Programming Assignment 1

Author: Gabriel Hofer

CSC-410 Parallel Computing

Instructor: Dr. Karlsson

Due: September 18, 2020

Computer Science and Engineering
South Dakota School of Mines and Technology

Sieve of Eratosthenes

Listing 1: Non-parallelized Sieve of Eratosthenes (prime.c)

```c
void erat(int n, int * pcnt){
  for(int i=2;i<=n;i++)
    sieve[i]=1;
  for(int i=2;i*i<=n;i++)
    if(sieve[i])
      for(int j=i*i;j<=n;j+=i)
        sieve[j]=0;
  *pcnt=0;
  for(int i=2;i<=n;i++)
    if(sieve[i])
      primes[(*pcnt)++]=i;
}
```

Listing 2: Parllelized Sieve of Eratosthenes (prime.c)

```c
void erat2(int n, int * pcnt){
  #pragma omp parallel for
  for(int i=2;i<=n;i++)
    sieve[i]=1;
  int sqrtn = sqrt((double)n);
  for(int i=2;i <= sqrtn;i++)
    if(sieve[i]){
      #pragma omp parallel for
      for(int j=i*i;j<=n;j+=i)
        sieve[j]=0;
    }
  *pcnt=0;
  for(int i=2;i<=n;i++)
    if(sieve[i])
      primes[(*pcnt)++]=i;
}
```

Listing 3: Measuring Runtime Performance (prime.c)

```
scanf("%i",&n);
double start, end;

pcnt=0;
start = omp_get_wtime();
erat(n,& pcnt);
end = omp_get_wtime();
print(pcnt);
printf("Elapsed␣time␣=␣%f␣seconds\n\n", end-start);

// reset primes and sieve.
for(int i=0; i<(1<<30); i++){
  sieve[i]=0;
  primes[i]=0;
}

pcnt=0;
start = omp_get_wtime();
erat2(n,& pcnt);
end = omp_get_wtime();
print(pcnt);
printf("Elapsed␣time␣=␣%f␣seconds\n\n", end-start);
```

Listing 4: Output in Terminal from prime sieve program (prime.c)

Monte Carlo Method

Listing 5: Non-parallelized Monte Carlo Method (monte.c)

```c
double monte(long long n){
  long long hits=0;
  double x, y, pi;
  for(int i=0; i<n; i++)
    hits += sq((double)rand()/((double)RAND_MAX)) +
      sq((double)rand()/((double)RAND_MAX))
      <= 1.0 ? 1 : 0;
  pi = 4.0*hits/(double)n;
  return pi;
}
```

Listing 6: Parallelized Monte Carlo Method (monte.c)

```c
double monte2(long long n){
  long long hits=0;
  double x, y, pi;
  #pragma omp parallel for
  for(int i=0; i<n; i++)
    hits += sq((double)rand()/((double)RAND_MAX)) +
      sq((double)rand()/((double)RAND_MAX))
      <= 1.0 ? 1 : 0;
  pi = 4.0*hits/(double)n;
  return pi;
}
```

Listing 7: Measuring Runtime Performance (monte.c)

```
long long n;
double start, end, _PI_;
scanf("%lld", & n);

printf("Monte␣Carlo␣Method␣NON-parallelized\n");
start = omp_get_wtime();
_PI_=monte(n);
end = omp_get_wtime();

printf("PI:␣%f\n",_PI_);
printf("Elapsed␣time␣=␣%f␣seconds\n\n", end-start);

printf("Monte␣Carlo␣Method␣parallelized\n");
start = omp_get_wtime();
_PI_=monte2(n);
end = omp_get_wtime();

printf("PI:␣%f\n",_PI_);
printf("Elapsed␣time␣=␣%f␣seconds\n\n", end-start);
```

Listing 8: Output in Terminal (monte.c)