并行与分布式计算作业

第4次作业

姓名：郝裕玮

班级：计科1班

学号：18329015

1. 问题描述

利用 Culler并行程序设计方法计算1000x1000的矩阵与1000x1的向量之间的乘积, 要求清晰地呈现 Culler 并行程序设计的四个步骤，并比较程序在不同阶段具有不同配置时如不同的子任务数量、不同的线程数量、不同的映射方案的性能差别。

1. 解决方案

Culler并行程序设计的4个步骤为：分解Decomposition，作业Assignment，编排Orchestration，映射Mapping

（1）分解Decomposition

本次任务可分解为将矩阵a(1000\*1000)第i行的元素与矩阵b(1000\*1)进行点乘即可得到结果矩阵c的第i行元素。即分解为1000个子任务。

（2）作业Assignment

Assignment主要是为了平衡工作量，降低沟通成本。每个线程可分配矩阵a上不同部分的等数量行数与矩阵b的点乘，如4个线程则各分配250行的运算，这里由OpenMP自动根据线程数和调度方式来进行分配。

（3）编排Orchestration

Orchestration主要是为了建立通信。由于OpenMP是共享内存式编程，所以我们只需将矩阵a,b,c和矩阵规模size设置成共享变量(shared)即可建立通信。

（4）映射Mapping

Mapping负责将线程映射到硬件执行单元。调度方式有静态调度，动态调度等方式，默认情况下是均匀分配的静态调度，即线程0—thread\_count-1均分配到n/thread\_count次迭代计算（n代表迭代总次数）。

不同线程数量下的代码如下所示（具体思路及详细分析均包含在代码注释中）详见Culler1.cpp：

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <omp.h>

#include <sys/time.h>

#include <time.h>

using namespace std;

#define GET\_TIME(now) { \

   struct timeval t; \

   gettimeofday(&t, NULL); \

   now = t.tv\_sec + t.tv\_usec/1000000.0; \

}

////最后一行将us转换为s，统一单位

//该结构体用于计算并行计算和串行计算的运行时间

void Serial(int \*\*a,int\*b,int\*c,int size){//串行计算

    int i,j;

    for(i=0;i<=size-1;i++){

        for(j=0;j<=size-1;j++){

            c[i]+=a[i][j]\*b[j];

            //将矩阵a(1000\*1000)第i行的元素与矩阵b(1000\*1)进行点乘即可得到结果矩阵c的第i行元素

        }

    }

}

void Parallel(int \*\*a,int\*b,int\*c,int size,int thread\_count){

    int i,j;

#  pragma omp parallel for num\_threads(thread\_count) default(none) \

      private(i,j) shared(a,b,c,size)

    //设置循环迭代的i,j为私密变量

    //设置a,b,c数组和数组大小size为各线程之间的共享变量

    for(i=0;i<=size-1;i++){

        for(j=0;j<=size-1;j++){

            c[i]+=a[i][j]\*b[j];

        }

    }

}

int main(){

    int size=1000;//数组大小(可修改)

    int i,j;

    int thread\_count=4;//并行线程数(可修改)

    //a,b,c数组的初始化

    int\*\* a;

    a=new int\*[size];

    for(i=0;i<=size-1;i++){

        a[i]=new int[size];

    }

    int \*b;

    b=new int[size];

    int \*c;

    c=new int[size];

    srand(time(NULL));//用时间初始化随机数生成种子

    //对a,b,c数组进行随机初始化(a,b数组的每个元素均为0-9的随机整数，c数组初始化为全0)

    for(i=0;i<=size-1;i++){

        for(j=0;j<=size-1;j++){

            a[i][j]=rand()%10;

        }

    }

    for(i=0;i<=size-1;i++){

        b[i]=rand()%10;

    }

    for(i=0;i<=size-1;i++){

        c[i]=0;

    }

    double t1,t2,t3,t4;

    //记录串行计算时间

    GET\_TIME(t1);

    Serial(a,b,c,size);

    GET\_TIME(t2);

    cout<<"串行计算时间为："<<t2-t1<<"秒"<<endl;

    //将结果数组c重新全部初始化为0

    for(i=0;i<=size-1;i++){

        c[i]=0;

    }

    //记录并行计算时间

    GET\_TIME(t3);

    Parallel(a,b,c,size,thread\_count);

    GET\_TIME(t4);

    cout<<"并行计算时间为："<<t4-t3<<"秒"<<endl;

    //计算加速比

    cout<<"加速比为："<<(t2-t1)/(t4-t3)<<endl;

    return 0;

}

不同子任务数量下的代码大体相同，只需修改Parallel函数（具体思路及详细分析均包含在代码注释中）详见Culler2.cpp：

void Parallel(int \*\*a,int\*b,int\*c,int size,int thread\_count){

    int i,j,cnt,div;

    div=10;//设置行数的合并数量

#  pragma omp parallel for num\_threads(thread\_count) default(none) \

      private(i,j,cnt) shared(a,b,c,size,div)

    //设置循环迭代的i,j,cnt为私密变量

    //设置a,b,c数组,数组大小size和行数合并数量div为各线程之间的共享变量

    for(cnt=0;cnt<=size-1;cnt+=div){//a矩阵的每div行合并到一起与矩阵b进行点乘运算，而不再单独计算a矩阵的每行

        for(i=cnt;i<=cnt+dvi-1;i++){//计算a矩阵当前的div行(cnt代表组数，div代表组内部的行数)

            for(j=0;j<=size-1;j++){

                c[i]+=a[i][j]\*b[j];

            }

        }

    }

}

不同映射方案下的代码大体相同，只需修改Parallel函数（具体思路及详细分析均包含在代码注释中），详见Culler3.cpp：

void Parallel(int \*\*a,int\*b,int\*c,int size,int thread\_count){

    int i,j;

#  pragma omp parallel for num\_threads(thread\_count) default(none) \

      private(i,j) shared(a,b,c,size) schedule(dynamic)//将调度方式修改为动态调度dynamic

    //设置循环迭代的i,j为私密变量

    //设置a,b,c数组和数组大小size为各线程之间的共享变量

    for(i=0;i<=size-1;i++){

        for(j=0;j<=size-1;j++){

            c[i]+=a[i][j]\*b[j];

        }

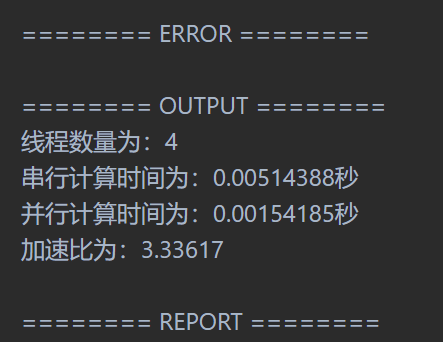
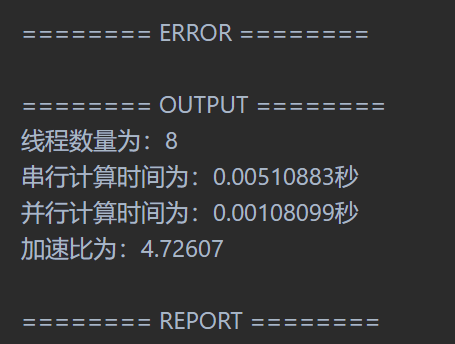
    }

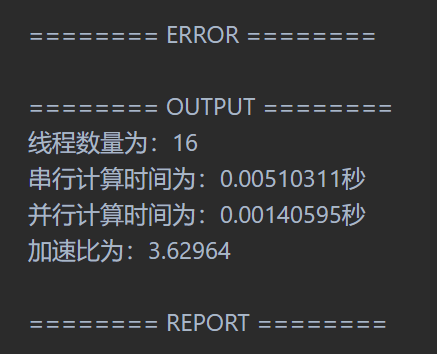
}

1. 实验结果

以下结果均在超算习堂上运行得出。

不同线程数量下的矩阵运算结果如下图所示：

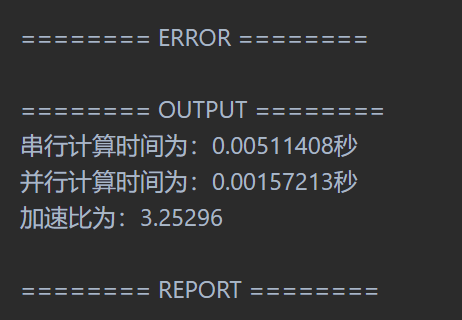
 



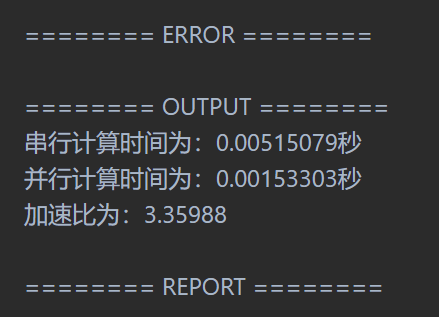
由上图结果可知，运算速度随着线程数量的增加先增加后降低。

不同子任务数量下的矩阵运算结果如下图所示：

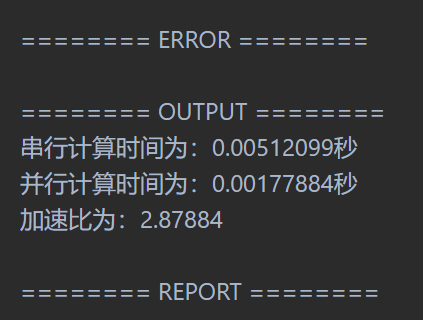
（1）子任务数量为1000/5=200（div=5）



（2）子任务数量为1000/20=50（div=20）



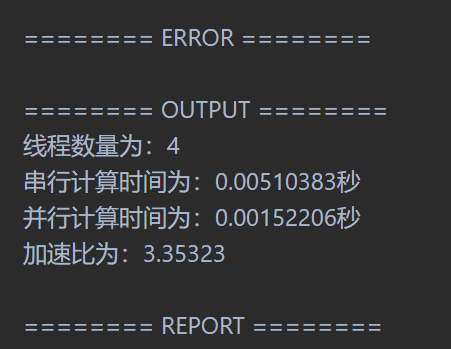
（3）子任务数量为1000/100=10（div=100）



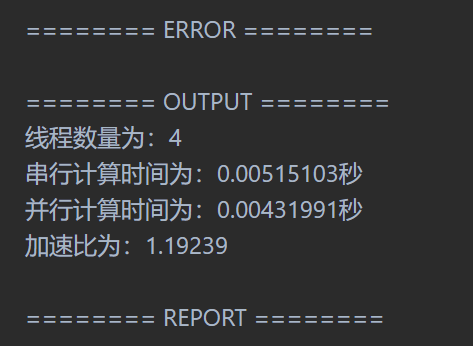
由上图结果可知，运算速度随着子任务数量的减少先增加后降低。

不同映射方案下的矩阵运算结果如下图所示：

（1）静态调度



（2）动态调度



由上图结果可知，运算速度：静态调度>动态调度。

四、遇到的问题及解决方法

本次实验唯一遇到的问题是自己在将串行程序用OpenMP进行并行化时少加了一个for，写成了下面这种形式：

#  pragma omp parallel num\_threads(thread\_count) default(none) \

      private(i,j) shared(a,b,c,size)

因为内部是双重循环，所以必须使用omp parallel for。所以我这种写法导致并行运算速度甚至慢于串行运算速度，在经过认真检查代码后找到了该问题，并得到了预想中的加速比。