

Московский государственный университет имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Кафедра математической физики

Киселёв Евгений Иванович

**Суперкомпьютерное моделирование и технологии**

ОТЧЁТ О ВЫПОЛНЕНИИ ЗАДАНИЯ

# **Математическая постановка задачи**

Рассмотрим задачу Дирихле для уравнения Пуассона:

Пусть теперь , и – замыкания областей . Пусть Г – граница . назовём фиктивной областью. Выберем и зафиксируем малое .

Рассмотрим задачу Дирихле в :

где ,

Требуется найти непрерывную в функцию , являющуюся решением данной задачи, такую, что

где , – вектор единичной нормали к границе в точке , определённый всюду или почти всюду на кривой.

Также известно, что .

# **Разностная схема и метод скорейшего спуска**

Краевую задачу предлагается решать численно методом конечных разностей.

Введём равномерную прямоугольную сетку

где .

Обозначим за внутренние узлы сетки. Рассмотрим линейное пространство функций, заданных на , и пусть – значение функции из в . Будем считать, что в задано скалярное произведение и евклидова норма: .

В методе конечных разностей дифференциальная задача математической физики заменяется конечно-разностной операторной задачей вида , где , . Такая задача называется разностной схемой. Решение этой задачи считается численным решением исходной дифференциальной задачи.

Дифференциальное уравнение во всех точках аппроксимируется следующим разностным уравнением:

где

и ,

где

Краевые условия аппроксимируется точным равенством для . Так как эти переменные исключаются из общей системы уравнений, то количество неизвестных совпадает с числом уравнений; при этом система является линейной и, следовательно, имеет единственное решение.

Интегралы в равенствах выше можно упростить.

Приближённое решение разностной схемы ищется методом скорейшего спуска:

где – невязка, – итерационный параметр.

Критерий остановки этого метода выглядит следующим образом:

*Замечание:* в численной схеме указано взять .

# **Программная реализация**

Этапы выполнения программы:

1. Инициализация данных
2. Заполнение матриц , ,
3. Цикл while (условие: пока ). В цикле
   1. Вычисление
   2. Вычисление
   3. Вычисление
   4. Вычисление
   5. В случае невыполнения условия цикла: копирование в
4. Сохранение решение
5. Освобождение памяти

Пункты 2, 3(a-e) задают циклы по сетке. Их можно распараллелить средствами OpenMP. Также в задании нужно было провести распараллеливание с помощью технологии MPI. В программе с помощью MPI\_Dims\_create выбирается наилучшее разбиение доступного числа процессоров в двумерную сетку. Инициализация , ,  происходит в локальных матрицах, размеры которых определяет сетка по процессорам. Расчёты для последовательной реализации (1 процессор и 1 нить) приведены для вариантов OpenMP и MPI+OpenMP соответственно, так как для компиляции использовались разные варианты компиляторов (IBM и mpic++ соответсвенно) и время работы отличалось:

|  |  |  |  |
| --- | --- | --- | --- |
| Реализация | Число точек сетки | Число итераций | Время решения |
| OpenMP |  | 13223 | 4.675[s] |
| OpenMP |  | 8817 | 12.587[s] |
| MPI+OpenMP |  | 13223 | 8.13[s] |
| MPI+OpenMP |  | 8817 | 24.38[s] |

График решения на сетке (180, 160) для :

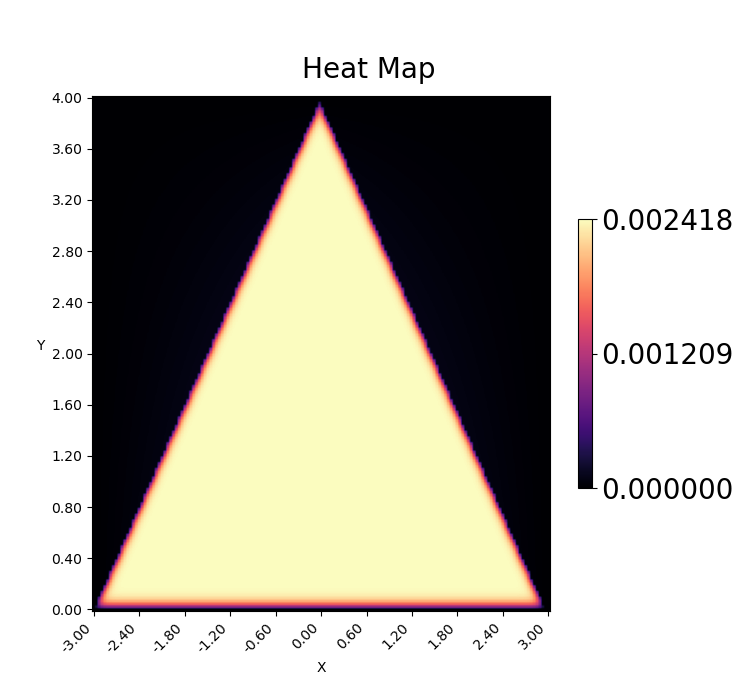


Таблица 1:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество нитей OpenMP | Число точек сетки | Число итераций | Время решения | Ускорение |
| 2 |  | 13223 | 2.461[s] | 1.9 |
| 4 |  | 13223 | 1.392[s] | 3.36 |
| 8 |  | 13223 | 1.114[s] | 4.2 |
| 16 |  | 13223 | 0.855[s] | 5.46 |
| 4 |  | 8817 | 3.288[s] | 3.83 |
| 8 |  | 8817 | 2.23[s] | 5.64 |
| 16 |  | 8817 | 1.956[s] | 6.43 |
| 32 |  | 8817 | 1.876[s] | 6.7 |

Таблица 2:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Количество процессоров  MPI | Количество нитей OpenMP | Число точек сетки | Число итераций | Время решения | Ускорение |
| 2 | 1 |  | 13233 | 4.41[s] | 1.84 |
| 2 | 2 |  | 13233 | 4.186[s] | 1.94 |
| 2 | 4 |  | 13233 | 2.362[s] | 3.14 |
| 2 | 8 |  | 13233 | 1.658[s] | 4.9 |
| 4 | 1 |  | 8817 | 5.93[s] | 4.1 |
| 4 | 2 |  | 8817 | 5.838[s] | 4.17 |
| 4 | 4 |  | 8817 | 4.68[s] | 5.2 |
| 4 | 8 |  | 8817 | 2.997[s] | 8.16 |

# **Заключение**

Таким образом была проделана работа по реализации метода конечных разностей в том числе и с помощью технологий OpenMP и MPI. В целом можно заключить, что с ростом числа процессоров/нитей ускорение программы увеличивается. Важно заметить, что времена работы программ, скомпилированных разными компиляторами, отличаются, вероятно из-за того, что компилятор IBM лучше подходит для железа распределённого кластера Полюс. Также можно заметить, что поведение ускорения программы имеет схожую динамику для случаев OpenMP и MPI+OpenMP реализаций (например, в случае OpenMP ускорение не сильно отличается для случаев 2-х и 4-х нитей, а у MPI+OpenMP реализации аналогичное поведение наблюдается для случаев 1-й и 2-х нитей).