Exercices autour d'OpenMP

Exercice 1- La suite de Fibonacci

Les nombres de Fibonacci sont définis comme suit

$$F(0) = 0; F(1) = 0$$
$$F(n) = F(n-1) + F(n-2) \qquad \forall n \geq 2$$

On considère le code suivant

```
long comp_fib_numbers(int n){
  //
  // Basic algorithm: f(n) = f(n-1) + f(n-2)
  //
  long  fnm1, fnm2, fn;
  if ( n == 0 || n == 1 ) return(1);

  fnm1 = comp_fib_numbers(n-1);

  fnm2 = comp_fib_numbers(n-2);

  fn   = fnm1 + fnm2;
  return fn ;
}
```

Listing 1: code `fibonacci.c`

Paralléliser à l'aide du paradigme de tâche le code ci-dessus.

Exercice 2- Triangle de Pascal

On souhaite paralléliser le calcul du triangle de Pascal pour une hauteur n à l'aide de tâches. Le triangle de Pascal pour un hauteur de 9 est

```
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

Le code `binomial.c` utilise la récurrence suivante

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k} \tag{1}$$

avec $\binom{n}{k} = \frac{n!}{(n-k)!k!}$ pour calculer le triangle

Exercice 3- Calcul de pi

On considère le code suivant pour calculer pi

```
int main() {

 double x, pi, sum = 0.0;
 double PI25DT = 3.14159265358979323846264643;
 long num_steps = 100000;
 double step = 1.0/(double) num_steps;
 int start = 1, end   = num_steps;

 for (int i=start;i<= end; i++){
   x = (i-0.5)*step;
   sum = sum + 4.0/(1.0+x*x);
 }

 pi = step * sum;
 printf("pi := %.16e  %.e\n", pi, fabs(pi - PI25DT));
 return 0 ;
 }
```

On parallélise la boucle par quatre différentes approches :

1. On parallélise à la main la réduction. Pour cela, on introduit un tableau `SUM` de dimension le nombre de threads. Chaque thread calcule sa somme locale et la stocke dans le tableau. Puis on évalue la somme globale.

   ```
   double sum[nb_threads];
   ```

2. Utiliser la directive `atomic` pour mettre à jour la valeur de pi avec la contribution locale.

   ```
   #pragma omp atomic update
   ```

3. Utiliser la clause `reduction` pour paralléliser le calcul

4. Utiliser la directive `taskloop` avec la clause `reduction` pour paralléliser le calcul

Comparer les temps de calculs en fonction du nombre de threads.
Expliquer ce qui se passe dans la méthode 1. Que faut-il faire pour accélérer cette méthode ?
Reprendre les exemples en utilisant la clause `simd`.