Отчет по заданию №2 «Исследование численного решения задачи Дирихле для уравнения Пуассона в прямоугольной области» по курсу «Суперкомпьютерное моделирование и технологии».

Асирян Александр, 624 группа. Вариант 2\_23.

**Математическая постановка задачи**

В прямоугольной области требуется найти дважды гладкую функцию , удовлетворяющую дифференциальному уравнению:

и дополнительному условию:

во всех граничных точках прямоугольника. Оператор Лапласа определен равенством:

Вариант 2\_23:



**Численный метод решения задачи**

В расчетной области определяется прямоугольная сетка

где – разбиение отрезка оси (𝑜𝑥),

– разбиение отрезка оси (𝑜𝑦).

Через обозначим множество внутренних, а через – множество граничных узлов сетки . Пусть – переменный шаг сетки по оси абсцисс и ординат соответственно. Средние шаги сетки определяются равенствами:

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

где – любые функции из пространства 𝐻.

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

Здесь предполагается, что функция определена во всех узлах сетки .

Приближенным решением задачи (1), (2) называется функция , удовлетворяющая уравнениям

Эти соотношения представляют собой систему линейных алгебраических уравнений с числом уравнений равным числу неизвестных и определяют единственным образом неизвестные значения . 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

во внутренних узлах сетки – любые числа. Метод является одношаговым. Итерация вычисляется по итерации согласно равенствам:

где невязка

Итерационный параметр

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

Существенно большей скоростью сходимости обладает метод сопряженных градиентов. Начальное приближение и первая итерация вычисляются так же, как и в методе скорейшего спуска. Последующие итерации осуществляются по формулам:

Здесь

вектор

коэффициент

Вектор невязки вычисляется согласно равенствам (5). Итерационный процесс останавливается, как только

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

Для аппроксимации дифференциальной задачи используется равномерная прямоугольная сетка:

Приближенное решение разностной схемы (4) следует вычислять методом сопряженных градиентов. Для остановки итерационного процесса предлагается использовать условие (6), положив .

**Постановка задачи**

Для функций :

* подобрать точное решение задачи Дирихле,
* методом сопряженных градиентов построить приближенное решение на сетке с числом узлов , определить погрешность решения
* методом сопряженных градиентов построить приближенное решение на сетке с числом узлов 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 .

**Проделанная работа по созданию гибридной реализации MPI/OpenMP**

Процессоры разбиваются на сетку , где

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

То есть если количество процессоров 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\_coor*ds и *MPI\_Cart\_shift* соответственно. Например, для топология будет иметь следующий вид:

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 топология будет иметь следующий вид (81 точка):

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |

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*. Вложенные циклы не распараллеливаются, потому что версия OpenMP не поддерживает директиву *collapse*. 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

Процессор1 Процессор2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
|  | 1 | 1 | 1 | 1 | 2 |
|  | 1 | 1 | 1 | 1 | 2 |
|  | 1 | 1 | 1 | 1 | 2 |
|  | 1 | 1 | 1 | 1 | 2 |
|  |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |
| 1 | 2 | 2 | 2 | 2 |  |
| 1 | 2 | 2 | 2 | 2 |  |
| 1 | 2 | 2 | 2 | 2 |  |
| 1 | 2 | 2 | 2 | 2 |  |
|  |  |  |  |  |  |

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, получаются 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 , коэффициент и проверить условие остановки итерационного процесса необходимо вычислить полное скалярное произведение. Для этого, с помощью функции *MPI\_Allreduce*, локальные частичные скалярные произведение суммируются и отправляются по всем процессорам.

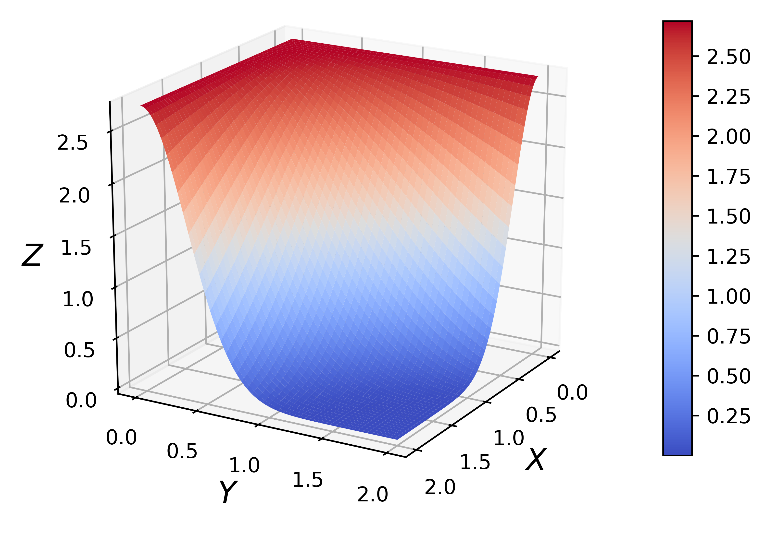
**Результаты расчетов**

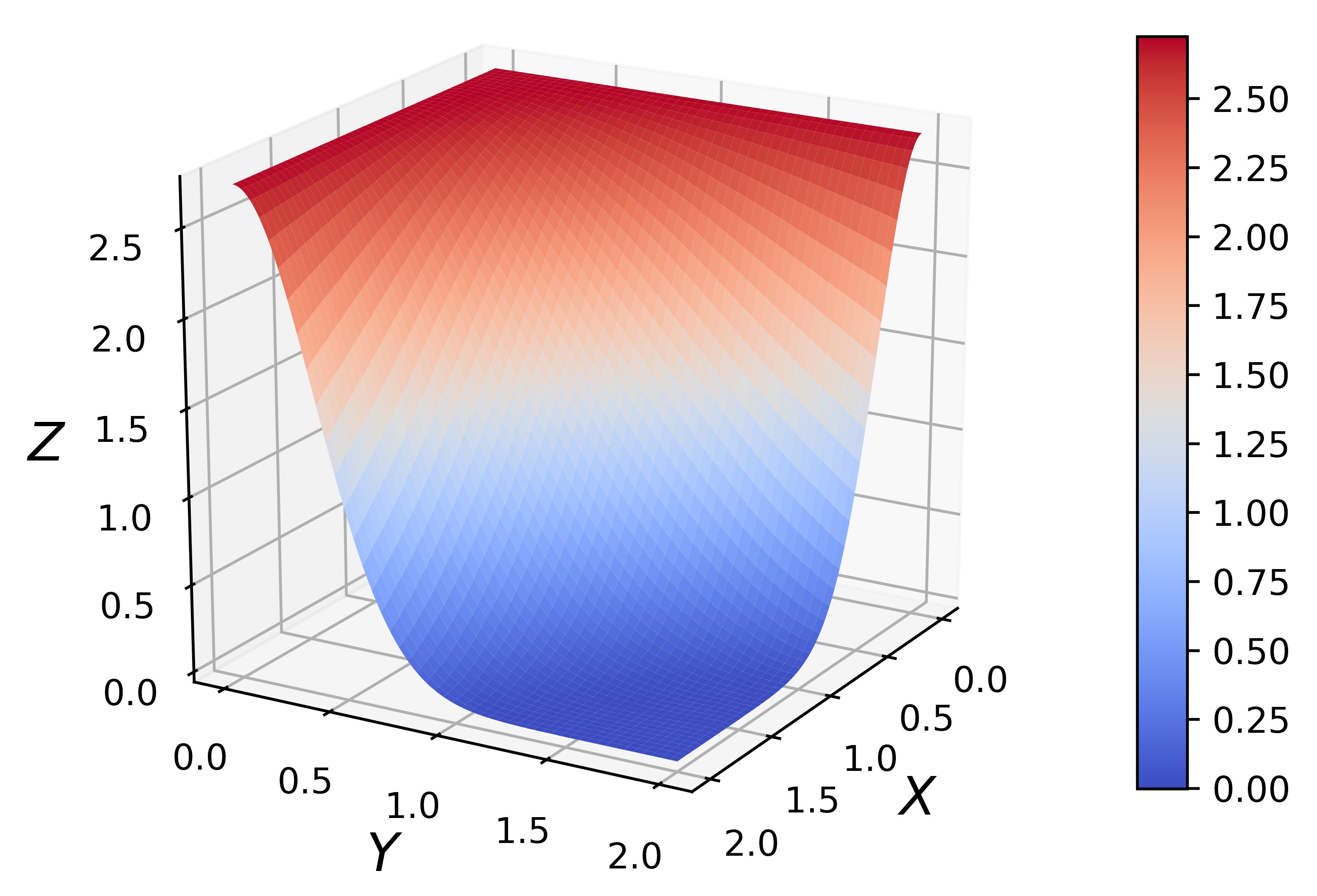
**Погрешность**

1000 х 1000: 0.011485

2000 х 2000: 0.011669

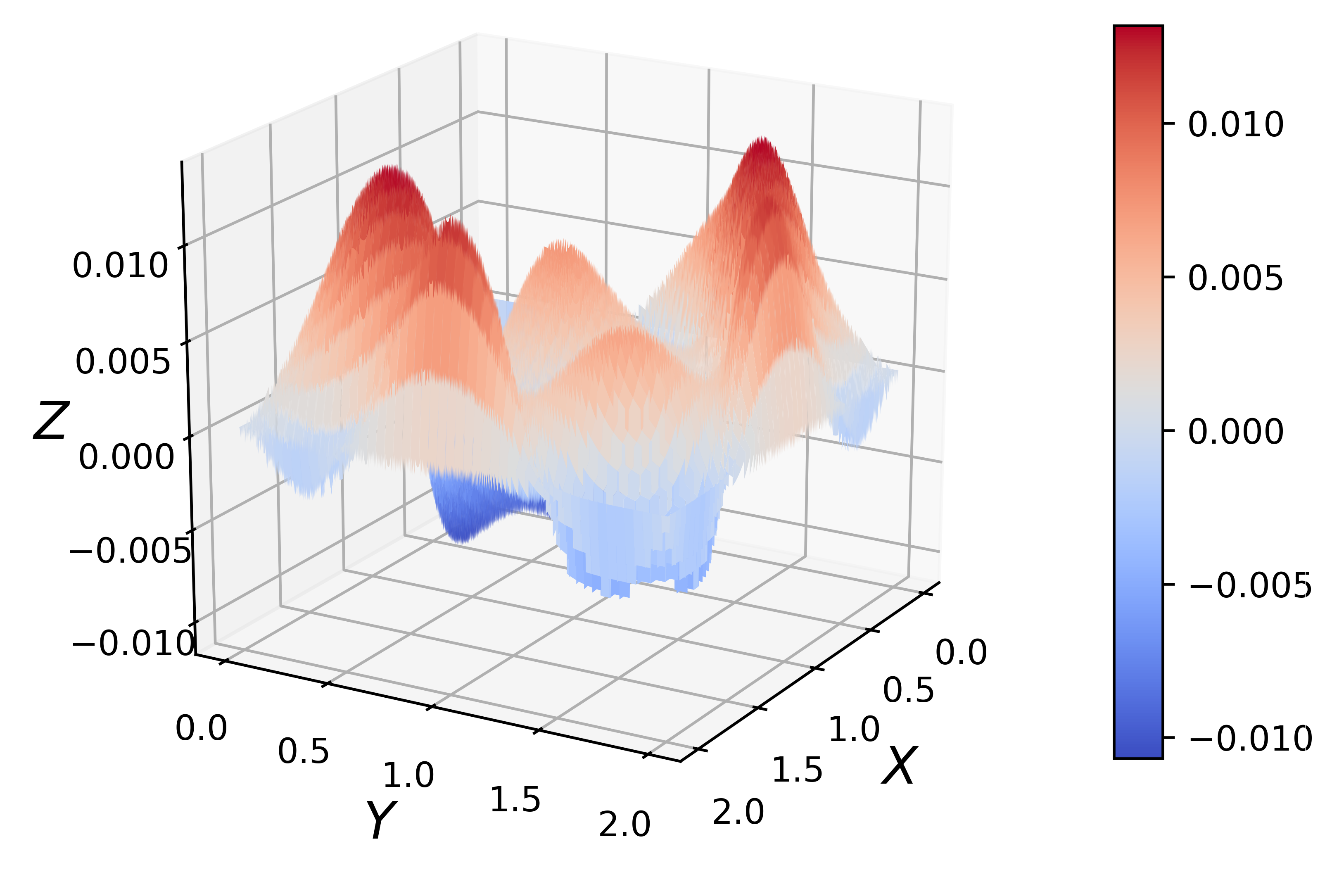
Приближенное решение Точное решение





Погрешность

**Время и ускорение**



Время рассчитывалось как усредненное значение трех запусков.

Ускорение .

**Blue Gene/P**

**MPI:**

строка компиляции: mpixlcxx -O3 task2.cpp -o task2

|  |  |  |  |
| --- | --- | --- | --- |
| Число процессоров | Число точек сетки | Время решения 𝑇 | Ускорение 𝑆 |
| 1 | 1000 x 1000 | 446.785 | 1.000000 |
| 128 | 1000 x 1000 | 4.400 | 101.547951 |
| 256 | 1000 x 1000 | 2.554 | 174.959449 |
| 512 | 1000 x 1000 | 1.483 | 301.209406 |
| 1 | 2000 x 2000 | 3991.830 | 1.000000 |
| 128 | 2000 x 2000 | 30.072 | 132.742372 |
| 256 | 2000 x 2000 | 15.265 | 261.497294 |
| 512 | 2000 x 2000 | 9.712 | 411.013497 |

**MPI/OpenMP:**

строка компиляции: mpixlcxx\_r -O3 -qsmp=omp task2.cpp -o task2\_omp

|  |  |  |  |
| --- | --- | --- | --- |
| Число процессоров | Число точек сетки | Время решения 𝑇 | Ускорение 𝑆 |
| 1 | 1000 x 1000 | 152.835 | 2,923316 |
| 128 | 1000 x 1000 | 2.309 | 193,4858 |
| 256 | 1000 x 1000 | 1.685 | 265,1884 |
| 512 | 1000 x 1000 | 1.370 | 326,0479 |
| 1 | 2000 x 2000 | 1359.220 | 2,936854 |
| 128 | 2000 x 2000 | 12.330 | 323,7621 |
| 256 | 2000 x 2000 | 7.126 | 560,14 |
| 512 | 2000 x 2000 | 4.907 | 813,5447 |

**Lomonosov**

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

|  |  |  |  |
| --- | --- | --- | --- |
| Число процессоров | Число точек сетки | Время решения 𝑇 | Ускорение 𝑆 |
| 1 | 1000 x 1000 | 40.801 | 1.000000 |
| 8 | 1000 x 1000 | 11.539 | 3.536073 |
| 16 | 1000 x 1000 | 4.093 | 9.967780 |
| 32 | 1000 x 1000 | 1.518 | 26.881058 |
| 128 | 1000 x 1000 | 1.102 | 37.039683 |
| 1 | 2000 x 2000 | 318.373 | 1.000000 |
| 8 | 2000 x 2000 | 108.983 | 2.921315 |
| 16 | 2000 x 2000 | 56.529 | 5.632065 |
| 32 | 2000 x 2000 | 23.965 | 13.285135 |
| 128 | 2000 x 2000 | 3.818 | 83.391712 |