**系 统 结 构**

**多核环境下OpenMP并行编程**

学院： 计算机学院

组员： 何炜11121173

赵金亮11121170

毛雪慧11123416

教师： 沈文枫老师

1. 实验目的及要求

1) 在Linux平台上编译和运行OpenMP程序；

2) 在Windows平台上编译和运行OpenMP程序。

3) 掌握OpenMP并行编程基础。

1. 实验环境

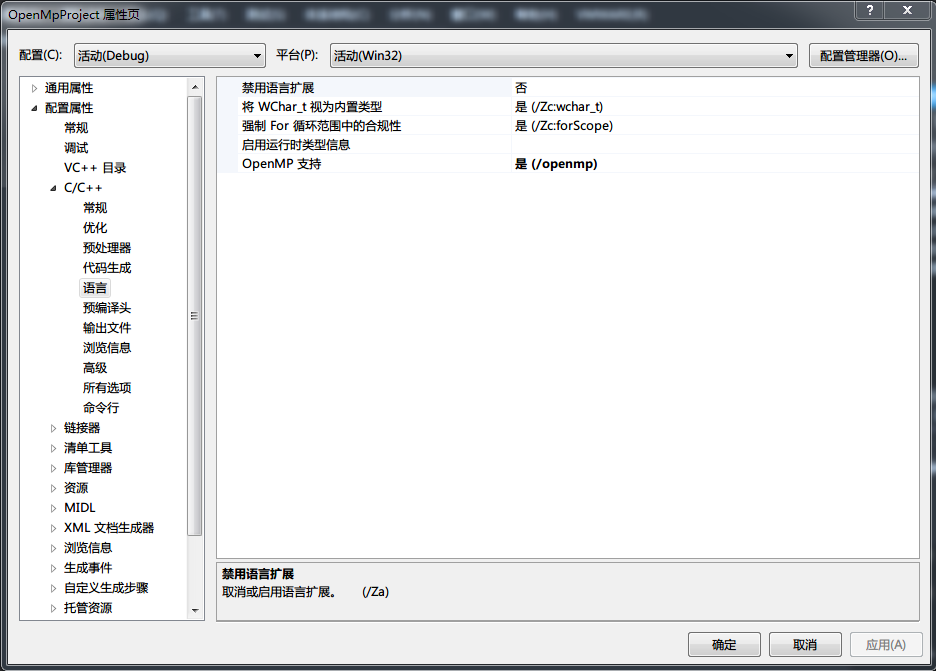
1) 硬件环境：计算机一台；

2) 软件环境：Linux、Windows、GCC、VS2012；

1. 实验内容及步骤
2. 配置实验环境——window环境
   1. 设置环境变量：

我的电脑 -> 属性 -> 高级 -> 环境变量，新建一个OMP\_NUM\_THREADS变量，值设为2，即为程序执行的线程数。

* 1. 设置VS2012环境，如截图所示：



1. 编写OpenMP程序

代码代码如下：  
 #include <omp.h>

#include <stdio.h>

int main(){

int nthreads,tid;

omp\_set\_num\_threads(8);

#pragma omp parallel private(nthreads,tid)

{

tid=omp\_get\_thread\_num();

printf("Hello World from OMP thread %d\n",tid);

if(tid==0){

nthreads=omp\_get\_num\_threads();

printf("Number of threads is %d\n",nthreads);

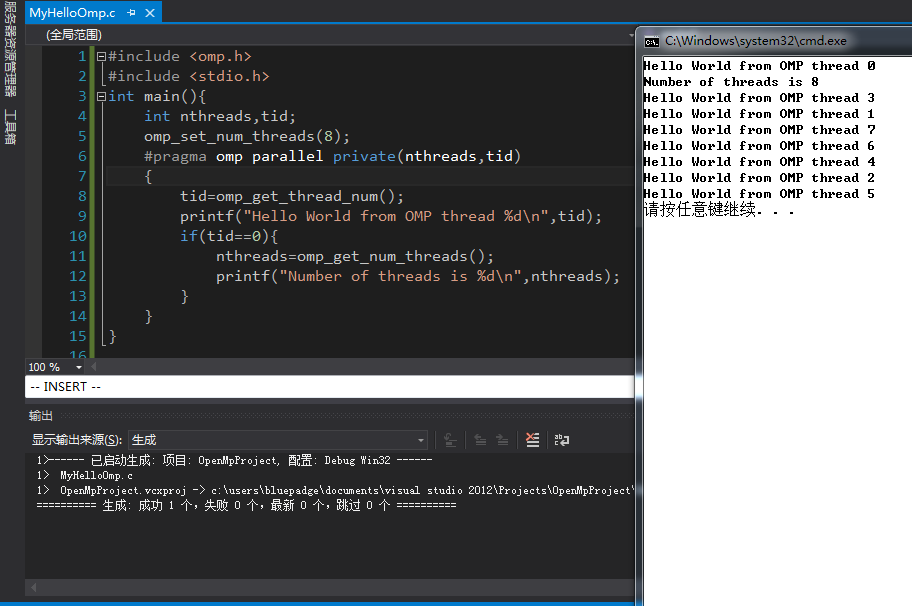
}

}

}

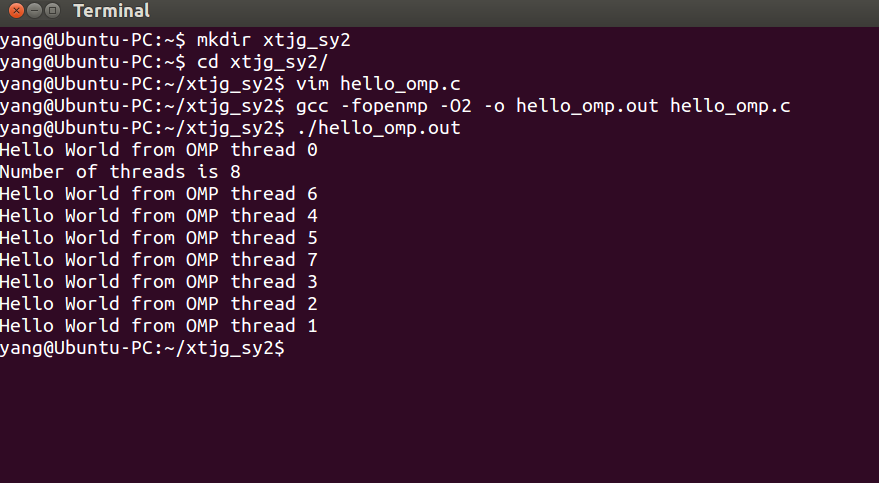
1. 实验结果及数据处理

Windows环境下的实验结果如图所示：

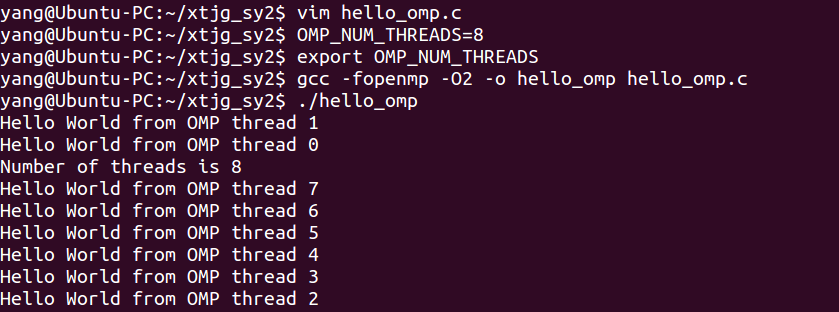


在Linux上的配置如下：

方法一  
(1).新建文件夹xtjg\_sy2  
(2).编译源代码  
(3).用gcc编译，生成源程序  
(4).运行



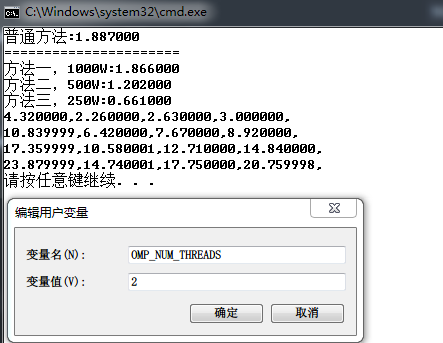
方法二  
1环境变量OMP\_NUM\_THREADS，并设置为8  
2去掉程序中设置线程数的语句  
3.再次用gcc编译，并运行生成的程序



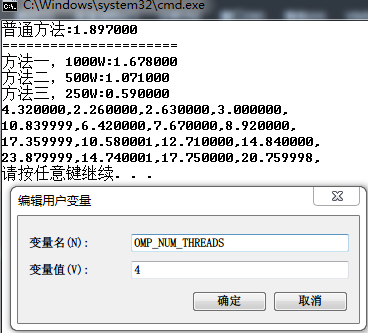
第二个实验：

1.程序在Windows上的执行结果：

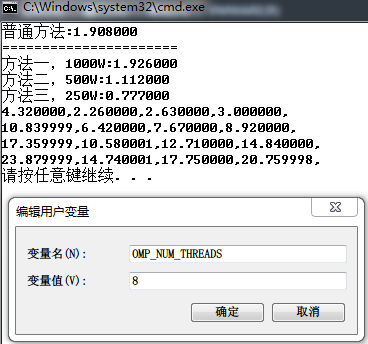
线程数：2



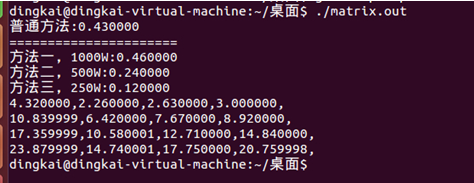
线程数：4



线程数：8



2. 在Linux虚拟机中再编译运行一遍程序，得到如下结果：



**3.** 分析矩阵相乘程序的执行时间、加速比和效率：方阵阶固定为100，节点数（线程数）分别取1、2、4、8时，为减少误差，每项实验进行3次，取平均值作为实验结果。

串行执行时程序的执行时间为：T = 0.599s

加速比=顺序执行时间/并行执行时间

效率=加速比/节点数

不同节点数下程序的执行时间（秒）

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 节点数  实验结果 | 1 | 2 | 4 | 8 |
| 第1次 | 0.594 | 0.151 | 0.183 | 0.281 |
| 第2次 | 0.592 | 0.156 | 0.167 | 0.227 |
| 第3次 | 0.595 | 0.158 | 0.174 | 0.237 |
| 平均值 | 0.594 | 0.155 | 0.175 | 0.248 |
| 加速比 | 1.0084 | 3.8645 | 3.4229 | 2.4153 |
| 效率 | 1.0084 | 1.9323 | 0.85573 | 0.3019 |

执行时间的分析：

随着节点数的增加，程序的执行时间减少，当节点数到了一定数目后，执行时间反而增加，这是由于计算量偏小时，线程分配所需的时间已经超过了多线程计算节省的时间。大概可以从结果中得出，随着节点书的增加一倍，执行时间减少近一半，当节点数达到一定数目时，时间反而增加。

加速比的分析：

在一定范围内，随着节点数的增加，程序的加速比增加，当节点数目增加一倍，加速相应的增加接近一倍。可是当超出多线程红利后，多开的线程反而影响了加速比。

效率的分析：

随着节点数的增加，程序的效率逐渐减少

**4.** 分析矩阵相乘程序的问题规模与效率的关系：固定节点数为4，让方阵阶从200到1600之间变化，每隔100取一个值。（注：循环次数是十）

相同节点数下不同问题规模程序的执行时间与效率（Windows版本）

|  |  |  |  |
| --- | --- | --- | --- |
| 方阵阶数 | 串行执  行时间 | 并行执  行时间 | 效率 |
| 200 | 0.467 | 0.226 | 0.516593 |
| 300 | 1.411 | 0.520 | 0.678365 |
| 400 | 3.870 | 1.227 | 0.788509 |
| 500 | 8.429 | 3.304 | 0.637788 |
| 600 | 18.234 | 6.982 | 0.652893 |
| 700 | 34.042 | 14.040 | 0.606161 |
| 800 | 60.038 | 20.524 | 0.731315 |
| 900 | 96.386 | 39.323 | 0.612784 |
| 1000 | 139.992 | 51.027 | 0.685872 |
| 1100 | 188.142 | 61.716 | 0.762128 |
| 1200 | 225.594 | 83.716 | 0.673688 |
| 1300 | 353.812 | 135.129 | 0.654582 |
| 1400 | 523.803 | 191.135 | 0.685122 |
| 1500 | 682.807 | 227.468 | 0.750443 |
| 1600 | 878.452 | 269.778 | 0.814055 |

Linux平台执行结果

|  |  |  |  |
| --- | --- | --- | --- |
| 方阵阶数 | 串行执  行时间 | 并行执  行时间 | 效率 |
| 200 | 0.650 | 0.410 | 0.396341 |
| 300 | 2.180 | 1.37 | 0.397810 |
| 400 | 5.230 | 3.250 | 0.402308 |
| 500 | 10.340 | 6.440 | 0.401398 |
| 600 | 18.410 | 11.490 | 0.400566 |
| 700 | 35.020 | 21.930 | 0.399983 |
| 800 | 59.240 | 36.950 | 0.400812 |
| 900 | 91.740 | 57.340 | 0.399983 |
| 1000 | 133.270 | 83.070 | 0.401077 |
| 1100 | 181.410 | 113.43 | 0.399828 |
| 1200 | 245.720 | 153.060 | 0.401346 |
| 1300 | 317.550 | 198.050 | 0.400846 |
| 1400 | 403.190 | 250.820 | 0.401872 |
| 1500 | 497.620 | 311.660 | 0.399169 |
| 1600 | 604.480 | 377.320 | 0.400509 |

1. 分析与讨论

通过矩阵计算，我们可以看出：

在一定范围内：随着节点数的增加，程序的执行时间减少，当节点数到了一定数目后，执行时间反而增加，这是由于计算量偏小时，线程分配所需的时间已经超过了多线程计算节省的时间。大概可以从结果中得出，随着节点书的增加一倍，执行时间减少近一半，当节点数达到一定数目时，时间反而增加。

对于加速比：在一定范围内，随着节点数的增加，程序的加速比增加，当节点数目增加一倍，加速相应的增加接近一倍。可是当超出多线程红利后，多开的线程反而影响了加速比。对于效率：随着节点数的增加，程序的效率逐渐降低。

1. 实验心得

OMP是一种比较简单的让程序获得并行化效果的方法。OpenMP实现起来比较简单，仅需要通过预编译指令以及一些简单的库函数就可以完成复杂的并行计算。这是OpenMP最大的优点。它降低了开发并行程序的门槛，为程序员提供了便利。并起其他的并行编程模型，它显得简单易上手。在今天这种，通过OMP并行化的处理，可以让需要大量计算的程序得到较大的性能上的提升。在现在这种多处理器十分普及的年代里，这是十分有意义的。不过，当计算规模非常小的时候，不适合开过多线程计算，另外需要注意的是程序中设置的线程数不宜超过实际物理机器太多，否则线程切换的开销将会上升，程序将得不到明显的性能提升。