Lab 5:
OpenMP Part 2

Jan Izquierdo
jan.izquierdo@

Laura LLorente
laura.llorente@

17/05/2024

Load the necessary modules:

```
module load openmpi/4.1.1
module load gcc/12.1.1
```

1 node running a single MPI process with 12 threads.

```
#SBATCH --nodes=1
#SBATCH --ntasks=1
#SBATCH --ntasks-per-node=1
----module load gcc/12.1.1
export OMP_NUM_THREADS= 12
mpirun hellohybrid
```

2 nodes running a single process in each node and each process is divided into 12 threads

```
#SBATCH --nodes=2
#SBATCH --ntasks=2
#SBATCH --ntasks-per-node=1
#SBATCH --cpus-per-task=12
----
export OMP_NUM_THREADS=12
mpirun hellohybrid
```

2 nodes running two processes each, with 6 threads per process

```
#SBATCH --nodes=2
#SBATCH --ntasks=4
#SBATCH --ntasks-per-node=2
#SBATCH --cpus-per-task=6
----
export OMP_NUM_THREADS=6
mpirun hellohybrid
```

2 nodes running 24 single-threaded MPI processes (12 per node)

```
#SBATCH --nodes=2
#SBATCH --ntasks=24
#SBATCH --ntasks-per-node=12
----
export OMP_NUM_THREADS=12
mpirun hellohybrid
```

1. **Modify the program to count ALL 0-sum subarrays that exist in a given list of integers**

   To count the number of sums equal to 0 we modified the existing function to add 1 to a counter instead of finishing with a return every time a sum equal to 0 was found:

```
int sumCounter(int arr[], int n){
   int c=0;

   for (int i=0;i<n;i++){

       int sum = arr[i]; //if element is 0 -> +1 to counter
       if (sum == 0){c++;}

       for (int j = i + 1; j < n; j++) { //starting from current
position and forward
           sum += arr[j]; //every time you advance a position add
the corresponding number
           if (sum == 0) // if the sum of the numbers from i so
far is 0, add 1 to the counter
               c++;
       }
   }
   return c;
}
```

2. **Parallelize the function that counts the number of 0-sum subarrays using the OpenMP library. The program must have two versions of the same function: the sequential and the parallel, to verify that the program provides the same correct results in both cases**

   In order to parallelize the sumCounter, we added the directive **#pragma omp parallel for reduction(+:c)** in order to execute each for loop with a different thread, the reduction clause gives each thread a private copy of the c counter and at the end of the parallel region, all the local copies are combined into the final result :

```
int PsumCounter(int arr[], int n){ //same reasonings as
sumCounter
```

```
   int i, j, c=0; //i, j defined here for parallels later

   #pragma omp parallel for reduction(+:c) //distributes an i for
each process, makes c private and adds them up after the for loop
is finished to ensure its correct
   for (i=0;i<n;i++){
       int sum = arr[i];
       if (sum == 0){c++;}

       #pragma omp parallel for reduction(+:c) // distributes a j
for each process, makes c private and adds them up when the loop
is finished to ensure its correct
       for (j = i + 1; j < n; j++){
           sum+=arr[j];
           if (sum==0){c++;}
       }
   }
   return c;
}
```

In order to test both functions, we used different arrays to check if the results were right. In the final test we set different seeds with a large array (2000). The result proves that both sequential and parallel functions always prints the same correct result:

```
[biohpc-25@clus-login LAB5]$ ./sum0


***********  Found a subarray with 0 sum ************


****  Found 8650 subarrays with 0 sum using a sequential program  ****


*****  Found 8650 subarrays with 0 sum using a parallel program  *****
```