并行程序设计与算法实验

5-基于OpenMP的并行矩阵乘法

**提交格式说明**

按照实验报告模板填写报告，需要提供源代码及代码描述至https://easyhpc.net/course/221。实验报告模板使用PDF格式，命名方式为“并行程序设计\_学号\_姓名”。如有疑问，在课程群（群号1021067950）中询问细节。

1. OpenMP通用矩阵乘法

使用OpenMP实现并行通用矩阵乘法，并通过实验分析不同进程数量、矩阵规模、调度机制时该实现的性能。

**输入：**三个整数，每个整数的取值范围均为[128, 2048]

**问题描述：**随机生成的矩阵及的矩阵，并对这两个矩阵进行矩阵乘法运算，得到矩阵.

**输出**：三个矩阵，及矩阵计算所消耗的时间。

**要求：**使用OpenMP多线程实现并行矩阵乘法，设置不同线程数量（1-16）、矩阵规模（128-2048）、调度模式（默认、静态、动态调度），通过实验分析程序的并行性能。

2. 构造基于Pthreads的并行for循环分解、分配、执行机制

模仿OpenMP的omp\_parallel\_for构造基于Pthreads的并行for循环分解、分配及执行机制。此部分可在下次实验报告中提交。

**问题描述：**生成一个包含parallel\_for函数的动态链接库（.so）文件，该函数创建多个Pthreads线程，并行执行parallel\_for函数的参数所指定的内容。

**函数参数：**parallel\_for函数的参数应当指明被并行循环的索引信息，循环中所需要执行的内容，并行构造等。以下为parallel\_for函数的基础定义，实验实现应包括但不限于以下内容：

**parallel\_for**(int start, int end, int inc,

void \*(\*functor)( int,void\*), void \*arg, int num\_threads)

* start, end, inc分别为循环的开始、结束及索引自增量；
* functor为函数指针，定义了每次循环所执行的内容；
* arg为functor的参数指针，给出了functor执行所需的数据；
* num\_threads为期望产生的线程数量。
* 选做：除上述内容外，还可以考虑调度方式等额外参数。

**示例：**给定functor及参数如下：

struct **functor\_args** {

    float \*A, \*B, \*C;

};

void \***functor**(int idx, void\* args){

    functor\_args \*args\_data = (functor\_args\*) args;

    args\_data->C[idx] = args\_data->A[idx] + args\_data->B[idx];

}

调用方式如下：

functor\_args args = {A, B, C};

**parallel\_for**(0, 10, 1, functor, (void\*)&args, 2)

该调用方式应当能产生两个线程，并行执行functor完成数组求和（）。当不考虑调度方式时，可由前一个线程执行任务{0,1,2,3,4}，后一个线程执行任务{5,6,7,8,9}。也可以实现对调度方式的定义。

**要求：**完成parallel\_for函数实现并生成动态链接库文件，并以矩阵乘法为例，测试其实现的正确性及效率。

**提示：**基于pthreads的多线程库提供的基本函数，如线程创建、线程join、线程同步等，构建parallel\_for函数，该函数实现对循环分解、分配和执行机制，函数参数包括但不限于(int start, int end, int increment, void \*(\*functor)(void\*), void \*arg , int num\_threads)；其中start为循环开始索引；end为结束索引；increment每次循环增加索引数；functor为函数指针，指向被并行执行的循环代码块；arg为functor的入口参数；num\_threads为并行线程数。

2）在Linux系统中将parallel\_for函数编译为.so文件，由其他程序调用。

3）将通用矩阵乘法的for循环，改造成基于parallel\_for函数并行化的矩阵乘法，注意只改造可被并行执行的for循环（例如无race condition、无数据依赖、无循环依赖等）。

举例说明：

将串行代码：

for ( int i = 0; i < 10; i++ ){

A[i]=B[i] \* x + C[i]

}

替换为------>

parallel\_for(0, 10, 1, functor, NULL, 2);

struct for\_index {

int start;

int end;

int increment;

}

void \* functor (void \* args){

struct for\_index \* index = (struct for\_index \*) args;

for (int i = index->start; i < index->end; i = i + index->increment){

A[i]=B[i] \* x + C[i];

}

}

==========================

编译后执行阶段：

多线程执行

在两个线程情况下：

Thread 0: start和end分别为0，5

void \* funtor(void \* arg){

struct for\_index \* index = (struct for\_index \*) args;

for (int i = index->start; i < index->end; i = i + index->increment){

A[i]=B[i] \* x + C[i];

}

}

Thread 1: start和end分别为5，10

Thread 2: ......

……