实验一 多线程计算π，性能分析

* 1. 实验内容

多线程计算π，性能分析

1.2 实验原理



1.3实现方法

编程语言：c++

多线程：windows多线程thread

平台：windows 7600(32bit)，Lenovo T400 Laptop，Intel Core 2 Duo P8400 @ 2.26GHz

集成开发环境：Visual Studio Team System 2008(32bit)

1.4程序流程图

开始

设定线程数

设定累加数N

创建多线程并计时

i=thread\_num;

i+=thread

pSum+=pi(i)

**…**

i=thread\_num;

i+=thread

pSum+=pi(i)

i=thread\_num;

i+=thread

pSum+=pi(i)

i=thread\_num;

i+=thread

pSum+=pi(i)

线程归约

Sum+=pSum[n]

结束

计时结束

1.5实验结果

线程数 NUM\_THREAD=4

N π Time-cost

100 3.14160098692312 3ms

1000 3.14159273692313 4ms

10000 3.14159265442313 5ms

100000 3.14159265359813 25ms

1000000 3.14159265358990 82ms

1.6性能分析

精度随叠加次数N的增大而趋近于π的真实值，计算时间也随之增高；相同的叠加次数下，因为是双核处理器，线程数为2时计算性能最高。理论性能提升有极限值，所以不会因为线程的增多而性能无限增强。当线程数很大时，计算时间增加很快。

1.7总结展望

第一次编写并行化的程序，对多线程编程有了初步的认识。由于是在Visual Studio平台下编程，很多知识是从Lunix平台移植过来的，虽然表现形式有少许差别，但核心思想一致。通过学习，对windows多线程编程有了一定的掌握。

实验二 3PCF计算多线程实现

2.1实验内容

* 定义：
  + 点集D、R。
  + 定义D中的点为ai∈D，R中的点为bi∈R。
  + 距离：r1、r2、r3、err
* 求：
  + 满足以下条件的三元组（空间中三角形）的数目
    - <ai, bm, bn>，|ai-bm|=r1±err且|ai-bn|=r2±err且|bm-bn|=r3±err

2.2实验原理

对于D中每一点ai，在R中找到与之距离为r1的点集R’，找到与之距离为r2的点集R’’。在点集R’与R’’中，查找两点间距离为r3的点组数目。累加。 多线程中将D点分组，分别验证符合要求的点组，累加数目。

2.3实现方法

编程语言：c++

多线程：windows多线程thread

平台：windows 7600(32bit)，Lenovo T400 Laptop，Intel Core 2 Duo P8400 @ 2.26GHz

集成开发环境：Visual Studio Team System 2008(32bit)

2.4程序流程图

开始

设定线程数

设定累加数N

创建多线程并计时

i=thread\_num;

i+=thread

验证点组

符合就累加

**…**

i=thread\_num;

i+=thread

验证点组

符合就累加

i=thread\_num;

i+=thread

验证点组

符合就累加

i=thread\_num;

i+=thread

验证点组

符合就累加

线程归约

Count+=tempSum[n]

结束

计时结束

2.5实验结果

r1=0.4 r2=0.8 r3=0.7 err=0.2

D/R count Time-cost

32 3341 0

256 1828275 0.25

512 13908434 1.80

1024 115062333 14.56

2048 919041742 115.21

4096 7367826607 1048.49

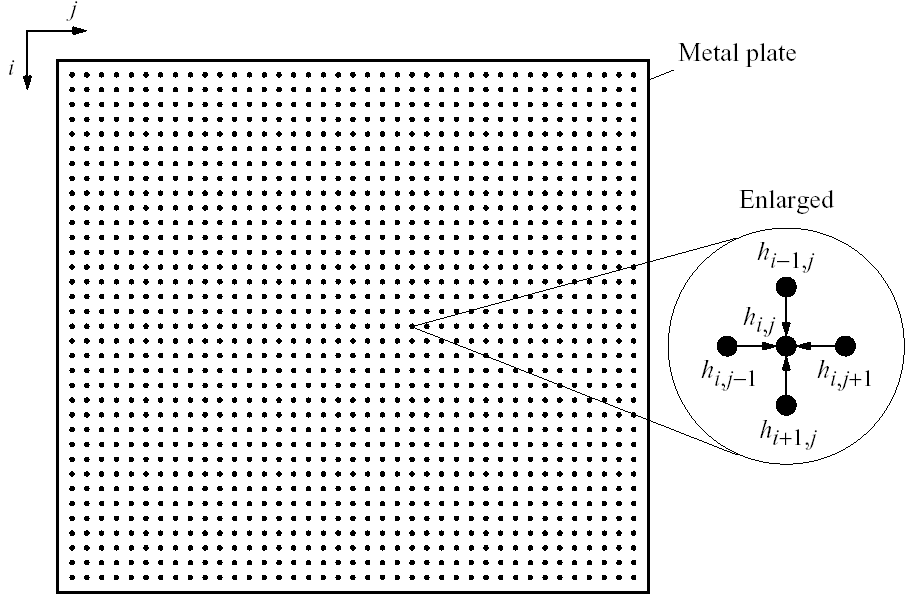
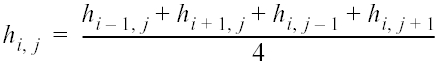
2.6性能分析

相同的线程数，点集增大时，计算时间增多；相同的点集，线程增多时，计算时间先减少后增多。这与理论相符，点集增大，计算量随之增加；线程增多，程序并行化提高，但由于计算单元有限，无限制增加线程会提高线程间通信成本，导致计算性能降低。

2.7总结展望

此次实验与第一次实验多线程实现方法相同，采用域划分，将相同的任务分配给不同的线程，将计算量均衡化，采用3PCF的基本思想，逐一验证三边，算法效率较低，可以设计新优化算法，利用矩阵乘法快速求出3PCF的结果，待实现。

实验三 MPI实现Jacobi迭代

3.1实验内容

3.2实验原理

通过MPI消息传递，实现并行计算。每个进程根据各自的Rank来划分数据域，并进行自己的迭代，覆盖原来的矩阵，进程之间互相通讯传递消息，继续循环迭代矩阵，直到达到迭代次数为止。将1024\*1024的矩阵初始化，将边界设置为10，不断进行迭代，可以看出矩阵向中央均衡化的过程，逐渐收敛。

3.3实现方法

编程语言：c++

多线程：MPI

平台：windows 7600(32bit)，Lenovo T400 Laptop，Intel Core 2 Duo P8400 @ 2.26GHz

集成开发环境：Visual Studio Team System 2008(32bit)

3.4程序流程图

Y

N

消息传递

消息传递

消息传递

开始

MPI初始化

分区

Jacobi迭代

分区

Jacobi迭代

分区

Jacobi迭代

分区

Jacobi迭代

覆盖原矩阵

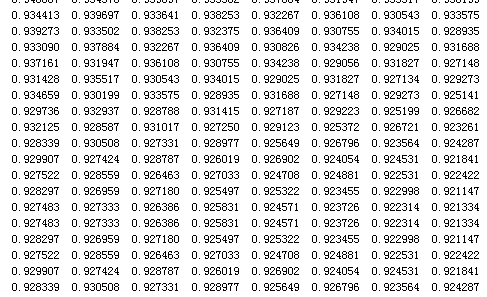
结束

循环结束

分别输出迭代结果

3.5实验结果

局部数据截图：



3.6总结展望

学习并实践了一种新的并行程序设计方法，掌握应用了MPI六个基本函数，对MPI模型有了基本了解。MPI是一种很好的实现并行计算的方法，并可应用于Cluster中，适合多台计算机或集群并行计算。由于时间有限，现只实现了MPI+VC的编程，未来将实现MPI在Lunix的应用。

实验四 基于MapRedue模型的WordCount算法

4.1实验内容

基于MapRedue模型的WordCount算法在Hadoop上的实现

4.2 实验原理

将数据源中的word和1作为map函数中的<key,value>对。map()生成一个或多个中间结果，以及与input相对应的一个output key。

reduce()函数将这些中间结果组合为多个对应于同一output key 的final value，将相同word中的value相加，最终的<key,value>代表key word出现了value次。

将基于Mapreduce的WordCount并行化，对每个文件分别Map,然后Reduce，之后将各个线程的结果最终归约，得到求解。

4.3实现方法

编程语言：java

平台：双i7四核，Ubuntu，SSH远程登录

4.4伪代码

map(String input\_key, String input\_value):

// input\_key: document name

// input\_value: document contents

for each word w in input\_value:

EmitIntermediate(w, "1");

reduce(String output\_key, Iterator intermediate\_values):

// output\_key: a word

// output\_values: a list of counts

int result = 0;

for each v in intermediate\_values:

result += ParseInt(v);

Emit(AsString(result));

4.5性能分析

将MapReduce并行化之后，每个线程处理数据域的文档，各自MapReduce之后将结果进行归约，得到最终的方案。计算量一定，线程数增多，计算时间先减少后增加，符合理论值。因为计算单元有限，所以超过一定数量后，通信成本过高，程序效率下降。即使有无限个计算单元，由于本身程序的可并行化结构所占比例一定，效率提升有上限。

4.6总结展望

了解并实践了hadoop的配置，实现了hadoop中MapReduce的WordCount功能。通过实践，对找<key,value>对有了一定的心得。MapReduce的关键是找好<key,value>对，这样之后的Map和Reduce才能顺利进行下去，得到想要的结果。之后有时间计划研究一下X10，并实现相关功能。