Курс «Суперкомпьютерное моделирование и технологии»Отчет по заданию №2 Вариант 4

Ван Син, студент 614 группы,второй поток

Факеультет вычислительной математики и кибернетики
МГУ им. М. В. Ломоносова
Москва, 2024 год

1 Введение

Требуется приближенно решить задачу Дирихле для уравнения Пуассона в криволинейной области . Задание необходимо выполнить на ПВС Московского университета IBM Polus.

2 Математическая постановка задачи

В области $D \subset R^2$, ограниченной контуром γ , рассматривается дифференциальное уравнение Пуассона.

$$-\triangle u = f(x, y), \tag{2.1}$$

в котором оператор Лапласа:

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

функция f(x,y) считается известной. Для выделения единственного решения уравнение дополняется граничными условием Дирихле:

$$u(x, y) = 0, \ (x, y) \in \gamma \tag{2.2}$$

Требуется найти функцию u(x,y), удовлетворяющую уравнению (2.1) в области D и краевому условию (2.2) на ее границе.

3 Численные методы решения

Метод фиктивных областей и т.д. Как указано в документе "Задание по курсу «Суперкомпьютерное моделирование и технологии»".

4 Программная реализация

Программа вводит M,N, точность и количество потоков (для openMP) в главную функцию main и постоянно вызывает функцию kernel в main для завершения решения задачи.

Функция **kernel** инициализирует координаты, соответствующие каждой точке (i,j), и предполагает, что исходное решение w, которое не подвергалось итерации,

является нулевой матрицей. Функция вычисляет соответствующую систему линей- ных уравнений Aw=F постоянный член F_{ij} и затем выполняет итерации в соответ- ствии с соответствующим численным методом. Для реализации вычитания между

матрицами, а также умножения констант на матрицы и матриц на матрицы в про- цессе итерации используются функция **MM_minus**, функция **NM_pro** и функция **ScalarPro**. Условием остановки итерации является то, что число парадигм разности

между предыдущим вычислением и текущим результатом меньше или равно неко- торому значению (все принимается равным 1e-6), где число парадигм - наибольшее из абсолютных значений всех элементов на матрице (бесконечное число парадигм).

При вычислении F_{ij} вызывается функция **F_ij**, которая учитывает позици- онную связь соответствующего узла с областью D (трапеция) в данной задаче и вы- числяет интеграл по схеме площади поперечника, где для суждения о позиционной связи используется функция PiontClassific.

В процессе вычисления Aw вызываются функции **a_ij** и **b_ij** для расчета необходимых коэффициентов. Для точек, расположенных на краях виртуальной области (не имеющих левых, правых, верхних или нижних соседей), принимается единое значение 0.

Для технологии openMP распараллеливание осуществляется с использовани- ем соответствующих операторов там, где это необходимо, что приводит к значитель- ному повышению эффективности программы.

Используя write2DVectorToFile, можно вывести w в виде матрицы в некоторый txt-файл, а затем с помощью программы на языке python нарисовать соответствую- щее изображение.

Вышеуказанная информация описывает мою первоначальную версию подхода с использованием OpenMP, которую я впоследствии улучшил, применив свойства OpenMP (чем сложнее вычисления в цикле, тем выше эффективность OpenMP). В результате в коде осталось только две основные функции:

1.Rk(const vector<vector<double>>& w, const int count)
Эта функция вычисляет r_k на основе известного w.

2.M_minus_TauRk(const vector<vector<double>>& wk, const vector<vector<double>>& rk, double& local_norm_square, const int count)
Эта функция вычисляет новое w_{k+1}на основе известных w_kи r_k, а также вычисляет квадрат нормы разности w_{k+1} и w_k (local_norm_square), полученной в процессе итерации.

Параметр count упрощает реализацию параллелизма с использованием MPI, гарантируя, что данные передаются и принимаются в строго определённое время в пределах одного макроцикла.

Большая матрица w делится на части, каждая из которых обрабатывается отдельным процессом. Использование MPI основано на неблокирующей передаче сообщений с помощью функций Isend и Irecv. Ограничение на количество параметров обеспечивает точность передачи каждого сообщения.

1.Алгоритм работы каждого процесса:

1. Сначала вызывается Isend для отправки сообщения.
2. Выполняются вычисления узлов в подматрице, не зависящих от данных других процессов.
3. Затем вызывается Irecv для приёма сообщения, после успешного завершения которого обрабатываются оставшиеся узлы.

2.О вычислении параметра τ:

Так как τ уникален для всех процессов, для его вычисления используется функция MPI_Allreduce.

3.Обеспечение корректности Isend **и** Irecv**:**

Для гарантии успешного выполнения передачи сообщений используется функция MPI_Wait.

Комбинируя возможности OpenMP и MPI, удалось значительно повысить производительность программы и её эффективность в параллельных вычислениях.

5 Численный эксперимент

Для последующих экспериментов равномерно возьмем $\varepsilon = \max(h1, h2)^2$ (параметр, используемый для вычисления коэффициентов a_{ij} и b_{ij}) с $\delta = 1e^{-6}$ (точность итерации). При компиляции все программы оптимизируются с помощью параметра -O2.

Путем экспериментов было рассчитано численное решение w для размеров 10, 20, 40, 80 и 160, решение было записано в txt-файл, а результаты отображены на графике с помощью программы на языке python следующим образом.

Таблица 1: Таблица с результатами расчетов на ПВС IBM Polus

Число точек сетки M ×N	Время решения(s)
10 × 10	0.077
20 × 20	2.90
40 × 40	21.75

После добавления OpenMP для распараллеливания последовательного кода были проведены эксперименты для задачи размером 40×40. Для каждого количества потоков (1, 2, 4, 8, 16) были выполнены расчёты, и было определено время выполнения.

Таблица 2: Таблица с результатами расчетов на ПВС IBM Polus.

Число OpenMP-нитей	Число точек сетки M ×N	Время решения(s)	Ускорение
1	40 ×40	7.320250	1
2	40 ×40	4.150532	1.763690
4	40 ×40	2.816288	2.599255
8	40 ×40	2.099350	3.486913
16	40 ×40	2.027918	3.609737

Таблица 3: Таблица с результатами расчетов на ПВС IBM Polus.

Кол-во OpenMP-нитей	Число точек сетки (M x N)	Число итераций	Время решения	Ускорение
1	80×90	67481	203.80	1
2	80×90	48381	67.18	3.02985
4	80×90	59261	47.90	4.25470
8	80×90	54267	35.65	5.71669
16	80×90	47582	31.21	6.52996
1	160×180	123276	1788.15	1
4	160×180	142375	1012.16	1.76667
8	160×180	123468	770.35	2.32121
16	160×180	129788	376.49	4.74952
32	160×180	130309	238.87	7.48587

Таблица 4: Таблица с результатами расчетов на ПВС IBM Polus.

Количество процессов MPI	Кол-во OpenMP -нитей	Число точек сетки (M x N)	Число итераций	Время решения	Ускорение
2	1	80×90	50087	187.15	1.08897
2	2	80×90	43478	145.78	1.39800
2	4	80×90	39978	101.16	2.01463
2	8	80×90	45505	77.35	2.63477
4	1	160×180	117259	508.73	3.51492
4	2	160×180	124352	329.42	5.42817
4	4	160×180	133042	288.65	6.19487
4	8	160×180	107983	143.46	12.46445

В рамках задания были выполнены следующие этапы:

1. Реализована параллельная программа на языке C++14 для приближённого решения двумерной задачи Дирихле для уравнения Пуассона.
2. Проведено ускорение синхронного кода средствами OpenMP и MPI.
3. Проверено качество работы алгоритма для различных размеров сеток и количества потоков на ПВС IBM Polus.
4. Выполнена визуализация полученных результатов.

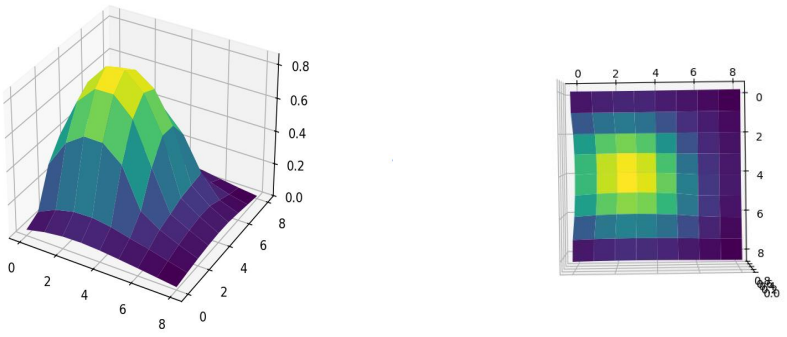

Рис. 1: Solution 10x10 1e-6

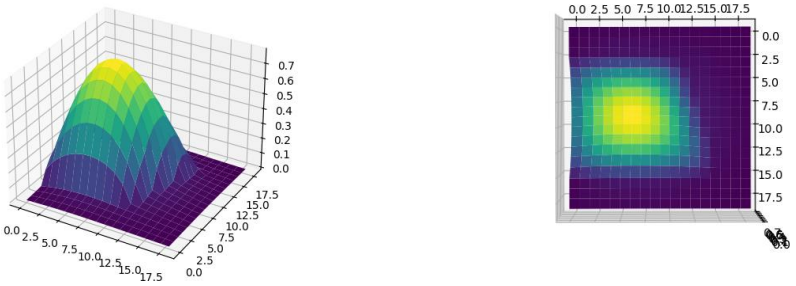

Рис. 2: Solution 20x20 1e-6

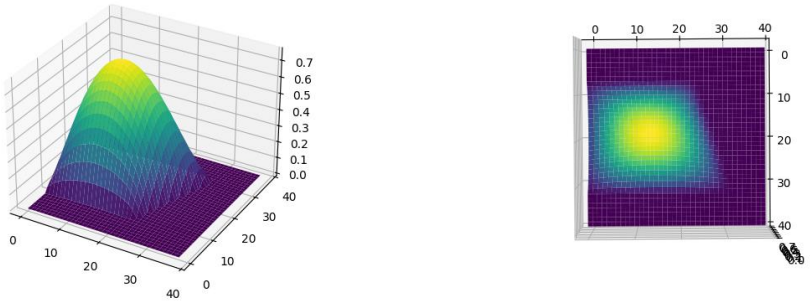

Рис. 3: Solution 40x40 1e-6

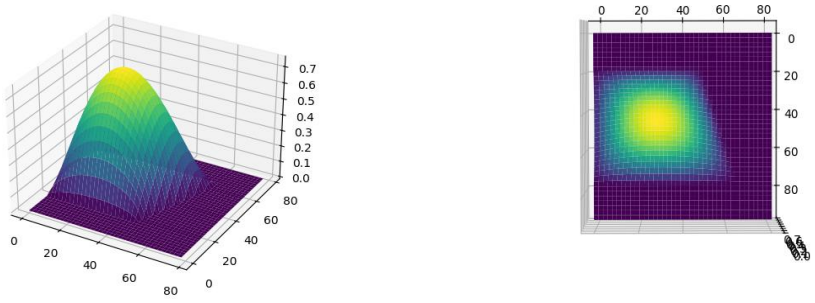

Рис. 4: Solution 80x90 1e-6

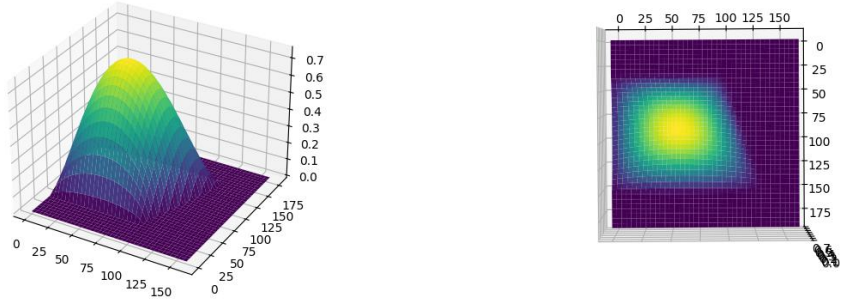

Рис. 5: Solution 160x180 1e-6

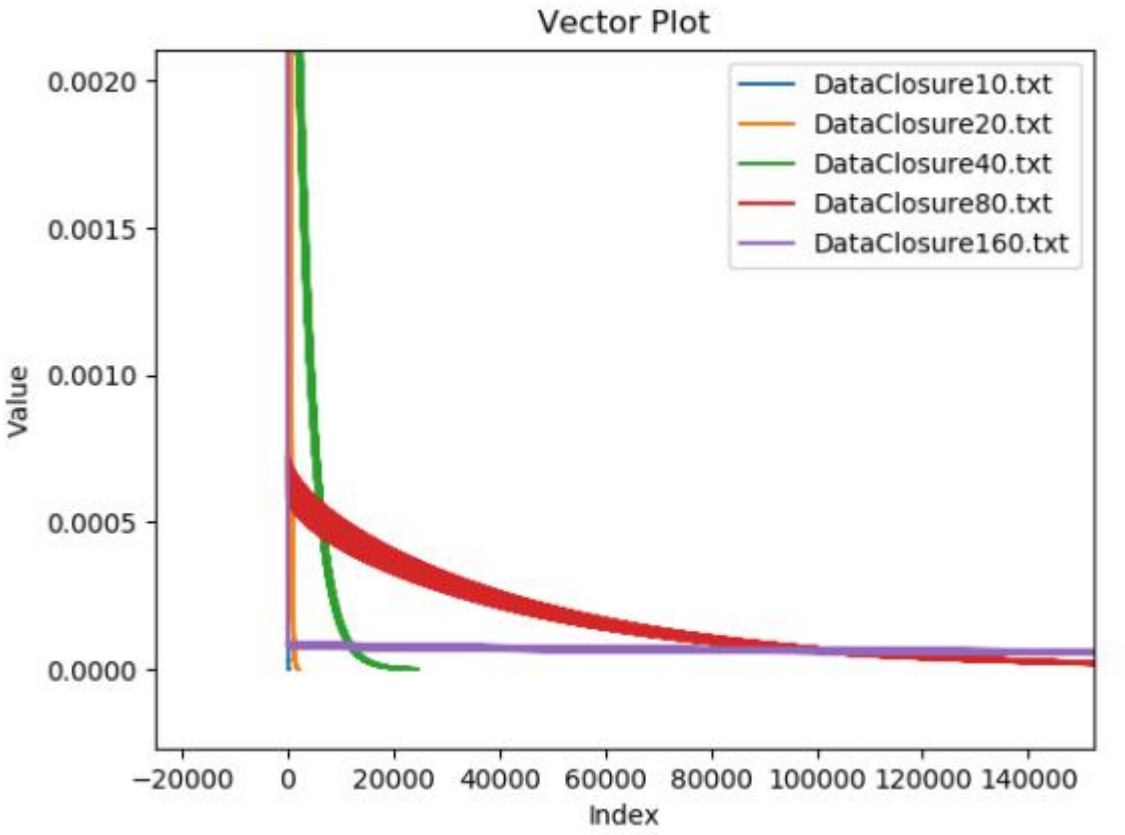

Рис. 6: График ошибок для 10×10, 20×20, 40×40, 80×90 и 160×18