实验名称 求素数个数

一. 实验内容:

问题描述:
给定正整数n，求小于等于n的所有素数个数。请使用确定性算法。

输入:
正整数n (2<n<10,000,000)

输出:
范围内的素数个数

要求：测试n=100000，500000，1000000时的实验性能

二. 实验环境:

操作系统:	ubuntu 16.04 LTS 64bit
编译器:	gcc 5.3.1 (for openmp)
	nvcc 7.5.17 (for cuda)
CPU:	Intel i7-4720HQ CPU @ 2.60GHz * 8
GPU:	NVIDIA GTX965M 2G GDDR5
memory:	8GB DDR3L

三. 实现步骤:

用的是naive的方法。对每一个i <= n，判断对每一个j <= i 是否满足 i mod j == 0， 如果满足，则说明i不是素数；否则，i是素数。因为某些i显然不会是素数，所以可以不判断这些i，进而进行优化。

1. OpenMP

```
#include <stdlib.h>
#include <stdio.h>
#include <omp.h>

#define NUM_THREAD 8

int prime_number (int n)
{
   int i;
   int j;
   int prime;
   int total = 1;

   # pragma omp parallel \
   shared(n) \
   private(i, j, prime)

   # pragma omp for reduction ( + : total )
```

```
    for (i = 3; i <= n; i += 2)
    {
        prime = 1;
        if (i % 2 == 0)
            prime = 0;
        for (j = 3; j < i / 2; j += 2)
        {
            if (i % j == 0)
            {
                prime = 0;
                break;
            }
        }
        total = total + prime;
    }

    return total;
}

int main (int argc, char *argv[])
{
    int n;
    double wtime;
    long primes;

    printf ("Number of processors available = %d\n", omp_get_num_procs ( ) );
    printf ("Number of threads =              %d\n", omp_get_max_threads ( ) );
    printf("\n\tn\t\tprime_num\ttime(second)\n");
    n = 100000;

    omp_set_num_threads(NUM_THREAD);

    wtime = omp_get_wtime();
    primes = prime_number(n);
    wtime = omp_get_wtime() - wtime;
    printf("\t%d\t\t%d\t\t%lf\n", n, primes, wtime);

    n = 500000;

    wtime = omp_get_wtime();
    primes = prime_number(n);
    wtime = omp_get_wtime() - wtime;
    printf("\t%d\t\t%d\t\t%lf\n", n, primes, wtime);

    n = 1000000;

    wtime = omp_get_wtime();
    primes = prime_number(n);
    wtime = omp_get_wtime() - wtime;
    printf("\t%d\t\t%d\t\t%lf\n", n, primes, wtime);

    return 0;
}
```

2. CUDA

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#include <cuda.h>

__global__ void generatePrimes(int *sizeD, int *maxPrimeD, int *numPrimesD) {
 __shared__ int numPrimesB;
 __shared__ int maxPrimeB;
 if(threadIdx.x == 0) {
       numPrimesB = 0;
       maxPrimeB = 0;
 }
 __syncthreads();
 int no = 9 + (blockIdx.x * 2000) + (2 * threadIdx.x);
 if(no < *sizeD) {
       int noRt = __double2int_ru(sqrt(__int2double_rn(no)));
       int k=3;
       for (k = 3; k <= noRt; k += 2) {
             if (no % k == 0) {
                   break;
             }
       }
       if (k > noRt) {
             atomicAdd(&numPrimesB, 1);
             atomicMax(&maxPrimeB, no);
       }
       __syncthreads();
       if(threadIdx.x == 0) {
             atomicAdd(numPrimesD, numPrimesB);
             atomicMax(maxPrimeD, maxPrimeB);
       }
 }
}

int main(int argc, char* argv[]) {

 struct timeval t;
 double start_t, end_t, time_spent;
 int maxPrime = 0, numPrimes = 0, size = atoi(argv[1]);
 int gridSize = (int)(floor(size/2000.0));
 dim3 dimGrid(gridSize+1);
 dim3 dimBlock(1000);

 int *sizeD, *maxPrimeD, *numPrimesD;
 cudaMalloc((void**)&sizeD, sizeof(int));
 cudaMalloc((void**)&maxPrimeD, sizeof(int));
 cudaMalloc((void**)&numPrimesD, sizeof(int));

 gettimeofday(&t, NULL);
 start_t = (t.tv_sec * 1000000.0) + t.tv_usec;
 cudaMemcpy(sizeD, &size, sizeof(int),cudaMemcpyHostToDevice);
 cudaMemcpy(maxPrimeD, &maxPrime, sizeof(int),cudaMemcpyHostToDevice);
 cudaMemcpy(numPrimesD, &numPrimes, sizeof(int),cudaMemcpyHostToDevice);
 generatePrimes<<<dimGrid, dimBlock>>> (sizeD, maxPrimeD, numPrimesD);
 cudaMemcpy(&maxPrime, maxPrimeD, sizeof(int),cudaMemcpyDeviceToHost);
 cudaMemcpy(&numPrimes, numPrimesD, sizeof(int),cudaMemcpyDeviceToHost);
 gettimeofday(&t, NULL);
 end_t = (t.tv_sec * 1000000.0) + t.tv_usec;
```

```
    time_spent = end_t - start_t;

    cudaFree(sizeD);
    cudaFree(maxPrimeD);
    cudaFree(numPrimesD);


    printf("Max prime number %d\ninput size %d\nTotal time %f secs\nprime number
%d\n",
                maxPrime, size, time_spent / 1000000.0, numPrimes+4);

    return 0;
}
```

四. 实验结果报表

1. OpenMP

本次实验代码总行数	71 lines
累计耗费时间	？？？

运行时间(sec)

规模 \ 核数	1	2	4	8
100000	0.378	0.262	0.160	0.113
500000	7.769	5.687	3.354	2.352
1000000	29.876	21.591	12.733	8.727

加速比

规模 \ 核数	1	2	4	8
100000	1	1.4427	2.3625	3.3451
500000	1	1.3361	2.3163	3.3031
1000000	1	1.3827	2.3463	3.4234

3.CUDA

本次实验代码总行数	71 lines
累计耗费时间	？？？

运行时间(sec)

规模	线程数 blocks_num = size / 2000 threads_num = 1000
100000	0.000366
500000	0.002958
1000000	0.008126

加速比

规模	线程数 blocks_num = size / 2000 threads_num = 1000
100000	1032
500000	2626
1000000	2676

五. 对本次实验的收获和总结

这是我第一次写并行程序。通过并行化，的确可以大大地提高运算效率。
这次我选用了openmp和cuda这两个编程模型。
其中，openmp是我在编写并行程序时首选的编程模型，因为openmp实现起来特别简单，只要在合适的地方插入编译制导语句、在少部分地方做一些修改，就可以把原来的串行程序并行化。
而我选择用CUDA的原因则是因为CUDA的计算能力真的很厉害。只要原来的问题适合并行化，就可以通过并行化，把原来的代码的计算时间缩短上百倍、上千倍。而且CUDA在训练神经网络时也有很好的应用。

六. 其他问题（请在符合自己情况的地方填入*）

1.你使用的编程模型是 (多选)

OpenMP	MPI	CUDA	MapReduce
*		*	

注：以下只需填实验中选择的实现方式对应的行，每行单选。

2.在进行并行编程时，我愿意经常使用此编程模型 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合

OpenMP	*				
MPI					
CUDA		*			
MapReduce					

3.我认为此编程模型有许多非必要而复杂的部分(单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP				*	
MPI					
CUDA				*	
MapReduce					

4.我认为在此编程模型下编程是容易的(单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP	*				
MPI					
CUDA				*	
MapReduce					

5.我认为我需要有技术人员的支持才能使用此编程模型 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP				*	
MPI					
CUDA				*	
MapReduce					

6.我认为在此编程模型中，有许多功能都很好地整合在一起 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP	*				
MPI					

CUDA		*			
MapReduce					

7.我认为在此编程模型中有许多不一致的地方 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP				*	
MPI					
CUDA				*	
MapReduce					

8.我认为大多数人都能很快地掌握此编程模型 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP		*			
MPI					
CUDA				*	
MapReduce					

9.我认为此编程模型不灵活 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP		*			
MPI					
CUDA				*	
MapReduce					

10.对于在此编程模型下编程，我对问题能够被解决感到很自信 (单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP			*		
MPI					
CUDA			*		
MapReduce					

11.在使用此编程模型前，我需要了解大量的知识(单选)

	非常符合	比较符合	不确定	比较不符合	非常不符合
OpenMP				*	
MPI					
CUDA		*			
MapReduce					