МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»

Институт информатики, математики и электроники
Факультет информатики
Кафедра технической кибернетики

Основы параллельных вычислений

Отчет по лабораторной работе №2

Студенты группы 6408 Яшакин А.Д.,
Коневский В.В.
Преподаватель Головашкин Д.Л.

САМАРА 2017

1 ВВЕДЕНИЕ

В настоящее время огромное количество задач требует большой производительности систем. Бесконечно увеличивать количество транзисторов на кристалле процессора не позволяют физические ограничения. Геометрические размеры транзисторов нельзя физически уменьшать, так как при превышении возможно допустимых размеров начинают проявляться явления, которые не заметны при больших размерах активных элементов — начинают сильно сказываться квантовые размерные эффекты. Транзисторы начинают работать не как транзисторы.
А закон Мура здесь ни при чем. Это был и остается законом стоимости, а увеличение количества транзисторов на кристалле — это скорее следствие из закона. Таким образом, для того, чтобы увеличивать мощность компьютерных систем приходится искать другие способы. Это использование мультипроцессоров, мультикомпьютеров. Такой подход характеризуется большим количеством процессорных элементов, что приводит к независимому исполнение подзадач на каждом вычислительном устройстве.

В данной лабораторной работе мы рассмотрим LU-разложение реализованное алгоритмами ikj ijk (строчно ориентированные), блочную реализацию gaxpy-разложения, а так же оценим эффективность применения SSE технологии.

2 ВЫЧИСЛИТЕЛЬНЫЕ АЛГОРИТМЫ

2.1 ikj – разложение.

Доступ к элементам матрицы A по строкам. Исключение по строкам. ГЭ по (i-1)-й строке.

```
for i = 2:N
        for k = 1:N-1
                L(i,k) =A(i,k)*A(k,k);
        For j=k+1:N
A(i,j) = A(i,j) – L(i,k)A(k,j)
                end;
        end;
end.
```

2.2 ijk-разложение

Доступ к элементам матрицы A по строкам. Исключения по строкам. Первый цикл по j находит элементы i-строки L. Второй цикл по j-элементы i-й строки U. ГЭ по (i-1)-й строке.

```
For i=2:n
 For j=2:i
  L(i,j-1)=A(i,j)/A(j-1,j-1)
   For k=1:j-1
        A(i,j)=A(i,j)-L(i,k)*A(k,j)
  For j=i+1:n
    For k=1:i-1
  A(i,j)=A(i,j)-L(i,k)*A(k,j)
   End;
 End;
End.
```

Gaxpy-версия исключения Гаусса. Главные подматрицы A(1:k,1:k) не вырождены для k=1:n-1. Данный алгоритм вычисляет разложение A=LU,где L-нижняя треугольная матрица, U-верхняя треугольная. Если i>j, то элементы A(i,j) содержат элементы L(i,j). Если i<=j,то элементы A(i,j) содержат элементы U(I,j).

```
For j=1:n
 For k=1:j-1
  For i=k+1:j-1
   A(i,j)=A(i,j)-A(i,k)A(k,j)
   End
End
For k=1:j-1
  For i=j:n
A(i,j)=A(i,j)-A(i,k)A(k,j)
  End
End
```

Блочная gaxpy-версия LU-разложения

```
La=1
 While La<=1
 u =min(n, La+r-1)
 r1=u-La+1
```

Выполняем r1 шагов алгоритма для A(La:n,La:n)

Замещаем A(La:u,u+1:n) в соответствии с решением

A(La:n,La:n)Z=A(La:u,u+1:n)

A(u+1:n,u+1:n)=A(u+1:n,u+1:n)-A(u+1:n,La:u)A(La:u,u+1:n)

```
For j=1:r1
 For k=1:j-1
  For i=k+1:j-1
La=u+1
   A(i,j)=A(i,j)-A(i,k)A(k,j)
   End
End
For k=1:j-1
  For i=j:r1
A(i,j)=A(i,j)-A(i,k)A(k,j)
  End
End
end
```

3 ВЫЧИСЛИТЕЛЬНЫЙ ЭКСПЕРИМЕНТ

3.1 Цель эксперимента

Эксперимент проводится с целью выявления зависимости скорости работы алгоритма от размера входных значений. А также для проверки эффективности технологии SSE.

3.2 Средства эксперимента

Эксперимент проводился с использованием следующих инструментов:

ПК: Intel Core i7 4790U 3.6 gHz, 16gb RAM.

ПО: VS 2015 Community x64.

Для реализации алгоритмов был использован язык C#, так как в данном языке имеется библиотека System.Numerics, позволяющая использовать SSE технологию.

3.3 Параметры эксперимента

Эксперимент будем проводить с квадратными матрицами, варьируя их размерность. Матрицы заполняются случайными числами. Начальный размер матриц –200×200 – подобран так, чтобы для всех алгоритмов время работы было отлично от 0 секунд. Проведем серию из 18 экспериментов с шагом в 100. Шаг подобран таким образом, чтобы была наглядно видна разница во времени между итерациями. Результаты этого эксперимента приведены в таблице 1 и на рисунке 2.

3.4 Теоретические ожидания

На матрицах малых размеров SSE технология не даст явного прироста скорости, однако для больших размерностей ожидается явный прирост во времени выполнения программы, так как технология SSE предполагает параллельное выполнение математических операций. Ожидаемая скорость работы алгоритмов ijk и ikj без использования технологии SSE равна $(2/3)*n^3$.

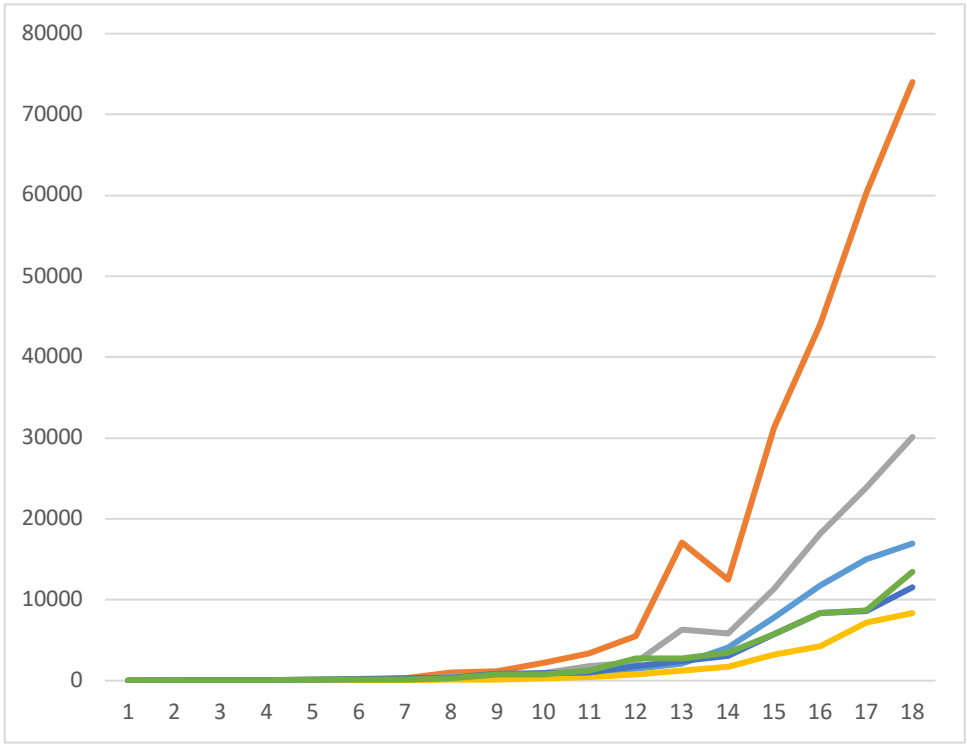
3.5 Результаты

3.5.1 Эксперимент 1

Таблица 1 – Время работы алгоритмов

	Время работы, мс					
Размер перемножаемых матриц	IKJ	IJK	SSE IJK	SEE IKJ	GAXPY	GAXPY SSE
200×200	5	8	4	1	8	5
300×300	7	11	6	2	37	30
400×400	15	29	22	6	28	23
500×500	31	60	30	10	61	53
600×600	35	75	54	11	111	106
700×700	60	102	52	17	178	121
800×800	144	256	149	39	275	157
900×900	266	981	559	112	392	318
1000×1000	459	1133	597	171	836	755
1100×1100	679	2158	934	280	938	800
1200×1200	1014	3372	1792	470	1011	1223
1300×1300	1461	5548	2298	785	1805	2713
1400×1400	2137	17074	6324	1212	2409	2764
1500×1500	4066	12487	5841	1712	3027	3493
1700×1700	7786	31362	11395	3250	5782	5792
1900×1900	11784	44049	18170	4272	8331	8379
2000×2000	14989	60285	23851	7158	8569	8654
2500×2500	16959	74013	30124	8357	11545	13457

Рисунок 2 – Время работы алгоритмов



3.6 Описание результатов

Из графика, изображенного на рисунке 2, видно, что время работы каждого метода экспоненциально зависит от роста размеров матрицы.

3.7 Анализ результатов

На основании проведенных экспериментов после оценки результатов можно сделать следующие выводы.

Все результаты являются приблизительными, так как на времени работы алгоритма сказывается и загруженность процессора в данный момент. Поэтому в разный момент времени, вычисления для одного и того же алгоритма для той же размерности матрицы занимает различное время.

Как и ожидалось, что при использовании технологии SSE на больших матрицах скорость работы ijk- и ikj-разложений увеличилась, так как данная технология выполняет параллельное вычисление математических операций. Использование блочной gaxpy-версии LU-разложения, показала большую скорость работы, чем SSE gaxpy-версия LU-разложения, так как процессор компьютера был загружен посторонними процессами.

Начиная с матриц размером 1000×1000, были заметны значительные различия в скорости работы программы.

.

4 ЗАКЛЮЧЕНИЕ

В процессе данной лабораторной работы были проанализированы алгоритмы разложения матрицы. Целесообразность и использование технологии SSE зависит от размера матриц, на малых размерах матриц данная технология не дает выигрыш по времени. Самые быстрые результаты показал ikj-разложение с использованием технологии SSE.

5 ПРИЛОЖЕНИЕ А. КОД ПРОГРАММЫ

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Numerics;

namespace L2OPV
{
   class Program
   {
      static void Main(string[] args)
      {
         int n = 200;
         while(n < 2500)
         {
            Console.Write("n = " + n);
            //int n = Int32.Parse(Console.ReadLine());
            Console.WriteLine();

            task_1(n);
            task_2(n);
            task_3(n);
            task_4(n);

            task_5(n, 2);
            task_6(n, 2);
            Console.WriteLine("CLICK ENTER !!!");
            //Console.ReadKey();
            Console.WriteLine();

            n += 100;
         }
         Console.ReadKey();
      }

      static void task_1(int n)
      {
         double[,] A = Genrator(n);
         double[,] test = Copy(A);
         double[,] L = GenratorE(n);

         var sw = new System.Diagnostics.Stopwatch();
         sw.Start();
         for (int i = 1; i < n; i++)
```

```
    {
        for (int k = 0; k < i; k++)
        {
            L[i, k] = A[i, k] / A[k, k];
            for (int j = k; j < n; j++)
            {
                A[i, j] = A[i, j] - L[i, k] * A[k, j];
            }
        }
    }
    sw.Stop();

    Console.WriteLine("IKJ: " + (sw.ElapsedMilliseconds) + " ms");
    if (EqvalsMatrix(ikjAlgorithm(L, A), test))
        Console.WriteLine("SUCCESS");
    else Console.WriteLine("ERROR");
}

static void task_2(int n)
{
    double[,] A = Genrator(n);
    double[,] test = Copy(A);
    double[,] L = GenratorE(n);

    var sw = new System.Diagnostics.Stopwatch();
    sw.Start();
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < j; k++)
            {
                A[i, j] = A[i, j] - L[i, k] * A[k, j];
            }
            if (i < j)
            {
                double sum = 0;
                for (int k = 0; k < i; k++)
                {
                    sum += L[j, k] * A[k, i];
                }
                L[j, i] = (A[j, i] - sum) / A[i, i];
            }
        }
    }
    sw.Stop();
```

```
    Console.WriteLine("IJK: " + (sw.ElapsedMilliseconds) + " ms");
    if (EqvalsMatrix(ikjAlgorithm(L, A), test))
    Console.WriteLine("SUCCESS");
    else Console.WriteLine("ERROR");
}

static void task_3(int n)
{
    Random rand = new Random();
    Vector<double>[] a = new Vector<double>[n * n];
    double[,] test = new double[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            double value = rand.Next(0, 10);
            a[Index(i, j, n)] = new Vector<double>(value);
            test[i, j] = value;
        }
    }
    Vector<double>[] l = new Vector<double>[n * n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            l[Index(i, j, n)] = new Vector<double>(0);
            if (i == j) l[Index(i, j, n)] = new Vector<double>(0);
        }
    }


    var sw = new System.Diagnostics.Stopwatch();
    sw.Start();
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < i; k++)
            {
                a[Index(i, j, n)] -= l[Index(i, k, n)] * a[Index(j, k, n)];
            }
            if (i < j)
            {
                Vector<double> sum = new Vector<double>(0);
                for (int k = 0; k < i; k++)
                {
```

```
                sum = l[Index(j, k, n)] * a[Index(k, i, n)] + sum;
            }
            l[Index(j, i, n)] = (a[Index(j, i, n)] - sum) / a[Index(i, i, n)];
        }
    }
}
sw.Stop();
Console.WriteLine("SSE IJK: " + (sw.ElapsedMilliseconds) + " ms");
}

static int Index(int i, int j, int n)
{
    return i * n + j;
}

static void task_4(int n)
{
    Random rand = new Random();
    Vector<double>[] a = new Vector<double>[n*n];
    double[,] test = new double[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            double value = rand.Next(0, 10);
            a[Index(i, j, n)] = new Vector<double>(value);
            test[i, j] = value;
        }
    }
    Vector<double>[] l = new Vector<double>[n * n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            l[Index(i, j, n)] = new Vector<double>(0);
            if(i == j) l[Index(i, j, n)] = new Vector<double>(0);
        }
    }


    var sw = new System.Diagnostics.Stopwatch();
    sw.Start();
    for (int i = 1; i < n; i++)
    {
        for (int k = 0; k < i; k++)
        {
```

```
            l[Index(i, k, n)] = a[Index(i, k, n)] / a[Index(k, k, n)];
            for (int j = k; j < n; j++)
            {
                a[Index(i, j, n)] -= a[Index(i, k, n)] * l[Index(k, j, n)];
            }
        }
    }
    sw.Stop();
    Console.WriteLine("SSE IKJ: " + (sw.ElapsedMilliseconds) + " ms");
}

static void task_5(int n, int r)
{
    double[,] A = Genrator(n);
    double[,] test = Copy(A);

    double[,] temp;

    int lymda = 0;
    int ny;
    int r_1;
    var sw = new System.Diagnostics.Stopwatch();
    sw.Start();
    while (lymda < n)
    {
        ny = Math.Min(n, lymda + r - 1);
        r_1 = ny - lymda + 1;
        temp = GaxpyBlockA(A, lymda, n);
        temp = Gaxpy(temp, r_1);
        A = GaxpyBlockB(temp, A, lymda, n);

        for (int i = ny + 1, s = lymda; i < n; i++, s++)
        {
            for (int j = ny + 1, k = lymda; j < n; j++, k++)
            {
                A[i, j] -= A[i, k] * A[s, j];
            }
        }

        lymda = ny + 1;
    }
    sw.Stop();
    Console.WriteLine("GAXPY: " + (sw.ElapsedMilliseconds) + " ms");
}

static double[,] GaxpyBlockA(double[,] a, int l, int r)
```

```
{
    int n = r - l;
    double[,] b = new double[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            b[i, j] = a[i + l, j + l];
        }
    }

    return b;
}

static double[,] GaxpyBlockB(double[,] a, double[,] b, int l, int r)
{
    for (int i = l, s = 0; i < r; i++, s++)
    {
        for (int j = l, k = 0; j < r; j++, k++)
        {
            b[i, j] = a[s, k];
        }
    }
    return b;
}


static double[,] Gaxpy(double[,] a, int n)
{
    for (int j = 0; j < n; j++)
    {
        for (int k = 0; k < j - 1; k++)
        {
            for (int i = k + 1; i < j - 1; i++)
            {
                a[i, j] -= a[i, k] * a[k, j];
            }
        }
        for (int k = 0; k < j - 1; k++)
        {
            for (int i = j; i < n; i++)
            {
                a[i, j] -= a[i, k] * a[k, j];
            }
        }
```

```
            for (int s = j + 1; s < n; s++)
            {
                a[s, j] = a[s, j] / a[j, j];
            }
        }
        return a;
    }

//_____________________________________________________________________
___
    static void task_6(int n, int r)
    {
        Random rand = new Random();
        Vector<double>[] A = new Vector<double>[n * n];
        double[,] test = new double[n, n];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                double value = rand.Next(0, 10);
                A[Index(i, j, n)] = new Vector<double>(value);
                test[i, j] = value;
            }
        }

        Vector<double>[] temp;

        int lymda = 0;
        int ny;
        int r_1;
        var sw = new System.Diagnostics.Stopwatch();
        sw.Start();
        while (lymda < n)
        {
            ny = Math.Min(n, lymda + r - 1);
            r_1 = ny - lymda + 1;
            temp = GaxpyBlockA(A, lymda, n);
            temp = Gaxpy(temp, r_1);
            A = GaxpyBlockB(temp, A, lymda, n, r_1);

            for (int i = ny + 1, s = lymda; i < n; i++, s++)
            {
                for (int j = ny + 1, k = lymda; j < n; j++, k++)
                {
                    A[Index(i, j, n)] -= A[Index(i, k, n)] * A[Index(s, j, n)];
                }
```

```
            }

            lymda = ny + 1;
        }
        sw.Stop();
        Console.WriteLine("GAXPY SSE: " + (sw.ElapsedMilliseconds) + " ms");
    }

    static Vector<double>[] GaxpyBlockA(Vector<double>[] a, int l, int r)
    {
        int n = r - l;
        Vector<double>[] b = new Vector<double>[n * n];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                b[Index(i, j, n)] = a[Index(i+l, j+l, n)];
            }
        }

        return b;
    }

    static Vector<double>[] GaxpyBlockB(Vector<double>[] a,
Vector<double>[] b, int l, int r, int n)
    {
        for (int i = l, s = 0; i < r; i++, s++)
        {
            for (int j = l, k = 0; j < r; j++, k++)
            {
                b[Index(i, j, n)] = a[Index(s, k, n)];
            }
        }
        return b;
    }


    static Vector<double>[] Gaxpy(Vector<double>[] a, int n)
    {
        for (int j = 0; j < n; j++)
        {
            for (int k = 0; k < j - 1; k++)
            {
                for (int i = k + 1; i < j - 1; i++)
                {
                    a[Index(i, j, n)] -= a[Index(i, k, n)] * a[Index(k, j, n)];
```

```
            }
        }
        for (int k = 0; k < j - 1; k++)
        {
            for (int i = j; i < n; i++)
            {
                a[Index(i, j, n)] -= a[Index(i, k, n)] * a[Index(k, j, n)];
            }
        }

        for (int s = j + 1; s < n; s++)
        {
            a[Index(s, j, n)] = a[Index(s, j, n)] / a[Index(j, j, n)];
        }
    }
    return a;
}

//_____
____
    static void Write(double[,] a)
    {
        for (int i = 0; i < a.GetLength(0); i++)
        {
            for (int j = 0; j < a.GetLength(0); j++)
            {
                Console.Write(a[i, j] + " ");
            }
            Console.WriteLine();
        }
    }
    static double[,] Copy(double[,] a)
    {
        int n = a.GetLength(0);
        double[,] c = new double[n, n];
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
            {
                c[i, j] = a[i, j];
            }
        }
        return c;
    }
    static double[,] Genrator(int n)
    {
```

```
    Random rand = new Random();
    double[,] a = new double[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            a[i, j] = rand.Next(0, 10);
        }
    }
    return a;
}
static double[,] GenratorE(int n)
{
    Random rand = new Random();
    double[,] a = new double[n, n];
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            a[i, j] = 0;
            if(j == i) a[i, j] = 1;
        }
    }
    return a;
}

static double[,] ikjAlgorithm(double[,] A, double[,] B)
{

    int n = A.GetLength(0);

    // initialise C
    double[,] C = new double[n, n];

    for (int i = 0; i < n; i++)
    {
        for (int k = 0; k < n; k++)
        {
            for (int j = 0; j < n; j++)
            {
                C[i, j] += A[i, k] * B[k, j];
            }
        }
    }
    return C;
}
```

```
        static bool EqvalsMatrix(double[,] a, double[,] b)
        {
            int n = a.GetLength(0);
            for (int i = 0; i < n; i++)
            {
                for (int j = 0; j < n; j++)
                {
                    if ((a[i, j] - b[i, j]) > 0.1) return false;
                }
            }
            return true;
        }
    }
}
```