Practical Session 3 : Synchronization and Tasks

1 Synchronization

1.1 Number of Threads

Study and correct the following code using two different approaches. You are only allowed to add OpenMP directive without the `reduction` clause

```
void nb_thread_kernel() {
  size_t nb_threads = 0;
  #pragma omp parallel
  {
    nb_threads++;
  }
  printf("nb_threads = %zu\n", nb_threads);
}
```

1.2 First Prime Numbers

Study and parallelize the following code [Link] :

```
void prime_kernel(size_t primes[], size_t* ptr_nb_primes) {
  size_t nb_primes = 0;
  size_t divisor;
  bool is_prime;

  for (size_t i = PRIME_MIN; i < PRIME_MAX; i+=2) {
    is_prime = true;
    divisor = PRIME_MIN;

    while ((divisor < i) && is_prime) {
      if ((i % divisor) == 0)
        is_prime = false;
      divisor += 2;
    }

    if (is_prime) {
      primes[nb_primes] = i;
      nb_primes++;
    }
  }

  *ptr_nb_primes = nb_primes;
}
```

1.3 Task Graph

Study and parallelize the following code assuming that your OpenMP runtime does not support nested parallelism (i.e., you cannot create a parallel region in a parallel region). You will perform a first parallelization without OpenMP tasks, then you will exploit the tasks that should allow you to further improve the execution time[Link] :

```
void dag_kernel(double r1, double r2, double r3, double* r) {
  double d1, d2, d3, d4, d5, d6;

  d1 = f(r1, r2, 1);
  d2 = f(r2, r3, 1);
  d3 = f(d1, d2, 1);
  d4 = f(r1, r3, 2);
  d5 = f(r2, d2, 1);
  d6 = f(d5, d4+d3, 1);

  *r = d6;
}
```

2 Non-Iterative Loops: Fibonacci Numbers

Study and parallelize the following code [Link] :

```
int fibok(int n) {
  if (n < 2)
    return n;

  return fibok(n-1) + fibok(n-2);
}

void fibonacci_kernel(int n, int* fibo) {
  *fibo = fibok(n);
}
```

3 Parallelization of a Real-Life Code

In real-life, codes which can be parallelizes without modification are rare species!

- Get the following codes (Credit: Dominique Béréziat, sorry it is in French hehe :-)!): [Link] and [Link].
  - This code computes Mandelbrot set.
  - Stores the result as an image using the ras format (you may open it with, e.g., LibreOffice).
- Read the code and study its parallelism.
- Modify it to enable its parallelization using OpenMP.
  - Find an equivalent, acceptable loop form [doc].
  - Modify some computation to expose parallelism.
- Implement a parallel version using OpenMP.