Отчёт: приближённое решение задачи Дирихле для уравнения Пуассона с использованием OpenMP, MPI и CUDA

[Сорокин Олег, 617]

7 декабря 2025 г.

Оглавление

1. Введение

В рамках задания требуется приближённо решить двумерную задачу Дирихле для уравнения Пуассона в криволинейной области с помощью метода фиктивных областей и некоторой разностной схемы. Для ускорения вычислений будут использоваться технологии OpenMP и MPI. С точки зрения математической физики задача состоит в нахождении функции $u(x, y)$, которая удовлетворяет уравнению Пуассона внутри заданной области и граничным условиям на её границе.

2. Математическая постановка задачи

Рассмотрим дифференциальное уравнение Пуассона в области $D \subset \mathbb{R}^2$, ограниченной кусочно-гладким контуром γ:

$$-\Delta u = f(x, y), \quad (x, y) \in D,$$

где $\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}$, а $f(x, y)$ — известная функция.

Граничное условие Дирихле:

$$u(x, y) = 0, \quad (x, y) \in \gamma.$$

В задании предлагается выбрать конкретные $f(x, y),\ D$, при подстановке которых в формулировку выше получаем следующую задачу:

$$\begin{cases} \Delta u = -1, & (x, y) \in D = \{(x, y) : y^2 < x < 1\}, \\ u(x, y) = 0, & (x, y) \in \gamma = \partial D. \end{cases}$$

3. Метод фиктивных областей

Для приближённого решения задачи используется метод фиктивных областей. Область D включается в прямоугольник Π. Разность множеств $\hat{D} = \Pi \setminus D$ называется фиктивной областью. Для рассматриваемой задачи целесообразно взять $\Pi = [0, 1] \times [-1, 1]$.

Вводится кусочно-постоянный коэффициент $k(x, y)$ и правая часть $F(x, y)$, которые позволяют свести задачу к решению уравнения в прямоугольнике с учётом граничных условий. В прямоугольнике Π рассматривается задача Дирихле:

$$-\frac{\partial}{\partial x}\left(k(x, y)\frac{\partial v}{\partial x}\right) - \frac{\partial}{\partial y}\left(k(x, y)\frac{\partial v}{\partial y}\right) = F(x, y), \quad (x, y) \in \Pi \setminus \gamma,$$

с кусочно-постоянным коэффициентом:

$$k(x,y) = \begin{cases} 1, & (x,y) \in D, \\ 1/\varepsilon, & (x,y) \in \hat{D}, \end{cases}$$

и правой частью:

$$F(x,y) = \begin{cases} f(x,y), & (x,y) \in D, \\ 0, & (x,y) \in \hat{D}. \end{cases}$$

4. Численный метод решения задачи

Для численного решения задачи применяется метод конечных разностей. В замыкании прямоугольника Π определяется равномерная прямоугольная сетка $\omega_h = \omega_1 \times \omega_2$, где $\omega_1 = \{x_i = A_1 + ih_1, i = 0, M\}$, $\omega_2 = \{y_j = A_2 + jh_2, j = 0, N\}$. Здесь $h_1 = (B_1 - A_1)/M$, $h_2 = (B_2 - A_2)/N$.

Дифференциальное уравнение аппроксимируется разностным уравнением, связывающим значения искомой функции в узлах сетки:

$$-\frac{1}{h_1}\left(a_{(i+1)j}\frac{w_{(i+1)j} - w_{ij}}{h_1} - a_{ij}\frac{w_{ij} - w_{(i-1)j}}{h_1}\right) - \frac{1}{h_2}\left(b_{i(j+1)}\frac{w_{i(j+1)} - w_{ij}}{h_2} - b_{ij}\frac{w_{ij} - w_{i(j-1)}}{h_2}\right) = F_{ij}$$

где коэффициенты a_{ij} и b_{ij} вычисляются следующим образом:

$$a_{ij} = \frac{1}{h_2}\int_{y_{j-1/2}}^{y_{j+1/2}} k(x_{i-1/2}, t)\,dt, \quad b_{ij} = \frac{1}{h_1}\int_{x_{i-1/2}}^{x_{i+1/2}} k(t, y_{j-1/2})\,dt.$$

Заметим, что если отрезок, соединяющий точки $P_{ij} = (x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}})$ и $P_{i(j+1)} = (x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}})$, целиком расположен в области D, то $a_{ij} = 1$. Если же указанный отрезок находится в фиктивной области $\hat{D}$, то $a_{ij} = \frac{1}{\varepsilon}$. В противном случае

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

где l_{ij} – длина той части отрезка $[P_{ij}, P_{i(j+1)}]$, которая принадлежит области D. Аналогичным образом вычисляются коэффициенты b_{ij}.

Правая часть разностного уравнения:

$$F_{ij} = \frac{1}{h_1 h_2}\int_{\Pi_{ij}} F(x,y)\,dx\,dy,$$

где $\Pi_{ij} = \{(x,y) : x_{i-1/2} \le x \le x_{i+1/2}, y_{j-1/2} \le y \le y_{j+1/2}\}$.

Очевидно, правая часть схемы F_{ij} равна нулю при всех $(i,j) : \Pi_{ij} \subset \hat{D}$. Если $\Pi_{ij} \subset D$, то правую часть предлагается приближенно заменить значением $f(x_i, y_j)$. В противном случае, когда прямоугольник Π_{ij} содержит точки оригинальной области D и фиктивной области $\hat{D}$, величина F_{ij} может быть вычислена приближенно как

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

где (x_i^*, y_j^*) – любая точка пересечения $\Pi_{ij} \cap D, S_{ij} = mes(\Pi_{ij} \cap D)$ – площадь пересечения множеств. Учитывая вид нашей задачи и заменяя подсчёт S_{ij} численным интегрированием, получаем общую формулу

$$F_{ij} = \frac{\hat{S}_{ij}}{h_1 h_2}$$

При подстановке конкретных $a_{ij},\ b_{ij},\ F_{ij}$ в разностную схему получаем систему линейных уравнений с трёхдиагональной матрицей. Для решения таких систем используется итерационный метод сопряжённых градиентов с диагональным предобуславливанием.

Пусть оператор $D : H \to H$ действует на сеточные функции $w \in H$ по правилу:

$$(Dw)_{ij} = \frac{(a_{i+1j} + a_{ij})}{h_1^2} w_{ij} + \frac{(b_{ij+1} + b_{ij})}{h_2^2} w_{ij}, \quad i = 1, M-1, j = 1, N-1.$$

Начальное приближение $w^{(0)}$ к решению разностной схемы можно выбрать равным нулю во всех точках расчётной сетки. Первая итерация совершается по формулам метода скорейшего спуска. Пусть $r^{(0)} = B - Aw^{(0)}$ — невязка начального приближения, функция $z^{(0)} \in H$ удовлетворяет уравнению $Dz^{(0)} = r^{(0)}$. Тогда направление спуска $p^{(1)} = z^{(0)}$, шаг вдоль направления спуска определяется параметром:

$$\alpha_1 = \frac{(r^{(0)}, z^{(0)})}{(Ap^{(1)}, p^{(1)})}.$$

Следующее приближение $w^{(1)}$ вычисляется согласно равенству:

$$w^{(1)} = w^{(0)} + \alpha_1 p^{(1)}.$$

Дальнейшие вычисления проводятся по следующим формулам. Пусть выполнено k итераций метода и функции $r^{(k-1)}$, $p^{(k)}$, $w^{(k)} \in H$, а также коэффициент α_k являются известными. Тогда невязка последней итерации:

$$r^{(k)} = r^{(k-1)} - \alpha_k A p^{(k)},$$

сеточная функция $z^{(k)} \in H$ вычисляется из уравнения $Dz^{(k)} = r^{(k)}$. Следующее направление спуска:

$$p^{(k+1)} = z^{(k)} + \beta_{k+1} p^{(k)},$$

где коэффициент:

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

Шаг спуска определяется параметром:

$$\alpha_{k+1} = \frac{(r^{(k)}, z^{(k)})}{(Ap^{(k+1)}, p^{(k+1)})}.$$

Следующее приближение к точному решению $w^{(k+1)}$ вычисляется согласно равенству:

$$w^{(k+1)} = w^{(k)} + \alpha_{k+1} p^{(k+1)}.$$

При работе метода важно отслеживать монотонность убывания функционала $J(w) = 0.5(Aw, w) - (B, w)$, и перезапускать метод с последней успешной итерации в случае нарушения этой монотонности. Отметим, что при исполении готовой программы ни разу такого нарушения замечено не было.

5. Использование OpenMP, MPI и CUDA

Сначала на языке C++ стандарта 2011 года была реализована последовательная программа, и на ней производились расчёты для небольших сеток для проверки. Для проверки корректности работы параллельным программ полученные ими решения поэлементно сравниваются с решением последовательной программы. Также критериями для параллельных программ являются ускорение и эффективность относительно исходной версии. В случае MPI+CUDA ожидается резкий рост ускорения за счёт перехода на графические процессоры даже в случае одного MPI процесса.

Для первой версии параллельной программы в последовательную версию добавлены прагмы OpenMP. Далее проводились эксперименты с MPI. При это множество узлов разбивается на подобласти так, чтобы в полученных прямоугольниках отношение длин сторон принадлежало бы отрезку $[1/2, 2]$. Дополнительное ограничение состоит в том, чтобы количество узлов по переменным в разных подобластях не различалось более, чем на 1. Полученные области используются для распараллеливания вычислений по процессам MPI.

Для дальнейшего ускорения вычислений вновь применяется технология OpenMP, таким образом получаем гибридную MPI + OpenMP программу. В параллельных нитях происходят: вычисление коэффициентов разностной схемы; вычисление правой части разностной схемы; части итерационного процесса метода сопряжённых градиентов.

Наиболее ресурсоёмкой частью алгоритма является решение системы линейных уравнений методом сопряжённых градиентов. Распараллеливание по главному циклу метода не представляется возможным, вместо этого параллельно вычисляются скалярные произведения, невязка и другие обновляемые параметры. Вычисление значений a_{ij}, b_{ij}, F_{ij} также было распараллелено, но в общее ускорение это вносит очень малый вклад (доли процента).

Также проводились эксперименты с MPI-версией, использующей CUDA для получения ускорения. При этом каждой GPU соответствует свой (единственный) MPI процесс.

6. Результаты

6.1 Ускорение от роста числа нитей OpenMP

Количество OpenMP нитей	Число точек сетки ($M \times N$)	Число итераций	Время решения	Ускорение
2	400×600	766	1.44	3.40
4	400×600	766	1.26	3.89
8	400×600	766	0.68	7.21
16	400×600	766	0.37	13.24
4	800×1200	1482	8.16	5.11
8	800×1200	1482	5.56	7.49
16	800×1200	1482	2.73	15.27
32	800×1200	1482	3.13	13.32

Таблица 1: Ускорение вычислений с использованием OpenMP

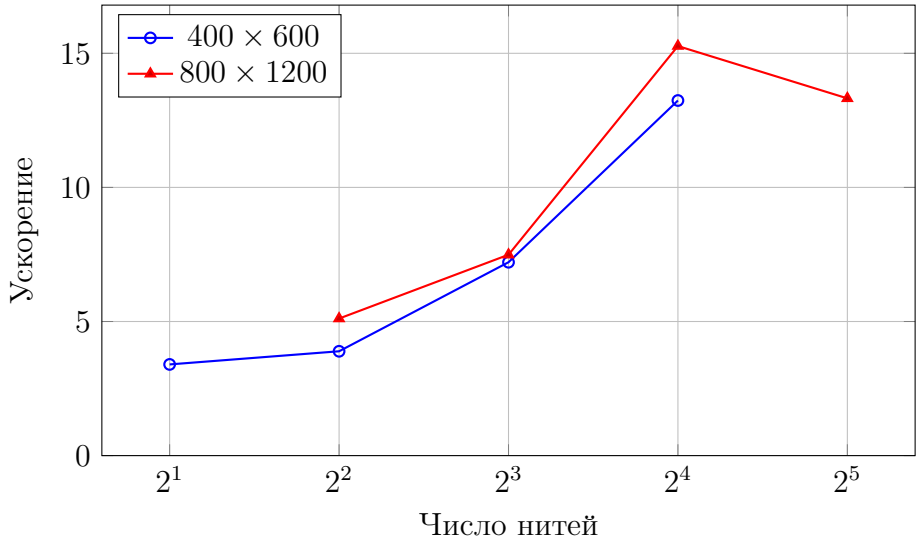

Рис. 1: Ускорение в зависимости от числа нитей

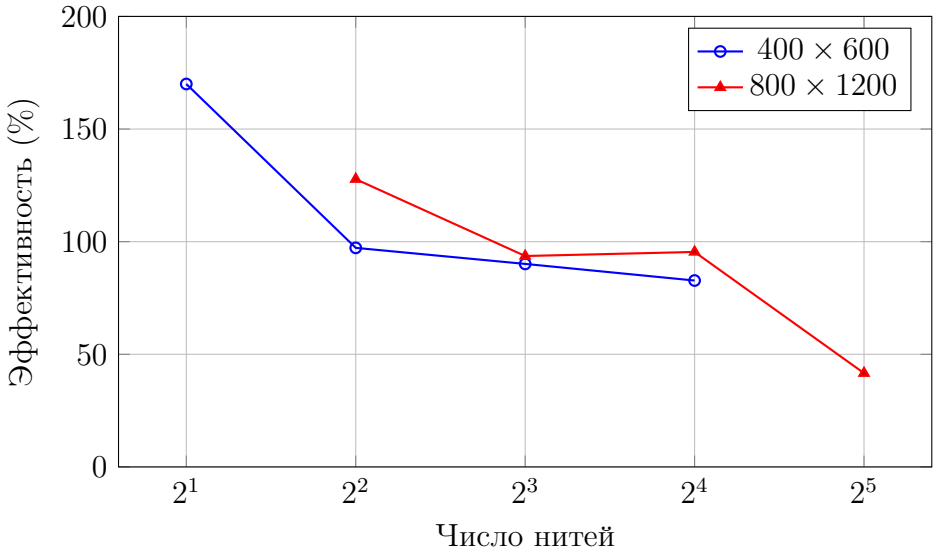

Рис. 2: Эффективность в зависимости от числа OpenMP тредов

Результаты для этой серии экспериментов ожидаемые: с ростом числа OpenMP нитей видим ускорение программы. Единственный выпадающий из закономерности случай – запуск на 32 нитях для сетки 800×1200. Предполагаю, что это связано с требованием привязки нити к ядру, так как последний запуск делался из-за ограничений Polus с аргументом "#BSUB -R affinity[core(20)]".

6.2 Ускорение от роста числа MPI процессов

Количество процессов MPI	Число точек сетки ($M \times N$)	Число итераций	Время решения	Ускорение
2	400×600	766	1.45	3.38
4	400×600	766	1.18	4.16
8	400×600	766	0.82	5.99
16	400×600	766	0.31	15.83
4	800×1200	1482	7.09	5.88
8	800×1200	1482	5.79	7.20
16	800×1200	1482	3.51	11.88
32	800×1200	1482	1.52	27.42

Таблица 2: Ускорение вычислений с использованием MPI

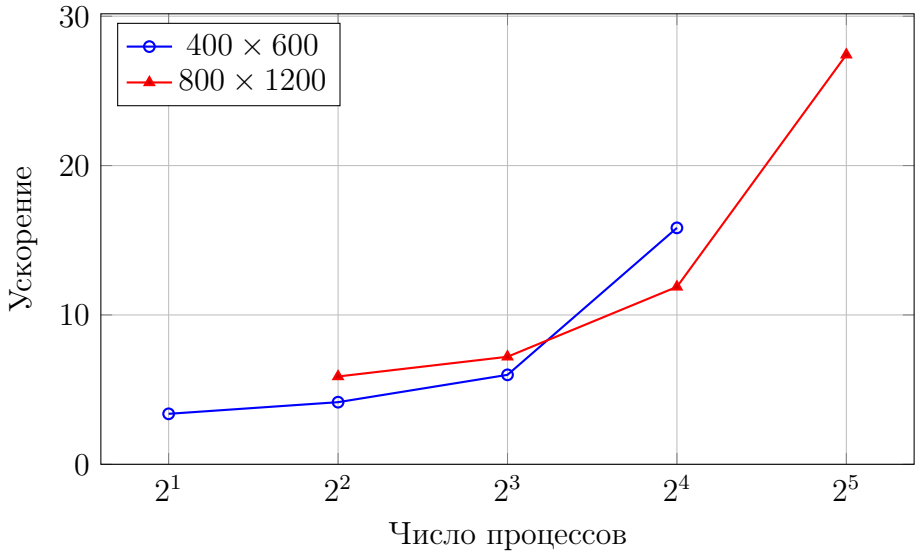

Рис. 3: Ускорение в зависимости от числа процессов

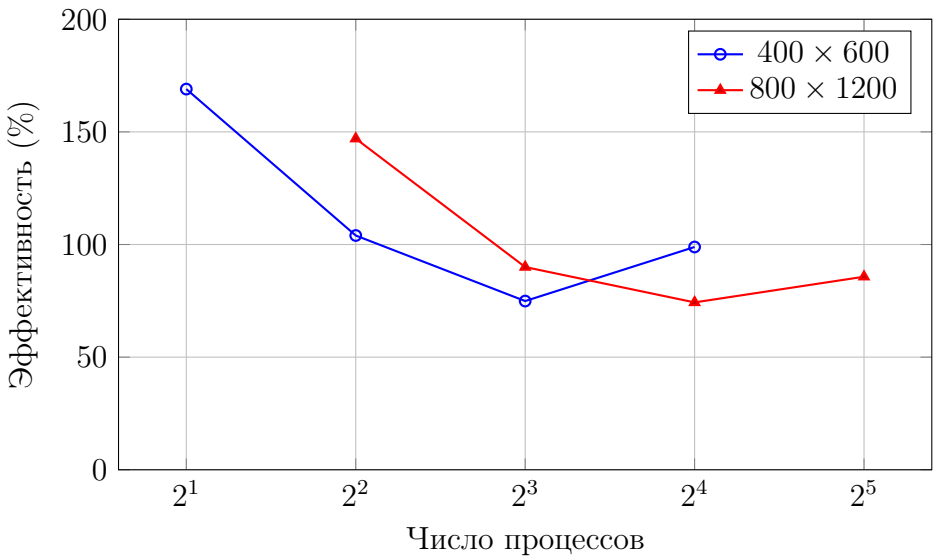

Рис. 4: Эффективность в зависимости от числа процессов

Результаты этой серии экспериментов совпадают с прогнозом: получили окололинейную зависимость ускорения от числа MPI процессов на обоих размерах задачи. Для всех запусков имеем хорошие показатели эффективности.

6.3 Ускорение от роста числа нитей на процесс MPI

Количество MPI процессов	Количество OpenMP нитей	Число точек сетки ($M \times N$)	Число итераций	Время решения	Ускорение
2	1	400×600	766	1.45	3.38
2	2	400×600	766	1.26	3.89
2	4	400×600	766	0.79	6.21
2	8	400×600	766	0.51	9.62
4	1	800×1200	1482	7.09	5.88
4	2	800×1200	1482	5.71	7.30
4	4	800×1200	1482	3.13	13.30
4	8	800×1200	1482	1.52	27.42

Таблица 3: Ускорение вычислений с использованием MPI

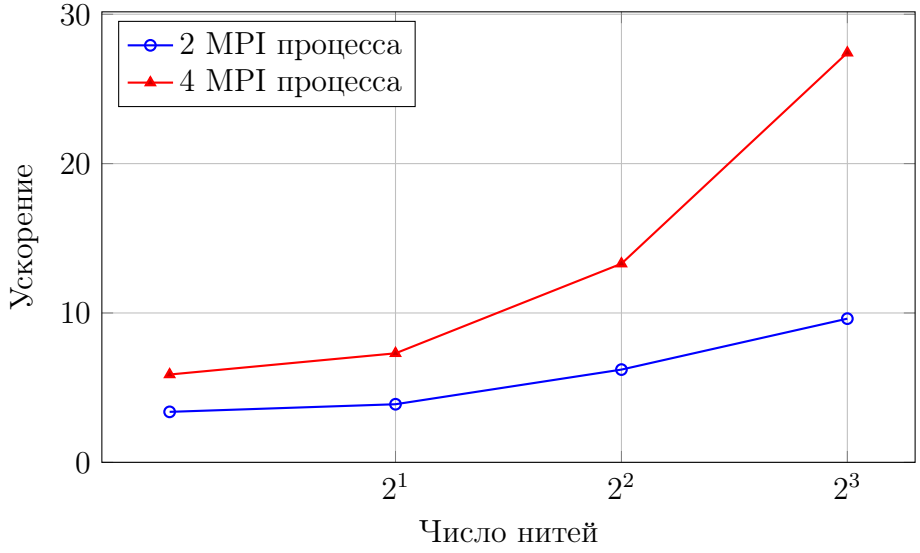

Рис. 5: Ускорение в зависимости от числа OpenMP тредов

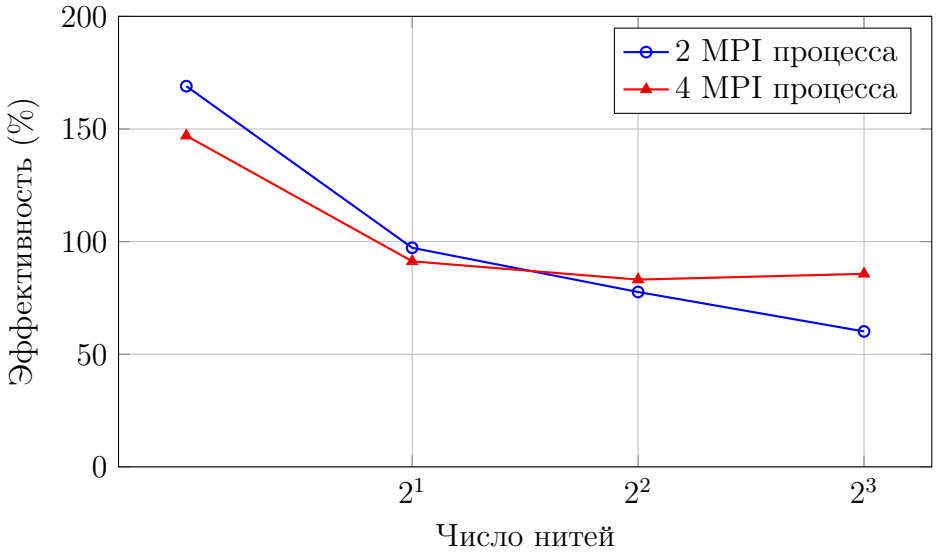

Рис. 6: Эффективность в зависимости от числа OpenMP тредов

В этой серии эксперимнетов снова получили результат, близкий к прогнозу: с ростом числа нитей на один MPI процесс в теории должны наблюдать ускорение, близкое к линейному.

6.4 Ускорение от MPI + CUDA

Эксперименты ниже проводятся на ресурсах одного узла Polus (1-2 GPU). Размер сетки выбран так, чтобы видеть хорошее ускорение, и равен 800×1200.

	Последовательная	1 GPU	2 GPU
Число итераций	1482	1482	1482
Время инициализации	0.03	0.03	0.02
Время коммуникаций и обменов	–	0.11	0.12
Время параллельных циклов	46.67	2.56	1.71
Общее время	46.77	2.65	1.86
Ускорение	–	18.27	27.35

Таблица 4: Результаты экспериментов с MPI + CUDA

По результатам выше можно сделать следующие выводы:

- Число итераций и время инициализации параметров системы для всех запусков практически совпадает. Последнее связано с тем, что инициализация составляет доли процента общего времени выполнения.
- Время коммуникаций и обменов для 1 и 2 GPU отличается незначительно. Скорее всего, это связано с тем что доминируют не сами коммуникации и обмены, а константные накладные расходы, связанные с ними. Например, речь идёт о минимальном времени требуемом для cudaMemcpy.
- В случае одной GPU уже получаем сильное ускорение относительно последовательной программы.
- С двумя GPU получаем ускорение относительно одной. Скорее всего, если увеличить размер задачи, эффективность в этом случае вырастет.

6.5 Визуализация полученного решения

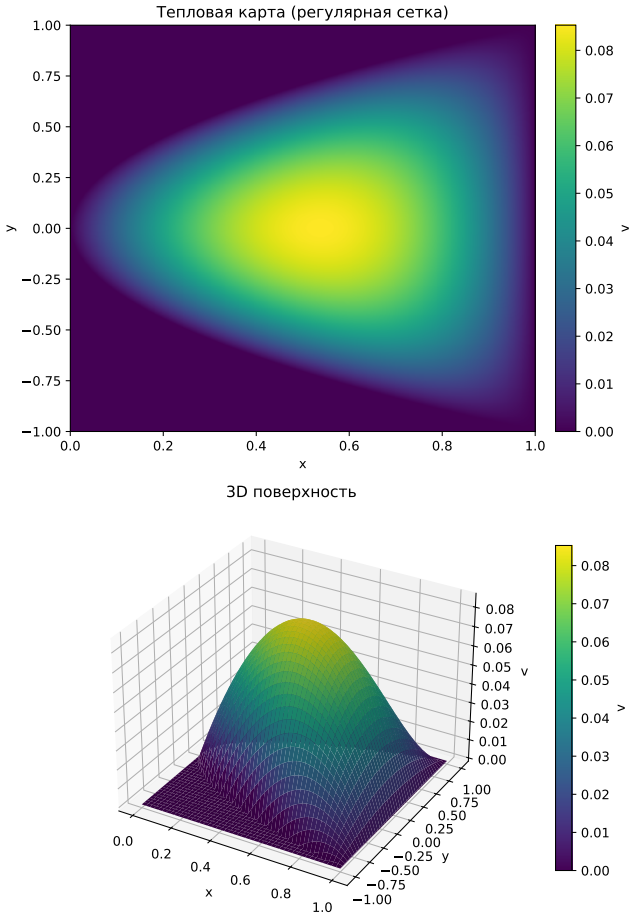

Рис. 7: Решение на сетке 800×1200

7. Заключение

В ходе выполнения задания рассмотрен один из методов численного решения двумерной задачи Дирихле для уравнения Пуассона. На языке C++ реализована последовательная программа, а также её OpenMP, MPI, гибридная MPI + OpenMP и MPI + CUDA версии. Произведено сравнение последовательной и параллельной программ. Исследована зависимость времени выполнения, ускорения и эффективности в зависимости от числа OpenMP нитей, MPI процессов, OpenMP тредов при фиксированном числе MPI процессов, а также числе GPU (в предположении 1 MPI процесс / 1 GPU). По каждому из пунктов сделаны выводы

о том, совпадает ли результат с прогнозом. Получена визуализация решения для одной из сеток. Все эксперименты проведены на кластере Polus.