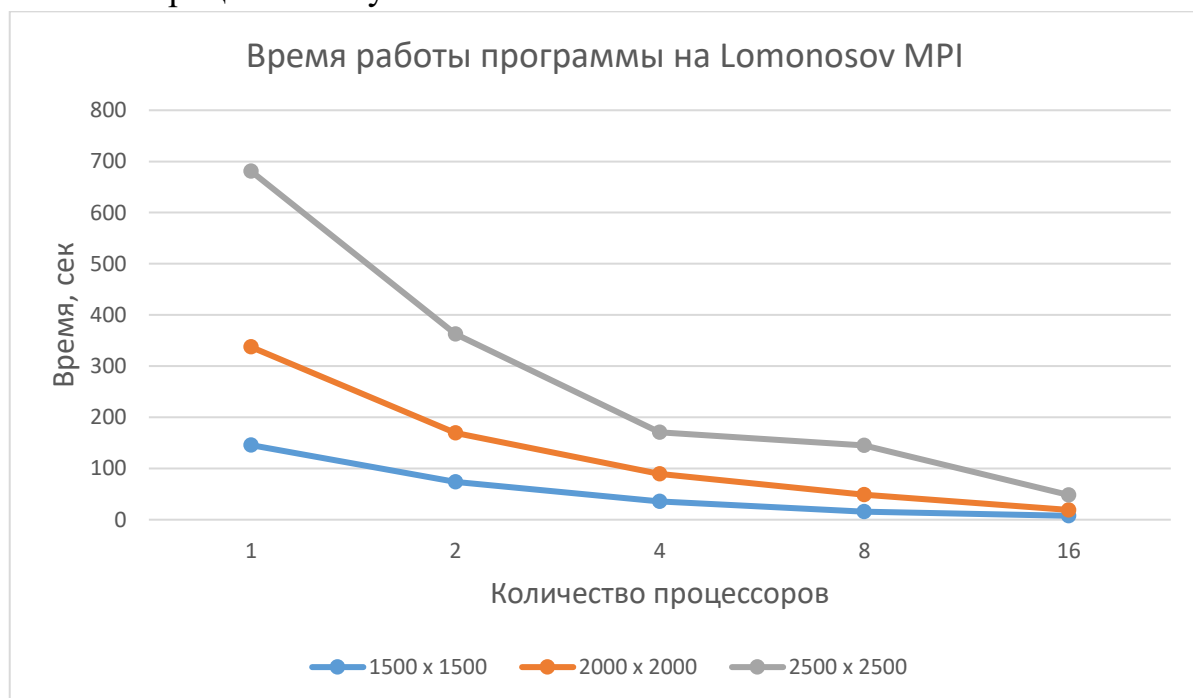
Время рассчитывалось как усредненное значение трех запусков.

$$\text{Ускорение } S = \frac{T_1}{T_p}$$

Lomonosov MPI

строка компиляции: `mpicxx -O3 task2.cpp -o task2`

- Очередь: test
- impi
- Количество процессов: 1, 2, 4, 8, 16
- Процессов на узел: 1



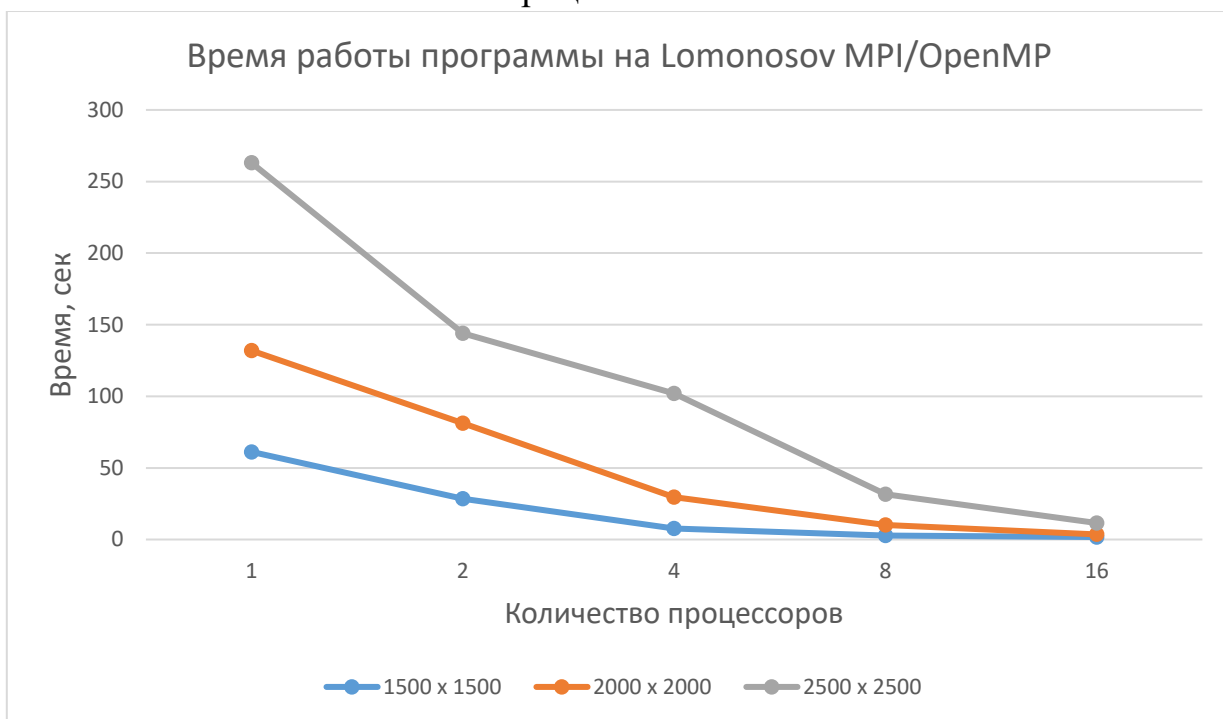
Число процессоров N_p	Число точек сетки N^3	Время решения T	Ускорение S
1	1500 x 1500	145.708	1.000
2	1500 x 1500	73.982	1.970
4	1500 x 1500	35.891	4.060
8	1500 x 1500	15.943	9.139
16	1500 x 1500	7.631	19.093
1	2000 x 2000	337.375	1.000
2	2000 x 2000	169.346	1.992
4	2000 x 2000	89.261	3.780
8	2000 x 2000	48.671	6.932
16	2000 x 2000	19.084	17.679

1	2500 x 2500	680.888	1.000
2	2500 x 2500	362.957	1.876
4	2500 x 2500	170.626	3.991
8	2500 x 2500	145.160	4.691
16	2500 x 2500	48.325	14.090

Lomonosov MPI/OpenMP

строка компиляции: `mpicxx -O3 -fopenmp task2.cpp -o task2_omp`

- Очередь: test
- `impi`
- Количество процессов: 1, 2, 4, 8, 16
- Процессов на узел: 1
- Количество ядер на процесс: 8
- Количество нитей на процесс: 16



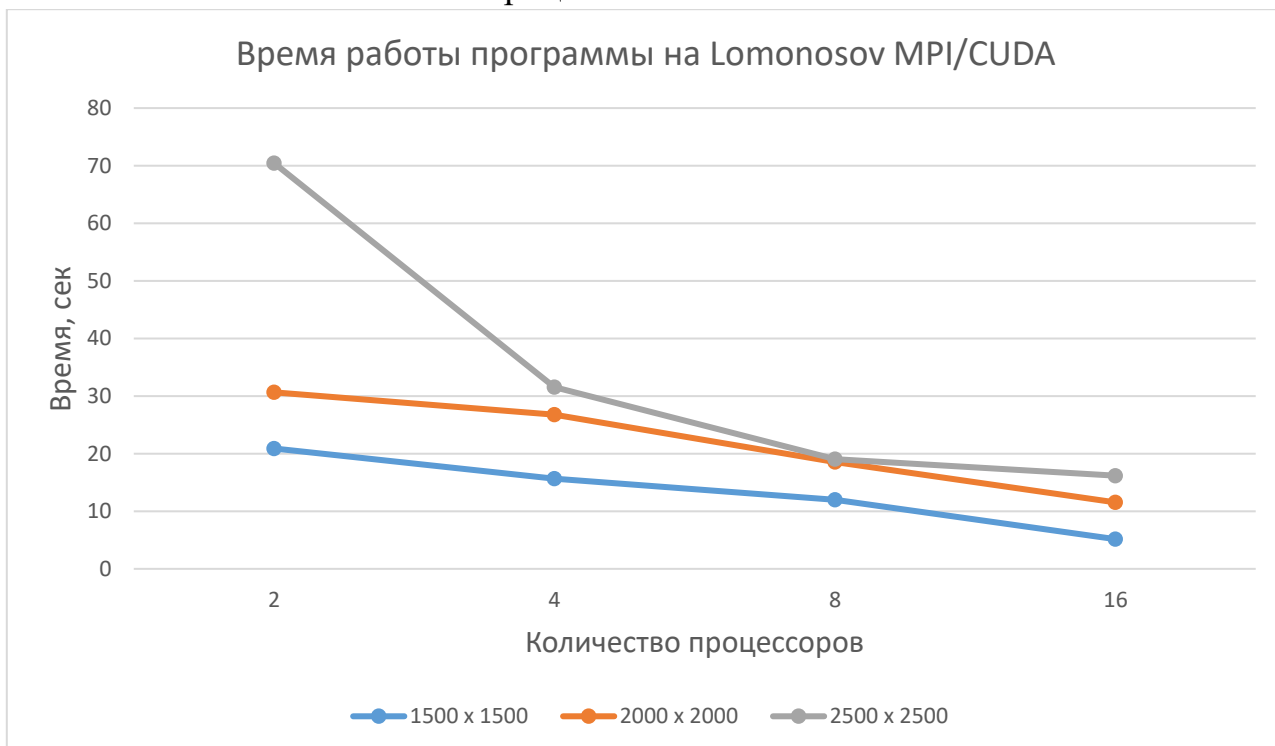
Число процессов N_p	Число точек сетки N^3	Время решения T	Ускорение S
1	1500 x 1500	61.188	2.381
2	1500 x 1500	28.486	5.115
4	1500 x 1500	7.578	19.227
8	1500 x 1500	2.797	52.092
16	1500 x 1500	1.723	84.587

1	2000 x 2000	131.903	2.558
2	2000 x 2000	81.212	4.154
4	2000 x 2000	29.573	11.408
8	2000 x 2000	10.129	33.307
16	2000 x 2000	3.551	95.017
1	2500 x 2500	263.000	2.589
2	2500 x 2500	143.972	4.729
4	2500 x 2500	102.064	6.671
8	2500 x 2500	31.622	21.532
16	2500 x 2500	11.507	59.171

Lomonosov MPI/CUDA

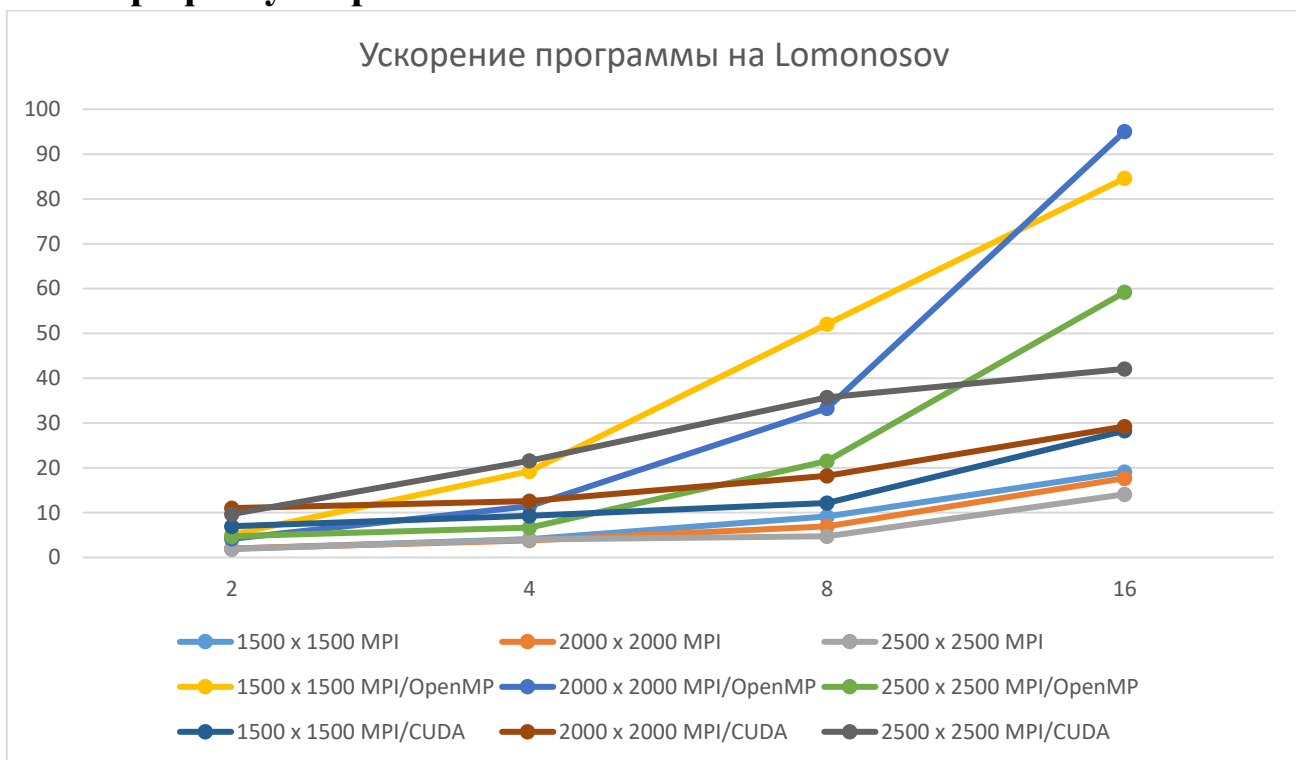
строка компиляции: `nvcc -arch=sm_20 -cubin mpicxx -o task3 -Xcompiler -O3 task3.cu`

- Очередь: `gputest`
- `impi`
- Количество процессов: 2, 4, 8, 16
- Процессов на узел: 2
- Количество ГПУ на процесс: 1



Число процессов N_p	Число точек сетки N^3	Время решения T	Ускорение S
2	1500 x 1500	20.901	6.971
4	1500 x 1500	15.64	9.316
8	1500 x 1500	11.996	12.146
16	1500 x 1500	5.156	28.260
2	2000 x 2000	30.649	11.008
4	2000 x 2000	26.757	12.609
8	2000 x 2000	18.549	18.188
16	2000 x 2000	11.542	29.230
2	2500 x 2500	70.46	9.663
4	2500 x 2500	31.559	21.575
8	2500 x 2500	19.054	35.735
16	2500 x 2500	16.182	42.077

График ускорений



Видно, что для программ MPI и MPI/CUDA (1500 x 1500) ускорение меняется незначительно до 8 процессоров и уже на 16 заметно возрастает. Это может быть связано с размером матрицы, так как при большем количестве

процессов, меньшее количество точек обрабатывается каждым процессом и графическим ускорителем. Также из-за небольшого порядка ($10^6 - 10^7$) количества точек MPI пересылки и обмен данными между CPU и GPU несут издержки, сопоставимые с временем подсчета. Графики для MPI/CUDA на сетке размером 2000 x 2000 и 2500 x 2500 выглядят более линейно. Это может быть связано с тем, что на сетке такого размера время расчетов превосходит накладные расходы. Для MPI/OpenMP программы ускорение примерно растет так же, как и количество процессов. В отличие от MPI/CUDA обмен не происходит, и задача распараллеливается практически без накладных расходов, разве что при редукции.