分布式系统作业

第2次作业

姓名：郝裕玮

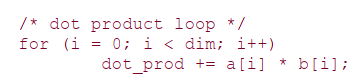
班级：计科1班

学号：18329015

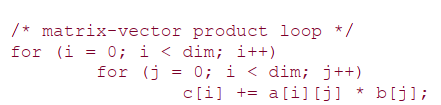
1. 问题描述

1、分别采用不同的算法（非分布式算法）例如一般算法、分治算法和Strassen算法等计算计算矩阵两个300x300的矩阵乘积，并通过Perf工具分别观察cache miss、CPI、mem\_load等性能指标, 找出特征或者规律。

2、考虑一个内存系统，其一级缓存为 32 KB，DRAM 为 512 MB，处理器运行频率为 1 GHz。L1 缓存的延迟为一个周期，DRAM 的延迟为 100 个周期。 在每个内存周期中，处理器获取四个字（缓存线大小为四个字）。 两个向量的点积的最高可实现性能是多少？ 注意：如有必要，请假设最佳缓存放置策略。



3、现在考虑使用双环点积公式将密集矩阵与向量相乘的问题。矩阵的维度为 4K x 4K。（矩阵的每一行占用 16 KB 的存储空间。）可实现的峰值性能是多少 这种技术使用基于双循环点积的矩阵向量积？



1. 解决方案

**对于第1题**：

我们先编写最为简单的一般算法，即通过三层循环来进行矩阵运算（即根据定义逐个相乘和累加）。代码及其分析如下所示（分析已放在代码注释中）：

#include<iostream>

using namespace std;

//矩阵一般乘法

void simple(int\*\*a,int\*\*b,int\*\*c,int n){

    int i,j,k;

    for(i=0;i<=n-1;i++){

        for(j=0;j<=n-1;j++){

            for(k=0;k<=n-1;k++){

                c[i][j]+=a[i][k]\*b[k][j];//根据矩阵计算公式得出（注意不能直接等于，而应该是累加）

            }

        }

    }

}

int main(){

    int n=512;//矩阵维度，在这里代表矩阵是512\*512的

    //因为分治和Strassen的应用场景均是矩阵维度为2的n次方，所以将300\*300统一修改为512\*512

    int i,j;

    int\*\* a,\*\*b,\*\*c;//声明二维数组指针，为何不直接声明二维数组：int a[n][n]会在Strassen程序中说明

    a=new int\*[n];

    b=new int\*[n];

    c=new int\*[n];

    for(i=0;i<=n-1;i++){

        a[i]=new int[n];

        b[i]=new int[n];

        c[i]=new int[n];

    }

    //以上均为创建二维数组的过程

    //给数组a赋值

    for(i=0;i<=n-1;i++){

        for(j=0;j<=j-1;j++){

            a[i][j]=i+1;

        }

    }

    //给数组b赋值

    for(i=0;i<=n-1;i++){

        for(j=0;j<=j-1;j++){

            b[i][j]=j+1;

        }

    }

    //为了与分治和Strassen保持一致（因为这两个程序涉及到递归函数），所以在这里不直接进行运算而是通过调用函数进行运算

    simple(a,b,c,n);

}

由于分治算法和Strassen算法原理是一致的，所以这里只写出更加优化的Strassen算法去与一般算法进行性能比较。

Strassen代码如下所示（分析已放在代码注释中）：

#include <iostream>

using namespace std;

//矩阵加法

void add(int\*\* a,int\*\* b,int\*\* c,int n){

    int i,j;

    for(i=0;i<=n-1;i++){

        for(j=0;j<=n-1;j++){

            c[i][j]=a[i][j]+b[i][j];

        }

    }

}

//矩阵减法

void sub(int\*\* a,int\*\* b,int\*\* c,int n){

    int i,j;

    for(i=0;i<=n-1;i++){

        for(j=0;j<=n-1;j++){

            c[i][j]=a[i][j]-b[i][j];

        }

    }

}

//矩阵乘法

void mul(int\*\* a,int\*\* b,int\*\* c,int n){

    int i,j,k;

    for(i=0;i<=n-1;i++){

        for(j=0;j<=n-1;j++){

            c[i][j]=0;//因为乘法需要累加，所以要先清空数组c原先的数据

        }

    }

    for(i=0;i<=n-1;i++){

        for(j=0;j<=n-1;j++){

            for(k=0;k<=n-1;k++){

                c[i][j]+=a[i][k]\*b[k][j];

            }

        }

    }

}

//Strassen算法实现

void strassen(int \*\*a,int \*\*b,int \*\*c,int n){

    int i,j;

    if(n<=64){

        mul(a,b,c,n);

        //查阅资料可知，最优的界限值在32～128之间，所以这里设置为当n缩小至64时不再需要继续四等分，直接进行普通矩阵乘法即可

        //继续递归分治会导致Strassen算法的效率降低

    }

    else{

        int\*\* a11,\*\*a12,\*\*a21,\*\*a22;

        int\*\* b11,\*\*b12,\*\*b21,\*\*b22;

        int\*\* c11,\*\*c12,\*\*c21,\*\*c22;

        int\*\* P1,\*\*P2,\*\*P3,\*\*P4,\*\*P5,\*\*P6,\*\*P7;

        int\*\* a\_final,\*\*b\_final;

        a11=new int\*[n/2];

        a12=new int\*[n/2];

        a21=new int\*[n/2];

        a22=new int\*[n/2];

        b11=new int\*[n/2];

        b12=new int\*[n/2];

        b21=new int\*[n/2];

        b22=new int\*[n/2];

        c11=new int\*[n/2];

        c12=new int\*[n/2];

        c21=new int\*[n/2];

        c22=new int\*[n/2];

        P1=new int\*[n/2];

        P2=new int\*[n/2];

        P3=new int\*[n/2];

        P4=new int\*[n/2];

        P5=new int\*[n/2];

        P6=new int\*[n/2];

        P7=new int\*[n/2];

        a\_final=new int\* [n/2];

        b\_final=new int\* [n/2];

        for(i=0;i<=n/2-1;i++){

            a11[i]=new int[n/2];

            a12[i]=new int[n/2];

            a21[i]=new int[n/2];

            a22[i]=new int[n/2];

            b11[i]=new int[n/2];

            b12[i]=new int[n/2];

            b21[i]=new int[n/2];

            b22[i]=new int[n/2];

            c11[i]=new int[n/2];

            c12[i]=new int[n/2];

            c21[i]=new int[n/2];

            c22[i]=new int[n/2];

            P1[i]=new int[n/2];

            P2[i]=new int[n/2];

            P3[i]=new int[n/2];

            P4[i]=new int[n/2];

            P5[i]=new int[n/2];

            P6[i]=new int[n/2];

            P7[i]=new int[n/2];

            a\_final[i]=new int[n/2];

            b\_final[i]=new int[n/2];

        }

        //根据原矩阵与四个子矩阵的位置关系将值赋给A和B的四个子矩阵

        for(i=0;i<=n/2-1;i++){

            for (j=0;j<=n/2-1;j++){

                a11[i][j]=a[i][j];

                a12[i][j]=a[i][j+n/2];

                a21[i][j]=a[i+n/2][j];

                a22[i][j]=a[i+n/2][j+n/2];

                b11[i][j]=b[i][j];

                b12[i][j]=b[i][j+n/2];

                b21[i][j]=b[i+n/2][j];

                b22[i][j]=b[i+n/2][j+n/2];

            }

        }

        //以上均为声明新数组a11,a12,a21,a22,b11,b12,b21,b22,c11,c12,c21,c22

        //以及P1,P2,P3,P4,P5,P6,P7,a\_final,b\_final

        //P1=A11\*(B12-B22)

        sub(b12,b22,b\_final,n/2);

        strassen(a11,b\_final,P3,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P2=(A11+A12)\*B22

        add(a11,a12,a\_final,n/2);

        strassen(a\_final,b22,P5,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P3=(A21+A22)\*B11

        add(a21,a22,a\_final,n/2);

        strassen(a\_final,b11,P2,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P4=A22\*(B21-B11)

        sub(b21,b11,b\_final,n/2);

        strassen(a22,b\_final,P4,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P5=(A11+A22)\*(B12+B22)

        add(a11,a22,a\_final,n/2);

        add(b11,b22,b\_final,n/2);

        strassen(a\_final,b\_final,P1,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P6=(A12-A22)\*(B21+B22)

        sub(a12,a22,a\_final,n/2);

        add(b21,b22,b\_final,n/2);

        strassen(a\_final, b\_final, P7,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //P7=(A11-A21)\*(B11+B12)

        sub(a11,a21,a\_final,n/2);

        add(b11,b12,b\_final,n/2);

        strassen(a\_final,b\_final,P6,n/2);//代入strassen继续递归，会在最底层进行乘法运算（调用mul函数）

        //C11=P5+P4-P2+P6;

        add(P5,P6,a\_final,n/2);

        sub(P4,P2,b\_final,n/2);

        add(a\_final,b\_final,c11,n/2);

        //C12=P1+P2;

        add(P1,P2,c12,n/2);

        //C21=P3+P4;

        add(P3,P4,c21,n/2);

        //C22=P5+P1-P3-P7;

        sub(P5,P3,a\_final,n/2);

        sub(P1,P7,b\_final,n/2);

        add(a\_final,b\_final,c22,n/2);

        //根据四个子矩阵与原矩阵的位置关系将值赋给数组c

        for(i=0;i<=n/2-1;i++){

            for(j=0;j<=n/2-1;j++){

                c[i][j]=c11[i][j];

                c[i][j+n/2]=c12[i][j];

                c[i+n/2][j]=c21[i][j];

                c[i+n/2][j+n/2]=c22[i][j];

            }

        }

    }

}

int main()

{

    int n=512;//矩阵维度，在这里代表矩阵是512\*512的

    int i,j;

    int\*\* a,\*\*b,\*\*c;//声明二维数组指针

    a=new int \*[n];

    b=new int \*[n];

    c=new int \*[n];

    for(i=0;i<=n-1;i++){

        a[i]=new int[n];

        b[i]=new int[n];

        c[i]=new int[n];

    }

    //以上均为创建二维数组的过程

    //给数组a赋值

    for(i=0;i<=n-1;i++){

        for(j=0;j<=j-1;j++){

            a[i][j]=i+1;

        }

    }

    //给数组b赋值

    for(i=0;i<=n-1;i++){

        for(j=0;j<=j-1;j++){

            b[i][j]=j+1;

        }

    }

    //初始化数组c

    for(i=0;i<=n-1;i++){

        for(j=0;j<=j-1;j++){

            c[i][j]=0;

        }

    }

    strassen(a,b,c,n);

}

对于一般程序和Strassen程序的补充：因为在Strassen程序中需要不断地将矩阵划分为四个子矩阵，所以在进行递归的过程中，矩阵大小在不断变化。

如果采取这种函数原型：

void strassen(int a[][n], int b[][n], int c[][n],int n)

则必须指定数组的列数，但又因为矩阵大小不断变化且列数不可以用变量表示（必须使用常量），所以最终只能采取下面这种函数原型：

void add(int\*\* a,int\*\* b,int\*\* c,int n)

这种函数原型使得只需传递二维数组的指针且无需传递数组大小，但是在创建数组时较为麻烦，如下所示：

    int\*\* a,\*\*b,\*\*c;//声明二维数组指针

    a=new int \*[n];

    b=new int \*[n];

    c=new int \*[n];

    for(i=0;i<=n-1;i++){

        a[i]=new int[n];

        b[i]=new int[n];

        c[i]=new int[n];

    }

    //以上均为创建二维数组的过程

**对于第2题：**

因为每个内存周期中，缓存线大小为4个字=32位，且一个int型变量占4 bytes=32位，所以缓存线一次可以存取4个元素。

易知第1次循环时对于数组a和b会发生cache miss，之后立刻会存入当前a和b的4个元素（a[i]~a[i+3]和b[i]~b[i+3]），使得后面3次循环均cache hit。

又因为每次循环都会使i+1，所以易知每4次循环会在第1次循环时发生2次cache miss。在此期间的浮点数运算次数为4\*2=8（4次循环，每次循环里1次加法1次乘法）次。

所以在每4次循环中，cache miss所用的时钟周期为2\*100 = 200 cycles，

cache hit所用的时钟周期数为（3\*4-2）\*1 = 10 cycles。

所以，最高实现性能 =

**对于第3题：**

对于数组a，它的cache miss发生在内部循环中，每4次内部循环就会发生1次cache miss。

对于数组b，在第1次外层循环（i=0）中，每4次内部循环就会发生1次cache miss，但由于一级缓存为32KB，而整个b数组只占用16KB（因为是一维数组），所以在第1次外层循环结束后，整个b数组都会缓存到一级缓存中，所以在之后的循环中不会再发生cache miss。

对于数组c，要在4K\*4=16K次循环后（外部循环执行4次），才会发生1次cache miss。

所以综上所述，数组b和c的cache miss可以忽略不计，只需计算数组a的cache miss。

在此期间的浮点数运算次数为4\*2=8（4次循环，每次循环里1次加法1次乘法）次。

在每4次循环中，cache miss所用的时钟周期为1\*100 = 100 cycles，

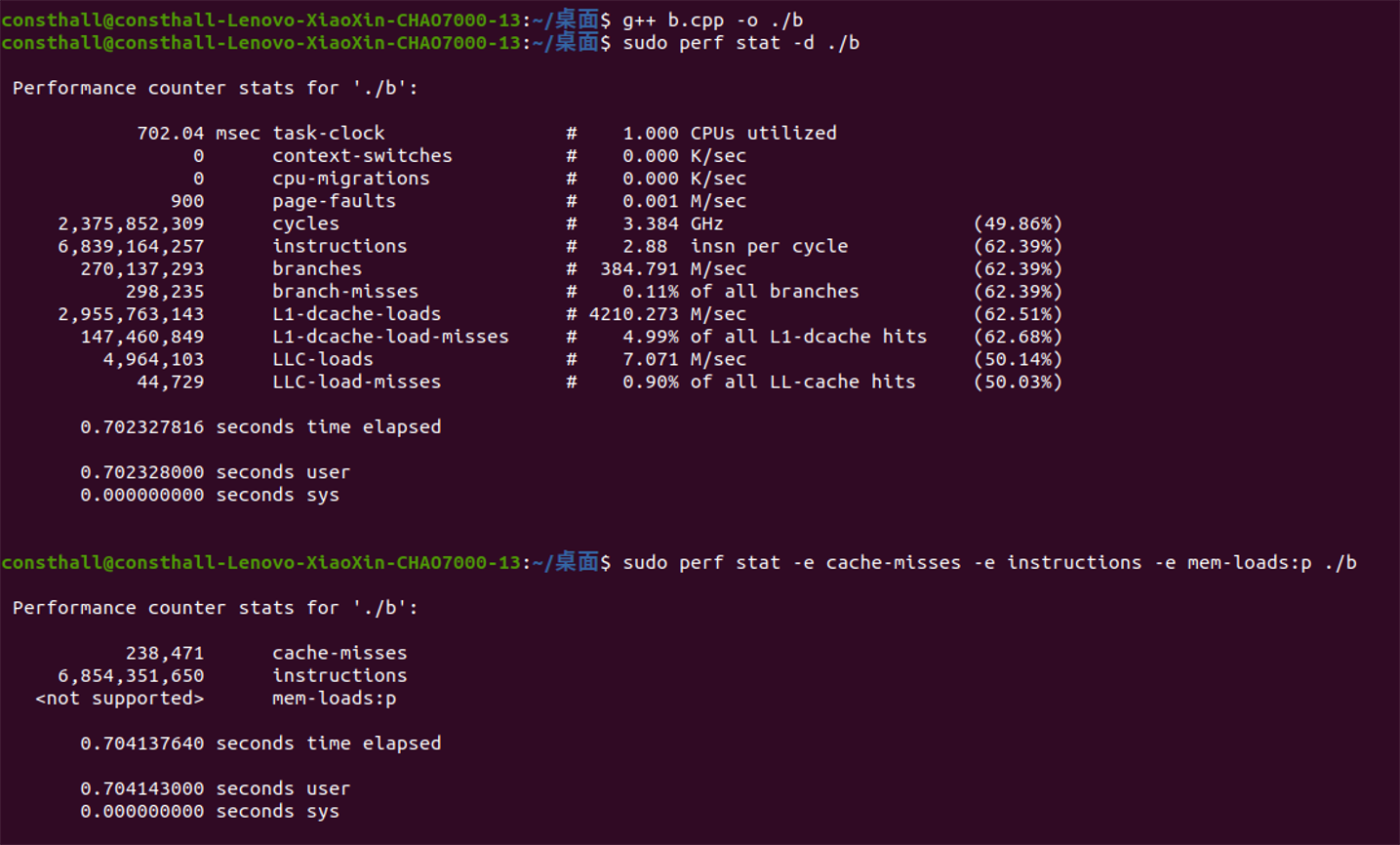
cache hit所用的时钟周期数为（3\*4-1）\*1 = 11 cycles。

所以，最高实现性能 =

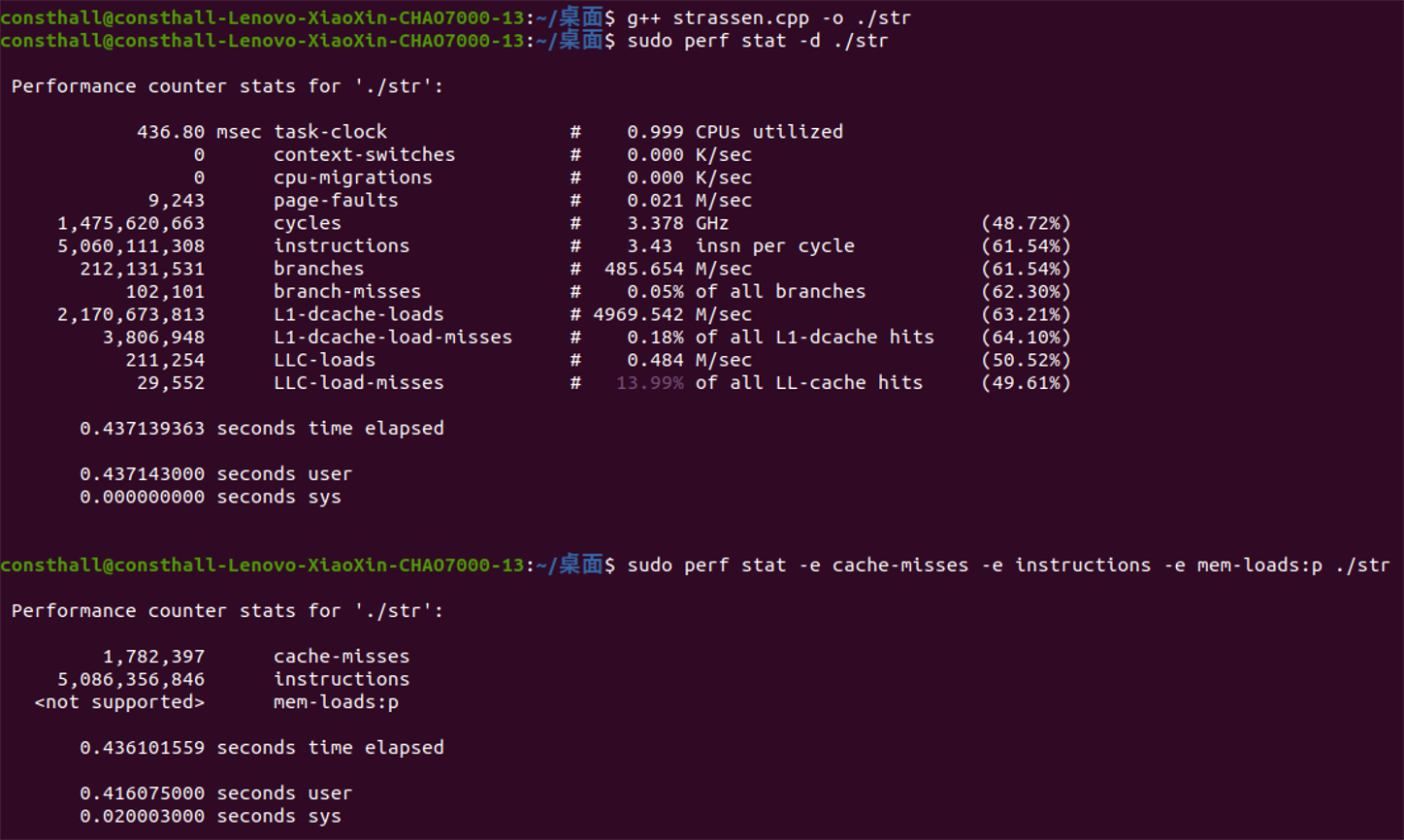
1. 实验结果

第1题的2种方法的实验结果如下所示：

（1）一般算法



（2）Strassen算法



由上面结果对比可知：

（1）CPI：图中给出的是IPC，IPC方面，一般 < Strassen，又因为CPI等于IPC的倒数，所以CPI方面，一般 > Strassen。

（2）周期数cycles和指令数instructions：均为一般 > Strassen，很明显是由于Strassen的优化使得这两者均小于一般算法。

（3）Cache-misses：一般 < Strassen，应该是因为Strassen算法不断生成新数组导致Cache-misses增大。

（4）mem-loads：经过多种尝试方法（使用双系统或者在虚拟机上运行）均失败，始终为<not supported>。

（5）运行时间：一般 > Strassen，很明显是由于Strassen的优化使得其时间比一般算法更短。

四、遇到的问题及解决方法

正如第三题所说，我上网查阅了很多资料，并且也参考了课程群里大家和老师的建议，均未能解决mem-load始终显示为<not supported>的问题，导致无法分析该指标，希望老师和助教可以在批改后在群里给出答案，或者给出一份优秀实验报告供我们学习参考。