中山大学计算机学院本科生实验报告

（2021学年秋季学期）

课程名称：高性能计算程序设计基础　　批改人：

实验	**LAB4**	专业	计算机科学与技术（超算）
学号	19335162	姓名	潘思晗
Email	pansh25@mail2.sysu.edu.cn	完成日期	2021.11.07

一、实验目的

> *1.*通过 *OpenMP*实现通用矩阵乘法（*Lab1*）的并行版本，*OpenMP*并行线程从*1*增加至*8*，矩阵规模从*512*增加至*2048*。
>
> *2.*分别采用 *OpenMP*的默认任务调度机制、静态调度*schedule(static, 1)*和动态调度*schedule(dynamic,1)*，调度*#pragma omp for*的并行任务，并比较其性能。
>
> *3.*构造基于*Pthreads*的并行*for*循环分解、分配和执行机制
>
> - 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*函数对循环分解、分配和执行机制
> - 在 *Linux*系统中将 *parallel_for*函数编译为 *.so*文件，由其他程序调用
> - 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

二、实验过程和核心代码

1.通过OpenMP实现通用矩阵乘法

（1）算法描述

通用矩阵乘法通常定义为: $C = AB$， $C_{m,n} = \sum_{n=1}^{N} A_{m,n}B_{n,k}$

串行版本`GEMM()`函数实现：传入矩阵A、B并为矩阵C申请空间，通过循环嵌套实现矩阵的相乘，最后返回计算结果的指针，如下所示

```
int** GEMM(int** matA,int** matB
{
    int** ans = (int**)malloc(sizeof(int*)*m);
    for(int i=0;i<m;i++){
        ans[i]=(int*)malloc(sizeof(int)*k);
    }
    for(int i=0;i<m;i++){
        for(int j=0;j<k;j++){
            ans[i][j] = 0;
            for(int t=0;t<n;t++){
                ans[i][j] += matA[i][t]*matB[t][j];
            }
        }
    }
    return ans;
}
```

（2）核心代码

使用子句`#pragma omp parallel for`开始并行，表示接下来的这个for循环将被多个线程同时运行的，也就是多个线程将同时运行该for循环，变量sum和t声明为 private，表示每个线程的循环都有单独的sum和t变量，这样就完成了并行化。

```
for (int i = 0; i < m;i++){
#pragma omp privite(sum,t) parallel for
    for (int j = 0; j < n;j++){
        double sum = 0;
        for (int t = 0; t < k; t++)
            sum += matA[i][t] * matB[t][j];
        matC[i][j] = sum;
    }
}
```

完成后标记时间并计算得到运行总时长，最后打印出三个矩阵A、B、C以及完成矩阵乘法所用时间。

2.基于OpenMP的通用矩阵乘法优化

2-1 默认任务调度

task-1的实现即为默认任务调度

```
for (int i = 0; i < m;i++){
#pragma omp privite(sum,t) parallel for
    for (int j = 0; j < n;j++){
        double sum = 0;
        for (int t = 0; t < k; t++)
            sum += matA[i][t] * matB[t][j];
        matC[i][j] = sum;
    }
}
```

2-2 静态调度

修改并行指令加上`schedule(static, 1)`

```
begin=clock();
#pragma omp parallel for schedule(static,1)
for (int i = 0; i < m;i++){
#pragma omp privite(sum,t) parallel for schedule(static,1)
    for (int j = 0; j < n;j++){
        double sum = 0;
        for (int t = 0; t < k; t++)
            sum += matA[i][t] * matB[t][j];
        matC[i][j] = sum;
    }
}
end=clock();
time=(double)(end-begin)/CLOCKS_PER_SEC;
printf("THE TIME OF STATIC: %f s\n",time);
```

打印静态调度算法时间。

2-3 动态调度

修改并行指令加上`schedule(dynamic, 1)`

```
begin=clock();
#pragma omp parallel for schedule(dynamic,1)
for (int i = 0; i < m;i++){
#pragma omp privite(sum,t) parallel for schedule(dynamic,1)
    for (int j = 0; j < n;j++){
        double sum = 0;
        for (int t = 0; t < k; t++)
            sum += matA[i][t] * matB[t][j];
        matC[i][j] = sum;
    }
}
end=clock();
time=(double)(end-begin)/CLOCKS_PER_SEC;
printf("THE TIME OF STATIC: %f s\n",time);
```

打印动态调度算法时间。

3.构造基于Pthreads的并行for循环分解、分配和执行机制

（1）改造文件

函数头文件

函数头文件中，不包含`parallel_for()`函数的实现，仅包含其定义和库文件

```
#ifndef PARALLEL_FOR_H
#define PARALLEL_FOR_H
void parallel_for(int start, int end, int increment, void *
(*functor)(void*), void *arg, int num_thread);
#endif
```

该文件命名为`parallel_for.h`。

函数实现文件

函数的实现文件与头文件同名，命名为`parallel_for.cpp`，算法的实现如下所示

```
#include<stdlib.h>
#include <pthread.h>
#include "parallel_for.h"

struct for_index
{
    int start;
    int end;
    int increment;
};

void parallel_for(int start, int end, int increment, void *
(*functor)(void *), void *arg, int num_thread)
{
    long thread;
    pthread_t *thread_handles;
    thread_handles=malloc(num_thread*sizeof(pthread_t));

    for (thread = 0; thread < num_thread; thread++)
    {
        int my_rank = thread;
        int my_first, my_last;
        int quotient = (end - start) / num_thread;
        int remainder = (end - start) % num_thread;
        int my_count;
        if (my_rank < remainder)
        {
            my_count = quotient + 1;
            my_first = my_rank * my_count;
        }
        else
        {
            my_count = quotient;
            my_first = my_rank * my_count + remainder;
        }
        my_last = my_first + my_count;
        struct for_index *index;
        index = malloc(sizeof(struct for_index));
        index->start = start + my_first;
```

```
39          index->end = start + my_last;
40          index->increment = increment;
41          pthread_create(&thread_handles[thread], NULL, functor,
    index);
42      }
43
44      for (thread = 0; thread < num_thread; thread++)
45      {
46          pthread_join(thread_handles[thread], NULL);
47      }
48      free(thread_handles);
49  }
```

调用函数的文件此处将其命名为`main.cpp`，在此文件中调用函数`parallel_for()`，并输出相应结果。

注意实现文件中都要包含函数头文件`parallel.h`。

（2）生成动态链接库

首先编译`.c`文件，生成`.o`文件，编译命令如下：

```
1  gcc -c -fPIC -o parallel_for.o parallel_for.c
```

然后使用`-shared`参数生成`.so`动态链接库：

```
1  gcc -shared -o lib_parallel.so parallel_for.o -lpthread
```

载入动态链接库，`-L`参数指明库文件所在路径，由于我将其放在同一文件夹下，因此使用`-L.`参数表示当前路径，命令如下：

```
1  gcc main.c -L. -l_parallel -o  main
```

最后将`.so`文件移动至`/usr/lib`，从而能在不同位置的程序中调用该动态库：

```
1  sudo mv lib_parallel.so /usr/lib
```

三、实验结果

打印CPU相关信息如下

```
emilylyly@emilylyly-VirtualBox:~$ lscpu
架构:                    x86_64
CPU 运行模式:            32-bit, 64-bit
字节序:                  Little Endian
CPU:                     1
在线 CPU 列表:           0
每个核的线程数:          1
每个座的核数:            1
座:                      1
NUMA 节点:               1
厂商 ID:                 GenuineIntel
CPU 系列:                6
型号:                    142
型号名称:                Intel(R) Core(TM) i7-8565U CPU @ 1.80GHz
步进:                    11
CPU MHz:                 1992.002
BogoMIPS:                3984.00
超管理器厂商:            KVM
虚拟化类型:              完全
L1d 缓存:                32K
L1i 缓存:                32K
L2 缓存:                 256K
L3 缓存:                 8192K
NUMA 节点0 CPU:          0
标记:                    fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
freq pni pclmulqdq monitor ssse3 cx16 pcid sse4_1 sse4_2 x2apic movbe
_l1d arch_capabilities
```

1.通过OpenMP实现通用矩阵乘法

测试时间

编译和运行程序的命令如下：

```
gcc  -g openmp_GEMM.c -o  openmp_GEMM -fopenmp
./openmp_GEMM
```

令矩阵规模从 512 增加到 2048，测试结果如下所示：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./openmp_GEMM
ENTER 3 INTERGERS (512~2048) :512 512 512
THE TIME OF OPENMP_GEMM: 0.702113 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./openmp_GEMM
ENTER 3 INTERGERS (512~2048) :1024 1024 1024
THE TIME OF OPENMP_GEMM: 15.309368 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./openmp_GEMM
ENTER 3 INTERGERS (512~2048) :2048 2048 2048
THE TIME OF OPENMP_GEMM: 181.195884 s
```

对比之前实现的pthread_GEMM：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab3$ ./pthread_GEMM 512 512 512
THE TIME OF PTHREAD_GEMM: 4.724121 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab3$ ./pthread_GEMM 1024 1024 1024
THE TIME OF PTHREAD_GEMM: 27.687084 s
```

可以看到，使用openmp并行算法的性能优于使用pthread，也比直接使用朴素通用矩阵乘法要迅速。

2.基于OpenMP的通用矩阵乘法优化

测试时间对比

编译和运行程序的命令如下：

```
gcc  -g schedule.c -o  schedule -fopenmp
./schedule
```

对比矩阵规模不同情况下三种调度方法的性能：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ gcc  -g schedule.c -o  schedule -fopenmp
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./schedule
ENTER 3 INTERGERS (512~2048) :512 512 512
THE TIME OF DEFAULT: 0.786257 s
THE TIME OF STATIC: 0.814005 s
THE TIME OF DYNAMIC: 0.676840 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./schedule
ENTER 3 INTERGERS (512~2048) :1024 1024 1024
THE TIME OF DEFAULT: 15.606461 s
THE TIME OF STATIC: 15.680167 s
THE TIME OF DYNAMIC: 14.512151 s
```

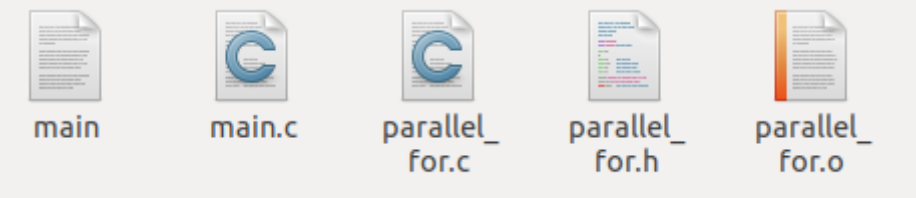
从测试情况来看，总体上三种调度算法所用时间差不多，动态调度略微有些优势，可能是动态调度算法使负载更均衡，因此性能更好。

3.构造基于Pthreads的并行for循环分解、分配和执行机制

首先执行下列命令

```
gcc -c -fPIC -o parallel_for.o parallel_for.c
gcc -shared -o lib_parallel.so parallel_for.o -lpthread
gcc main.c -L. -l_parallel -o  main
sudo mv lib_parallel.so /usr/lib
```

执行之后可以看到下列文件（以及移动到`/usr/lib`的`.so`文件）



使用`ldd`命令判断动态链接库是否链接成功

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ldd ./main
        linux-vdso.so.1 (0x00007ffca7be8000)
        lib_parallel.so => /usr/lib/lib_parallel.so (0x00007ff15b739000)
        libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007ff15b348000)
        libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007ff15b129000)
        /lib64/ld-linux-x86-64.so.2 (0x00007ff15bb3e000)
```

如图所示，动态连接成功。

下面测试程序`main`能否正常调用`parallel_for`函数，使用命令：

```
./main <number of threads>
```

得到结果如下：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./main 4
ENTER 3 INTERGERS (512~2048) :512 512 512
THE TIME OF OPENMP_GEMM: 1.396776 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./main 4
ENTER 3 INTERGERS (512~2048) :1024 1024 1024
THE TIME OF OPENMP_GEMM: 10.953008 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./main 8
ENTER 3 INTERGERS (512~2048) :512 512 512
THE TIME OF OPENMP_GEMM: 0.873574 s
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ ./main 8
ENTER 3 INTERGERS (512~2048) :1024 1024 1024
THE TIME OF OPENMP_GEMM: 10.393214 s
```

程序能够成功运行，因此封装和调用动态库成功。

四、实验感想

这次实验使用openmp对矩阵乘法做了改变，重温了编译链接库函数的知识，总体而言实验过程对比之前要顺利一些，但也遇到了部分问题，比如在编译`.o`文件时遇到：

`error: invalid application of ‘sizeof’ to incomplete type ‘struct for_index’`

错误原因：

sizeof不能用在extern变量，sizeof 的计算发生在代码编译时刻，而extern标注的符号在链接的时刻解析，所以 sizeof 不知道这个符号到底占用了多少空间

解决方法：

在parallel_for.c文件中增加struct for_index的定义。

之后再编译，没有报错：

```
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ gcc -c -fPIC -o parallel_for.o parallel_for.c
parallel_for.c: In function ‘parallel_for’:
parallel_for.c:31:31: error: invalid application of ‘sizeof’ to incomplete type ‘struct for_i
ndex’
         index = malloc(sizeof(struct for_index));
                               ^~~~~~
parallel_for.c:32:14: error: dereferencing pointer to incomplete type ‘struct for_index’
         index->start = start + my_first;
              ^~
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ gcc -c -fPIC -o parallel_for.o parallel_for.c
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ gcc -shared -o lib_parallel.so parallel_for.o
-lpthread
emilylyly@emilylyly-VirtualBox:~/HPC_lab/lab4$ gcc main.c -L. -l_parallel -o  main
```

总体而言，通过这次实验，我对于openmp程序的编写有了新的理解，也学到了很多，有了新的体会。

五、参考资料

1.https://www.cnblogs.com/xudong-bupt/p/3622101.html

2.https://blog.csdn.net/whatday/article/details/93202205

3.https://blog.csdn.net/u012206617/article/details/94383568