中山大学计算机学院本科生实验报告

年级	2019级	专业（方向）	计算机科学与技术（超算）
学号	19335074	姓名	黄玟瑜
开始日期	2021年4月11日	完成日期	2021年4月11日

实验题目

用pthread中的semaphore计算π的值。

问题分析

计算π的公式如下：

$$\pi = (1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \ldots + (-1)^n \frac{1}{2n+1})$$

根据上式编写代码，串行运算代码如下：

```
    double sum = 0.0;
    double factor = 1.0;

    for (i = 0; i < n; i++, factor = -factor) {
       sum += factor/(2*i+1);
    }
    return 4.0*sum;
```

并行化这个程序：将for循环分块后交给各个线程处理，并将sum设为全局变量。

各个线程分别计算完自己范围内（my_first_i~my_last_i）的和后再汇总到全局变量sum中。

随着n的增加，估算结果也越来越准确。然而，当n值较大时，多线程的计算结果反而变糟：多次运行同一个多线程程序，尽管n未变，但每次的结果都不同，这是因为当多个线程尝试更新同一个共享变量时，结果可能是无法预测的。

对如下语句：

```
sum+=my_sum;
```

机器的处理过程一般比这个式子更加复杂。因为sum和my_sum的值都存储在计算机的主存中，无法直接进行加法运算，需要先将它们从主存加载到CPU的寄存器中后，才能进行加法运算。当运算完成后，必须将结果再从寄存器重新存储到主存中。

为了防止多个线程同时对sum进行修改导致错误的结果产生，需要保证一旦某个线程开始执行`sum+=my_sum;`，其他线程在他未完成前不能执行操作。

`sum+=my_sum;`成为了该程序的临界区。

为使结果正确，考虑使用信号量来实现路障。

涉及到的函数如下：

sem_init函数

该函数用于创建信号量，其原型如下：

```
int WINPTHREAD_SEMA_API sem_init(sem_t * sem, int pshared, unsigned int
value);
```

其中：

- sem为指向信号量结构的一个指针；
- pshared不为 0 时此信号量在进程间共享，否则只能为当前进程的所有线程共享；
- value给出了信号量的初始值。

调用成功时返回0，失败返回-1。

sem_wait函数

该函数用于以原子操作的方式将信号量的值减1。原子操作就是，如果两个线程企图同时给一个信号量加1或减1，它们之间不会互相干扰。它的原型如下：

```
int WINPTHREAD_SEMA_API sem_wait(sem_t *sem);
```

sem指向的对象是由sem_init调用初始化的信号量。调用成功时返回0，失败返回-1。

sem_post函数

该函数用于以原子操作的方式将信号量的值加1。它的原型如下：

```
int WINPTHREAD_SEMA_API sem_post(sem_t *sem);
```

sem指向的对象是由sem_init调用初始化的信号量。调用成功时返回0，失败返回-1。

定义以下几个全局变量：

- **thread_count** 线程数目
- **n** 级数项数
- **sum** 全局总和
- **sem** 信号量

定义以下几个并行程序中的局部变量：

- **factor** 当前项系数
- **my_sum** 局部总和
- **my_n** 局部项数
- **my_first_i** 起始项
- **my_last_i** 终止项

代码解释

```
    double factor;
    double my_sum;
    long long i;
    long long my_n=n/thread_count;
    long long my_first_i=my_n*my_rank;
    long long my_last_i=my_first_i+my_n;

```

定义变量。

局部项数为当前线程需要处理的项的数目，大小为项数总和n除以线程数目thread_count；

起始项为局部项数my_n乘以当前线程号my_rank；

终止项为起始项加上局部项的数目。

```
    if(my_first_i%2==0)
       factor=1.0;
    else
       factor=-1.0;

    for(i=my_first_i;i<my_last_i;i++, factor=-factor){
       my_sum+=factor/(2*i+1);
    }
```

若当前项为偶数项，则系数因子为1.0，若当前项为奇数项，则系数因子为-1.0；

用for循环求出每一项，并将其累加到my_sum中。

```
    if(my_rank==0){
       sum+=my_sum;
       sem_post(&sem);
    }else{
       sem_wait(&sem);
       sum+=my_sum;
       sem_post(&sem);
    }
```

为了使代码产生正确结果，需要保证一旦某个线程开始执行`sum+=my_sum;`，其他线程在他未完成前不能执行操作，使主线程作为第一个进入该临界区的线程，其他线程再执行`sum+=my_sum;`前需先等待`sem_wait(&sem);`，拿到“令牌”后执行`sum+=my_sum;`，执行完毕后释放`sem_post(&sem);`。

实验结果

打开超算习堂>在线编程，上传代码，选择"pthread"，输入编译参数和运行参数：

编译参数：

`-g` 表示向生成的文件中加入debug信息供gdb使用

`-Wall` 告诉编译器显示所有编译器警告信息

`-lm` 表示连接系统的数学库libm.a

`-lpthread` 表示链接到pthread的库

计算到n=10000时，输出如下：

```
======== OUTPUT ========
With n = 10000 terms,
Our estimate of pi = 3.141492653590044
The elapsed time is 3.399849e-04 seconds
Single thread est = 3.141492653590034
The elapsed time is 6.294250e-05 seconds
pi = 3.141592653589793
```

并行程序误差 $E_{parallel} = 9.999999974930063 \times 10^{-5}$

串行程序误差 $E_{serial} = 9.999999975907059 \times 10^{-5}$

并行程序和串行程序结果之差：$1.021405182655144 \times 10^{-14}$

计算到n=100000时，输出如下：

```
======== OUTPUT ========
With n = 100000 terms,
Our estimate of pi = 3.141582653589787
The elapsed time is 5.559921e-04 seconds
Single thread est = 3.141582653589720
The elapsed time is 6.368160e-04 seconds
pi = 3.141592653589793
```

并行程序误差 $E_{parallel} = 1.000000000628276 \times 10^{-5}$

串行程序误差 $E_{serial} = 1.000000007289614 \times 10^{-5}$

并行程序和串行程序结果之差：$6.661338147750939 \times 10^{-14}$

计算到n=1000000时，输出如下：

```
======== OUTPUT ========
With n = 1000000 terms,
Our estimate of pi = 3.141591653589781
The elapsed time is 1.938820e-03 seconds
Single thread est = 3.141591653589774
The elapsed time is 6.417036e-03 seconds
pi = 3.141592653589793
```

并行程序误差 $E_{parallel} = 1.000000012130187 \times 10^{-6}$

串行程序误差 $E_{serial} = 1.000000019235614 \times 10^{-6}$

并行程序和串行程序结果之差：$7.105427357601002 \times 10^{-15}$

计算到n=10000000时，输出如下：

```
======== OUTPUT ========
With n = 10000000 terms,
Our estimate of pi = 3.141592553589743
The elapsed time is 1.633406e-02 seconds
Single thread est = 3.141592553589792
The elapsed time is 6.400108e-02 seconds
pi = 3.141592653589793
```

并行程序误差 $E_{parallel} = 1.000000500184228 \times 10^{-7}$

串行程序误差 $E_{serial} = 1.000000011686097 \times 10^{-7}$

并行程序和串行程序结果之差：$4.884981308350689 \times 10^{-14}$

计算到n=100000000时，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589817
The elapsed time is 1.614630e-01 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.548350e-01 seconds
pi = 3.141592653589793
```

并行程序误差 $E_{parallel} = 9.999975958407958 \times 10^{-9}$

串行程序误差 $E_{serial} = 1.000046712107405 \times 10^{-8}$

并行程序和串行程序结果之差：$4.911626660941693 \times 10^{-13}$

由以上结果可知，并行程序的计算结果和串行程序的计算结果大致相同（误差的数量级在10^{-14}左右），大概在计算到小数点后14位小数时，并行程序计算的结果才出现了和串行程序不相符的情况。

随着n的增大，串行程序和并行程序的计算结果越来准确，由结果观察误差减小的倍数和n增大的倍数成反比。

将线程数从4增加到8时，计算到n=100000000，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589880
The elapsed time is 8.085394e-02 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.439931e-01 seconds
pi = 3.141592653589793
```

将线程数从8增加到16时，计算到n=100000000，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589896
The elapsed time is 8.606195e-02 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.410792e-01 seconds
pi = 3.141592653589793
```

将线程数从16增加到32时，计算到n=100000000，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589664
The elapsed time is 6.121707e-02 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.445050e-01 seconds
pi = 3.141592653589793
```

将线程数从32增加到64时，计算到n=100000000，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589875
The elapsed time is 5.037403e-02 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.492541e-01 seconds
pi = 3.141592653589793
```

将线程数从64增加到128时，计算到n=100000000，输出如下：

```
======== OUTPUT ========
With n = 100000000 terms,
Our estimate of pi = 3.141592643589707
The elapsed time is 5.092812e-02 seconds
Single thread est = 3.141592643589326
The elapsed time is 6.400039e-01 seconds
pi = 3.141592653589793
```

由以上数据可知，随着线程数的增加，串行程序的运算时间基本不变，并行程序的计算时间明显减少，但当线程数增加到一定程度时，提速的速度减慢，此时再增加线程数，耗费的时间减少得不明显。

总结

通过这次实验，我学会了如何使用信号量semaphore来实现路障，从而实现各个线程对临界区的访问，使程序输出正确的结果。

在本次实验中尝试过修改课本中给的算法，将数据每间隔my_n项分配给各个进程，但不知为什么这样分所耗费的时间会更多，尽管结果是正确的，但最后还是采用了参考的算法。

总而言之，此次实验使我受益匪浅。

参考

https://blog.csdn.net/u013457167/article/details/78318932