**并行计算 作业**

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 王少博 | 学号 | 181110315 |
| 总分 | 10 | 实际得分 |  |
| 作业内容（问题，思路，程序，结果，过程中遇到问题的解决方法） | | | |
| 问题一  **编写Pthreads程序实现梯形法则求面积**   1. **使用共享变量对线程的计算结果进行累加** 2. **使用busy-waiting，mutexes和semaphores实现对临界区域的互斥** 3. **根据结果分析每种方法的优缺点** 4. **思路**   面积公式为：   1. **梯形法则求面积的并行程序的伪代码：**   h = (b-a)/n;  approx = (f(a))+f(b)))/2.0;  for (int i=1;i<=n-1;i++){  x\_i = a+i\*h;  approx += f(x\_i);  }  approx = h\*approx;   1. **主要代码（详细代码请见附件）** 2. **busy-waiting**   **我们使用了以下的全局变量.在主线程中对flag初始化为0，使用flag来忙等待，控制进入临界区。**  **/\*global variable \*/**  **long thread\_count;//in**  **long long n;//in**  **double a;//left\_endpt in**  **double b;//right\_endpt in**  **long double h;**  **long long trap\_count;**  **long double sum;**  **int flag;**  **主要函数。**  **/\*To calculate the trapezoidal area by multi-thread\*/**  **void\* Thread\_sum(void\* rank);**  **/\* Only executed by main thread \*/**  **void Get\_args(int argc, char\* argv[]);**  **void Usage(char\* prog\_name);**  **double Serial\_trap(long long n);**  **/\* Function we're integrating \*/**  **double f(double x);**  **int main(int argc, char\* argv[])**  **{**  **long thread; /\* Use long in case of a 64-bit system \*/**  **pthread\_t\* thread\_handles;**  **double start, finish, elapsed;**  **/\* Get number of threads from command line \*/**  **Get\_args(argc, argv);**  **thread\_handles = (pthread\_t\*) malloc (thread\_count\*sizeof(pthread\_t));**  **sum = 0.0;**  **flag = 0;**  **h = (long double)(b-a)/n;**  **trap\_count = n/thread\_count;**  **GET\_TIME(start);**  **for (thread = 0; thread < thread\_count; thread++)**  **pthread\_create(&thread\_handles[thread], NULL, Thread\_sum, (void\*)thread);**  **for (thread = 0; thread < thread\_count; thread++)**  **pthread\_join(thread\_handles[thread], NULL);**  **GET\_TIME(finish);**  **elapsed = finish - start;**  **sum = h\*sum;**  **printf("With n = %lld terms,\n", n);**  **printf(" Our estimate of area = %.15Lf\n", sum);**  **printf("The elapsed time is %e seconds\n", elapsed);**  **GET\_TIME(start);**  **sum = Serial\_trap(n);**  **GET\_TIME(finish);**  **elapsed = finish - start;**  **printf(" Single thread est = %.15Lf\n", sum);**  **printf("The elapsed time is %e seconds\n", elapsed);**  **printf(" accurate area = 2.666666666666667\n");**  **free(thread\_handles);**  **return 0;**  **} /\* main \*/**  **线程的实现：我们可以减少临界区代码的执行次数，得到以下的线程函数：**  **void\* Thread\_sum(void\* rank)**  **{**  **long my\_rank = (long) rank;**  **long double x;**  **long double my\_sum = 0.0;**  **long double start = a+trap\_count\*my\_rank\*h;**  **long double end = a+trap\_count\*(my\_rank+1)\*h;**  **my\_sum = (f(start) + f(end))/2.0;**  **for (long long i = 1; i <= trap\_count; i++)**  **{**  **x = start+i\*h;**  **my\_sum += f(x);**  **}**  **sem\_post(&semaphore\_p);**  **sum += my\_sum;**  **sem\_wait(&semaphore\_p);**  **return NULL;**  **} /\* Thread\_sum \*/**   1. **mutexes**   **在主函数中，我们对mutexes进行初始化，并且在程序结束前销毁mutexes。**  **在之前的主函数中做一些修改：**  **增加**pthread\_mutex\_init函数**和pthread\_mutex\_destroy函数：**      **使用mutexes的线程函数实现如下所示：**  **void\* Thread\_sum(void\* rank)**  **{**  **long my\_rank = (long) rank;**  **long double x;**  **long double my\_sum = 0.0;**  **long double start = a+trap\_count\*my\_rank\*h;**  **long double end = a+trap\_count\*(my\_rank+1)\*h;**  **my\_sum = (f(start) + f(end))/2.0;**  **for (long long i = 1; i <= trap\_count; i++)**  **{**  **x = start+i\*h;**  **my\_sum += f(x);**  **}**  **pthread\_mutex\_lock(&mutex);**  **sum += my\_sum;**  **pthread\_mutex\_unlock(&mutex);**  **return NULL;**   1. **semaphores**   **在主函数中，我们对semaphores进行初始化，并且在程序结束前销毁semaphores。**  **在之前的主函数中增加了sem\_init和sem\_destroy函数：**      **使用semaphores的线程函数的实现如下所示：**  void\* Thread\_sum(void\* rank)  {  long my\_rank = (long) rank;  long double x;  long double my\_sum = 0.0;  long double start = a+trap\_count\*my\_rank\*h;  long double end = a+trap\_count\*(my\_rank+1)\*h;  my\_sum = (f(start) + f(end))/2.0;  for (long long i = 1; i <= trap\_count; i++)  {  x = start+i\*h;  my\_sum += f(x);  }  sem\_post(&semaphore\_p);  sum += my\_sum;  sem\_wait(&semaphore\_p);  return NULL;  } /\* Thread\_sum \*/   1. **结果**   测试结果太多，所以选取部分结果截图展示，详细见附件的result.txt文件。当选取**4**个线程，划分**40000个**小梯形，计算在上从0到2的面积。分别使用**busy-waiting，mutexes和semaphores三种方式。结果依次如下所示。**      当n比较小的时候，大概小于1e5数量级的时候，并行计算的所耗时间和串行计算所耗时间相差不多，准确度也不是很高。当n大于10的5次方数量级时，准确度就越高，并行计算的所耗时间就越短。而且在相同的线程数量和n的条件下，busy-waiting所花费的时间小于mutexes所花费的时间，小于semaphores所花费的时间。所以说使用semaphores来保护临界区是一个比较理想的做法。 | | | |
| 教师评价 | | | |
|  | | | |