WITS UNIVERSITY

School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

COMS4040A & COMS7045A: High Performance Computing & Scientific Data Management
Exercise

2019-2-21

Objectives

- Design and implement parallel algorithms using OpenMP.

Problems

1. Based on the given serial program for computing the number π, complete the various approaches discussed in the class, namely, Method I, Method II, and using loop construct with a reduction.
2. Suppose OpenMP does not have the reduction clause. Implement an efficient parallel reduction for summation. Compare the performances of your implementation with using the OpenMP reduction.
3. Use OpenMP directives to implement a parallel program for the *Sieve of Eratosthens*. Benchmark your program for various values of `n` (find the prime numbers within `n`) and `t` (number of threads).
4. For each of the following code segments, use OpenMP pragmas to make the loop parallel, or explain why the code segment is not suitable for parallel execution.

a)
```
.....
  for(int i=0; i< n-1; i++)
  {
    a[i]=b[i]+c[i];
    b[i]=b[i+1]-c[i];
    c[i]=a[i]-b[i];
  }
.....
```

b)
```
.....
  for(int i = 0; i < (int)sqrt(x); i++)
  {
    a[i] = 2.3 * i;
    if(i < 10)
      b[i] = a[i];
  }
.....
```

c)
```
......
  flag = 0;
  for(int i = 0; (i < n) & (!flag); i++)
  {
    a[i] = 2.3 * i;
    if(a[i] < b[i])
      flag = 1;
  }
......
```

d)
```
......
  for(int i = 0; i < n; i++)
  {
    a[i] = foo(i);
  }
......
```

e)
```
......
  for(int i = 0; i < n; i++)
  {
    a[i] = foo(i);
    if(a[i] < b[i])
      break;
  }
......
```

f)
```
......
  dotp = 0;
  for(int i = 0; i < n; i++)
  {
    dotp += a[i] * b[i];
  }
......
```

g)
```
......
  for(int i = k; i < 2*k; i++)
  {
    a[i] = a[i] + a[i - k];
  }
......
```

h)
```
......
  for(int i = k; i < n; i++)
  {
```

```
4       a[i] = b * a[i - k];
5     }
6  ......
```

5. Find the loop carried dependency in the following code. Can you remove it? How?

```
1  ......
2    double factor=1.0;
3    double sum=0.0;
4    for(k=0;k<n;k++){
5      sum+=factor/(2*k+1);
6      factor=-factor;
7    }
8    val=4.0*sum;
9  ......
```

6. Matrix operations play a key role in many scientific applications. We consider the problem of multiplying a matrix A of size $n \times n$ by a vector b of $n \times 1$, where the result is a vector y of $n \times 1$.
   (a) Implement a serial version of $y = Ab$.
   (b) Once you have a serial version, parallelize it using OpenMP.
7. Parallelize the program `fib.c` for computing the nth Fibonacci number using OpenMP `sections/section` construct and `task` construct, respectively. Compare the performances of these two approaches.
8. Parallelize the sequential *quicksort* program (`qsort.c`) using OpenMP `task` construct and `sections/section` construct. Compare the performances of three versions of quicksort implementations, i.e., the sequential implementation, the OpenMP implementation using `sections/section` construct, and the OpenMP implementation using `task` construct.