**МИНОБРНАУКИ РОССИИ**

Федеральное государственное автономное образовательное

учреждение высшего профессионального образования

**«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»**

Факультет математики, механики и компьютерных наук

Кафедра алгебры и дискретной математики

**УЧЕБНАЯ ПРАКТИКА**

**по программированию**

Направление: Прикладная математика и информатика

Тема: «Стандартный и блочный алгоритмы LU – разложения матрицы».

Отчет студентки 3 курса Пирумян Маргариты Рубеновны

Руководитель практики: доц. кафедры А и ДМ, к.ф.-м.н. Столяр А.М.

Консультант: аспирант кафедры А и ДМ Юрушкин М.В.

Июль, 2014

Г.Ростов-на-Дону

**Содержание:**

**Содержание…………………………………………………………………1**

**1 Постановка задачи…………………………………………………………2**

**2 Описание алгоритма………………………………………………………5**

**3 Результаты расчетов….…………………………………………………...7**

**4 Литература………………………………………………………………….10**

**5 Текст программы…………………………………………….…………….11**

1. **Постановка задачи:**

**Написать стандартный и блочный алгоритмы LU - разложения матрицы. Сравнить их производительность.**

Задача Дирихле представляет собой задачу поиска решений дифференциального уравнения Лапласа при заданных значениях искомой функции на границе расчетной области.

Необходимо написать на любом языке программирования стандартный и блочный алгоритмы решения задачи Дирихле методом сеток. Это численный метод нахождения функции , удовлетворяющей в области уравнению:



и принимающей значения на границе области . В данном случае, для простоты, область -квадрат, и функция -тождественный ноль.



Метод конечных разностей (метод сеток) заключается в том, что элемент u(i, j) рассчитывается, в нашем случае, по формуле:

u(i, j)=1/4(u(i+1, j)+ u(i-1, j)+ u(i, j+1)+ u(i, j-1))

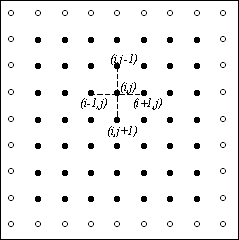


Рисунок 1 – графическое изображение расчета значения элемента u(i, j).

**2 Описание алгоритма:**

**2.1 Общие сведения**   
Программа имеет наименование «Lu decomposition». Данная программа функционирует на любых версиях операционных систем Windows и Linux, специализированного программного обеспечения для работы не требуется. Алгоритм реализован с помощью языка программирования C++ в среде разработки Microsoft Visual Studio 2010, на этапе тестирования использовался компилятор: GNU GCC Compiler для проверки правильности и скорости работы программы.  
  
**2.2 Функциональное назначение**  
Программа предназначена для изучения зависимости производительности работы программы от выбранного алгоритма решения задачи. Для этого реализуются два алгоритма LU - разложения матриц: стандартный и блочный. Реализация программы демонстрирует, что деление матриц на блоки уменьшает количество кэш-промахов, что способствует возрастанию скорости выполнения.   
  
**2.3 Описание логической структуры**  
**2.3.1 Алгоритм программы**  
***Стандартный алгоритм:***  
1) Создание матрицы и заполнение ее крайних элементов случайными числами. Все остальные элементы обнуляются.  
2) Передача параметров в функцию, представляющую собой стандартный алгоритм решения задачи Дирихле методом сеток.  
а) Работа с матрицей начинается с элемента с номером (1,1)  
б) Далее все элементы матрицы, кроме крайних, рассчитываются по формуле Matrices[i][j]=0.25\*(Matrices[i-1][j]+Matrices[i+1][j]+Matrices[i][j-1]+ Matrices[i][j+1])

в) Цикл по обходу матрицы выполняется заданное количество раз (число итераций).   
***Блочный алгоритм:***   
1) Создание матрицы и заполнение ее крайних элементов случайными числами. Все остальные элементы обнуляются.  
2) Передача параметров в функцию, представляющую собой стандартный алгоритм решения задачи Дирихле методом сеток.  
а) Матрица разбивается на некоторое число блоков, это число зависит от размеров матрицы.  
б) Над всеми элементами матрицы, лежащими внутри одного блока, выполняется стандартный алгоритм решения задачи Дирихле.

**2.3.2 Структура программы**  
Программа представляет собой 3 вспомогательные подпрограммы и тело программы.   
«Dirichlet» - реализация стандартного алгоритма решения задачи Дирихле, вызывается функцией «main»  
«Dirikhlet\_block» – реализация блочного алгоритма решения задачи Дирихле, вызывается функцией «main»  
«Print\_of\_Matrices» – простейшая функция вывода на экран матрицы для проверки правильности и идентичности алгоритмов на матрицах небольшого размера, вызывается функцией «main».  
«main» - тело программы.  
  
**2.4 Связи программы с другими программами**  
При работе программа связей с другими программными продуктами не имеет, но для экспериментального определения оптимального размера блока необходимы компиляторы или специализированные текстовые редакторы.  
  
**2.5 Используемые технические средства**  
При проектировании и тестировании использовались ЭВМ со следующими характеристиками:   
1) Процессор: Intel® Core™ i7 CPU 2.2GHz 3.2GHz  
ОЗУ: 4,00 Гб  
Тип системы: 64-разрядная ОС Windows 8.1  
  
**2.2.6 Вызов и загрузка**  
Вызывалась программа в среде разработки Microsoft Visual Studio 2010  
**2.2.7 Входные данные**   
Входными данными является матрица, определенная в пункте 2.3.1, размер матрицы и количество итераций выполнения алгоритма.   
**2.2.8 Выходные данные**  
Выходными данными являются матрицы, получающиеся после выполнения подпрограмм со стандартным и блочным алгоритмами, а также время выполнения каждого из методов.

1. **Результаты расчетов:**

Для определения разницы в скорости выполнения стандартного и блочного алгоритмов решения задачи Дирихле было выполнено несколько тестов для разных входных данных:

1. Для размера матрицы 100х100:



Рисунок 1 – графики зависимости скорости выполнения программы для стандартного и блочного алгоритмов при размере матрицы 100х100 от числа итераций выполнения алгоритма.

Таблица 1 – зависимость скорости

выполнения программы для стандартного и блочного алгоритмов при размере матрицы 100х100 от числа итераций выполнения алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| итерации | 10 | 100 | 1000 | 10000 |
| Стандартный (секунды) | 0,085 | 0,087 | 0,145 | 0,71 |
| Блочный (секунды) | 0,08 | 0,084 | 0.102 | 0,3 |

2) Для размера матрицы 1000х1000:



Рисунок 2 – графики зависимости скорости выполнения программы для стандартного и блочного алгоритмов при размере матрицы 1000х1000 от числа итераций выполнения алгоритма.

Таблица 2 – зависимость скорости

выполнения программы для стандартного и блочного алгоритмов при размере матрицы 1000х1000 от числа итераций выполнения алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| итерации | 10 | 100 | 1000 | 10000 |
| Стандартный (секунды) | 0,265 | 1,38 | 7,8 | 72 |
| Блочный (секунды) | 0,222 | 0,9 | 2,95 | 24 |

1. Для размера матрицы 10000х10000:



Рисунок 3 – графики зависимости скорости выполнения программы для стандартного и блочного алгоритмов при размере матрицы 10000х10000 от числа итераций выполнения алгоритма.

Таблица 3 – зависимость скорости

выполнения программы для стандартного и блочного алгоритмов при размере матрицы 10000х10000 от числа итераций выполнения алгоритма.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| итерации | 10 | 100 | 1000 | 10000 |
| Стандартный (секунды) | 11,5 | 107 | 1156 | 11097 |
| Блочный (секунды) | 6,2 | 59 | 664 | 5973 |

Из полученных результатов видно, что при малых размерах матрицы и при небольшом количестве итераций разница практически незаметна. Но при увеличении размера матрицы и количества итераций получаем выигрыш в скорости выполнения у блочного алгоритма, в среднем, от 35 до 60%. Во всех измерениях размеры блока были равны 4. Опытным путем было установлено, что это оптимальный размер блока для данной задачи. Также подходит и значение 3. Другие же размеры блоков не приведут к максимальному ускорению.

1. **Литература:**
2. Гергель В. П. Высокопроизводительные вычисления для многопроцессорных многоядерных систем: Учебник – М.: Издательство Московского университета, 2010. – 544 с., ил.
3. Веб-сайт: <http://ru.wikibooks.org/wiki/Решения_задачи_Дирихле_для_двумерного_уравнения_Лапласа_методом_сеток>
4. **Текст программы:**

Программа сравнения скорости выполнения стандартного и блочного алгоритмов задачи Дирихле.

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#define MATRICS\_SIZE 1000 //задаем размер матрицы

#define ITERATION 1000 //задаем количество итераций выполнения алгоритмов

#define BLOCK\_SIZE 4 //задаем размер блока для блочного алгоритма

**using namespace** **std;**

//стандартный алгоритм решения задачи Дирихле

**void** Dirikhlet(**double** \*Matrics){

**int** i, j; //переменные для цикла по обходу матрицы

**for**(**int** L=0; L<ITERATION; ++L) //цикл по итерациям

{

**for**(i=1; i<MATRICS\_SIZE-1; ++i) //обходим матрицу

**for**(j=1; j<MATRICS\_SIZE-1; ++j)

Matrics[i\*MATRICS\_SIZE+j]=(Matrics[(i-1)\*MATRICS\_SIZE+j]+Matrics[i\*MATRICS\_SIZE+j-1] +Matrics[(i+1)\*MATRICS\_SIZE+j]+Matrics[i\*MATRICS\_SIZE+j+1])/4; //формула для вычисления элемента Matrics[i],[j]

}

}

//блочный алгоритм решения задачи Дирихле

**void** Dirikhlet\_block(**double** \*Matrics){

**int** i, j; //переменные для цикла по обходу блока

**for**(**int** L=0; L<ITERATION; ++L) //цикл по итерациям

//разбиение матрицы на блоки

**for**( **int** n=0; n<MATRICS\_SIZE; n+= BLOCK\_SIZE)

**for**(**int** m=0; m<MATRICS\_SIZE; m+= BLOCK \_SIZE)

{//выносим из цикла повторяющиеся вычисления для увеличения скорости

**int** n0=**max**(1,n);

**int** m0=**max**(1,m);

**int** n1=**min**((n+ BLOCK\_SIZE), (MATRICS\_SIZE-1));

**int** m1=**min**((m+ BLOCK\_SIZE), (MATRICS\_SIZE-1));

//внутри блока работает стандартный алгоритм

**for**(i=n0; i<n1; ++i)

**for**(j=m0; j<m1; ++j)

{

**int** INDEX=i\*MATRICS\_SIZE;

//формула для вычисления элемента Matrics[i],[j]

Matrics[INDEX+j]=(Matrics[INDEX-MATRICS\_SIZE+j]+Matrics[INDEX+j-1] +Matrics[INDEX+MATRICS\_SIZE+j]+Matrics[INDEX+j+1])/4;

}

}

}

//функция печати матрицы

**void** Print\_of\_Matrics(**double** \*Matrics){

**for**(**int** i=0; i<MATRICS\_SIZE; ++i)

{

**for**(**int** j=0; j<MATRICS\_SIZE; ++j)

**cout**<<Matrics[i\*MATRICS\_SIZE+j]<<**" "**;

**cout**<<**endl**;

}

}

**int** main(){

**int** rand();

**int** i,j=0;

//создаем матрицу в виде одномерного массива

**double** \*Matrics = **new double**[MATRICS\_SIZE\*MATRICS\_SIZE];

//создание матрицы описанной в пункте 2.3.1 (все элементы нулевые, кроме граничных)

**for**(i=0; i<MATRICS\_SIZE; ++i)

{

**for**( j=0; j<MATRICS\_SIZE; ++j)

Matrics[i\*MATRICS\_SIZE+j]=0;

}

//заполнение граничных элементов матрицы произвольными числами

**for**(i=0; i<MATRICS\_SIZE; ++i)

{Matrics[i\*MATRICS\_SIZE]=rand()%100;

Matrics[i\*MATRICS\_SIZE+MATRICS\_SIZE-1]=rand()%100;

Matrics[i]=rand()%100;

Matrics[MATRICS\_SIZE\*(MATRICS\_SIZE-1)+i]=rand()%100;

}

// Print\_of\_Matrics(Matrics);

//по очереди запускаем то один алгоритм, то другой для расчета времени (меняем флаг: 1 – для выполнения стандартного алгоритма, 0 – для блочного)

#if 0

Dirikhlet(Matrics);

//ниже приведена функция печати, для определения правильности выполнения стандартного и блочного алгоритмов (для самого сравнения граничные элементы матрицы заполняются одними и теми же константами, а не произвольными числами.) Она используется только для матриц небольшого размера. Скорость же выполнения проверяется на больших матрицах.

// Print\_of\_Matrics(Matrics);

#else

Dirikhlet\_block(Matrics);

// Print\_of\_Matrics(Matrics);

#endif

**return** 0;

}