**深 圳 大 学 实 验 报 告**

|  |  |
| --- | --- |
| **课程名称** | **：并行计算** |
| **实验项目名称** | **：矩阵乘法的OpenMP并行程序** |
| **学院** | **：计算机与软件学院** |
| **专业** | **：计算机科学与技术** |
| **指导教师** | **：陆克中** |
| **报告人** | **：郑雨婷** |
| **学号** | **：2021150122** |
| **实验时间** | **：2024年03月13日** |
| **实验报告提交时间** | **：2024年03月16日** |

**教务部制**

## 一、实验目的

1. 学会编写简单的OpenMP程序；

2. 掌握for编译制导语句；

3. 对并行程序进行简单的性能分析；

## 二、实验环境

1. 硬件环境：64核CPU、256GB内存的共享内存并行计算平台；

2. 软件环境：Ubuntu Linux、gcc、g++（g++ -O3 -fopenmp -o a.out a.cc）；

3. 远程登录：本地PowerShell中执行ssh bxjs@hpc.szu.edu.cn；

4. 传输文件：本地PowerShell中执行scp c:\a.cpp [bxjs@hpc.szu.edu.cn:/home/bxjs/](mailto:bxjs@hpc.szu.edu.cn:/home/bxjs/)或<ftp://hpc.szu.edu.cn>。

## 三、实验内容

1. 用OpenMP编写两个*n*阶方阵*a*和*b*的并行相乘程序，结果存放在方阵*c*中，其中矩阵乘法部分用for编译制导语句实现并行化操作。为了验证结果正确性，将矩阵乘法的串行计算结果存放在方阵*d*中，并比较是否与*c*相等。

2. 测试并行程序在不同线程数下的执行时间和加速比（与线程数=1时的执行时间相比）。其中，*n*固定为1000，线程数分别取1、2、4、8、16、32、64时，为减少误差，每项实验进行5次，取平均值作为实验结果。

## 四、代码描述

#include <omp.h>  
#include <iostream>  
#include <cmath>  
using namespace std;  
const int n = 1000;  
double a[n][n], b[n][n], c[n][n], d[n][n];  
int main()  
{  
 *//初始化矩阵a,b* int i, j, k;  
 for (i = 0; i < n; i++)  
 for (j = 0; j < n; j++)  
 {  
 a[i][j] = rand() \* 1.0 / **RAND\_MAX**;  
 b[i][j] = rand() \* 1.0 / **RAND\_MAX**;  
 }  
 for(int p=0;p<5;p++){  
 cout<<(p+1)<<":"<<endl;  
 *//串行化计算* for (i = 0; i < n; i++)  
 for (j = 0; j < n; j++){  
 d[i][j] = 0;  
 for (k = 0; k < n; k++)  
 d[i][j] += a[i][k] \* b[k][j];  
 }  
  
 *//并行化计算  
 //设置线程数* omp\_set\_num\_threads(4);  
 *//记录并行起始时间* double t0 = omp\_get\_wtime();  
 *//并行代码，编译制导语句，设置i,j,k为私有变量* #pragma omp parallel shared(a,b,c) private(i,j,k)  
 {  
 *//for编译制导语句* #pragma omp for  
 for (i = 0; i < n; i++)  
 for (j = 0; j < n; j++){  
 c[i][j]=0;  
 for (k = 0; k < n; k++)  
 c[i][j] += a[i][k] \* b[k][j];  
 }  
 }  
 *//记录并行结束时间* double t1 = omp\_get\_wtime();  
 cout<<"parallel computing time is "<<(t1-t0) <<" seconds"<<endl;  
 *//验证代码* int tag=0;  
 for(i=0;i<n;i++)  
 for (j = 0; j < n; j++)  
 if(fabs(c[i][j]-d[i][j])>=1e-5) tag=1;  
 if(tag) cout<<"wrong result!"<<endl;  
 }  
 return 0;  
}

## 五、实验结果和分析

*实验结果以及对实验结果的比较分析和综合概括。*

表1 并行程序在不同线程数下的执行时间（秒）和加速比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 线程数  执行时间 | 1 | 2 | 4 | 8 | 16 | 32 | 64 |
| 第1次 | 3.32847 | 2.4118 | 1.22255 | 0.722993 | 0.415708 | 0.332872 | 0.309056 |
| 第2次 | 3.39324 | 2.8624 | 1.26249 | 1.01838 | 0.423153 | 0.36023 | 0.274775 |
| 第3次 | 3.24923 | 2.66483 | 1.46476 | 0.77558 | 0.393571 | 0.401267 | 0.277313 |
| 第4次 | 3.24866 | 2.83298 | 1.36592 | 0.766309 | 0.397303 | 0.323721 | 0.275861 |
| 第5次 | 3.26721 | 2.42818 | 1.57578 | 0.802943 | 0.396515 | 0.361148 | 0.369751 |
| 平均值 | 3.297362 | 2.640038 | 1.3783 | 0.817241 | 0.40525 | 0.3558476 | 0.3013512 |
| 加速比 | 1 | 1.2489827 | 2.39234 | 4.034749 | 8.136612 | 9.26622 | 10.94192 |

在不同线程数下，我们观察到并行程序的执行时间以及相应的加速比如表1所示。可以看到，加速比不严格按照线程数那样成倍数增长。可能是因为并行化过程中会引入额外的开销，如线程创建和销毁、任务调度等。但在大体趋势上，依旧是随着线程数的增加，程序的执行时间呈现出递减的趋势，同时加速比也呈现出随线程数增加而增加的趋势。因此，并行计算能够提升程序的性能。

## 六、实验结论

通过本次实验，我学习并掌握了使用OpenMP编写简单并行程序的方法，并且了解了如何进行性能分析。实验结果表明，在适当的并行环境下，合理地利用并行计算资源可以有效提升程序的执行效率。在实验中，我们运用了OpenMP的编译制导语句来实现并行化操作，并对程序进行了性能分析，以便更好地理解并行计算的效果。通过这些实验，我们得出了一些客观结论，为进一步深入研究并行计算提供了基础。