Exercise Sessions 2&3 : Program Parallelization

In all the exercises of this session, you have to study the dependencies between instructions and between the different instances (iterations) of the same instruction. Find the operations that can run concurrently and parallelize the codes using OpenMP directives. Code alterations may be necessary in addition to OpenMP directive insertions. If necessary, explain your code alterations, your work sharing and data scoping choices to get correct results. Then discuss the performance issues.

1 OpenMP Directives

In this part, only OpenMP directive insertion is allowed, do not modify the C code apart from this.

1.1 Vector Addition

Parallelize the following code [Link] :

```
void addvec_kernel(double c[N], double a[N], double b[N]) {

  for (size_t i = 0; i < N; i++) {
    c[i] = a[i] + b[i];
  }
}
```

1.2 Sum of Vector Elements

Parallelize the following code [Link] :

```
void sum_kernel(double* psum, double a[N]) {
  double sum = 0.;

  for (size_t i = 0; i < N; i++) {
    sum += a[i];
  }
  *psum = sum;
}
```

1.3 Matrix-Vector Product

Parallelize the following code [Link] :

```
void matvec_kernel(double c[N], double A[N][N], double b[N]) {
  size_t i, j;

  for (i = 0; i < N; i++) {
    c[i] = 0.;
    for (j = 0; j < N; j++) {
      c[i] += A[i][j] * b[j];
    }
  }
}
```

2 Program Restructuring

In this part, parallelize the following codes by adding OpenMP directives and modifying the codes if necessary. Changes to the original code should be as limited as possible.

2.1 1D Stencil

Parallelize the following code [Link] :

```
void stencil1D_kernel(double a[N], double b[N]) {
  for (size_t i = 1; i < N; i++) {
    a[i] = (a[i] + a[i - 1]) / 2;
    b[i] = (b[i] + b[i - 1]) / 2;
  }
}
```

2.2 Reduction and Reinitialization

Parallelize the following code [Link], (1) in the case where it is not acceptable to have precision errors and (2) in the case where they can be tolerated.

```
void reduction_reinit_kernel(double A[N][N], double* sum) {
  *sum = 0.;

  for (size_t i = 0; i < N; i++) {
    for (size_t j = 0; j < N; j++) {
      *sum += A[i][j];
      A[i][j] = 0.;
    }
  }
}
```

2.3 Multiplication of Polynomials

Parallelize the following code [Link] :

```
void polynomial_multiply_kernel(double c[2*N-1], double a[N], double b[N]) {
  for (size_t i = 0; i < N; i++)
    for (size_t j = 0; j < N; j++)
      c[i+j] += a[i] * b[j];
}
```

3 Algorithm Restructuring

Parallelize the following codes using OpenMP by adding OpenMP directives, modifying codes or even algorithms if necessary. Alterations to the original code or algorithm should be as limited as possible.

3.1 Bubble Sort

Parallelize the following code [Link] :

```
void bubble_sort_kernel(double tab[N]) {
  size_t i, j;
  double temp;

  for (i = 0; i < N ; i++) {
    for (j = 0; j < N - i - 1; j++) {
      if (tab[j] > tab[j + 1]) {
        temp = tab[j + 1];
        tab[j + 1] = tab[j];
        tab[j] = temp;
      }
    }
  }
}
```