Assignment 4: OpenMP - loops

The purpose of this assignment is for you to learn more about

- the parallel loop construct of OpenMP,
- how loop scheduling works in OpenMP,
- how easy (or not so easy) is writting parallel codes with OpenMP loop construct.

As usual all time measurements are to be performed on the cluster.

The grade thresholds for this assignments are $A \geq 70$; $B \geq 60$; $C \geq 50$; $D \geq 40$.

Activate OpenMP in GCC by passing `-fopenmp` to the compiler and linker. Note that if you omit this parameter, the code will probably still compile. But its execution will be sequential. Note that the `Makefile` provide the parameters you need, so you do not need to worry about it in this assignment.

You can control the number of threads in OpenMP in two ways: the environment variable `OMP_NUM_THREADS` or by calling the `omp_set_num_threads` function. The number of thread to use will be passed to the program as a parameter to `main`, so use the `omp_set_num_threads` function call to set the number of threads.

Loop scheduling in OpenMP can be controlled either by specifying `schedule(runtime)` in the parallel for construct and specifying a `OMP_SCHEDULE` environment variable, or by using the `omp_set_schedule` function. The scheduling policy is passed to the program as a parameter to `main`, so use the `omp_set_schedule` function call.

The first two problems are here to kickstart your OpenMP programming and why people love OpenMP (and only use pthreads when necessary). The latter one are more complex.

Question: Before starting, run all sequential codes in mamba using `make bench`.

1 Reduce (20 pts)

Question: Write a program that compute the sum of an array in parallel using the OpenMP for loop construct. Measure the time computing the sum takes. Write the program so that you can control the length of the array, the number of thread and the scheduling policy; passed as parameters to `main`. Output the value of the reduction to `stdout` and the time taken to compute it to `stderr`. Use the template of `reduce/reduce.cpp`; note that the array to reduce is obtain by calling the provided `generateReduceData` function. Test it works with `make test`.

Question: Run the code on mamba, in the `reduce/` directory, using `make bench`. Plot speedup charts with `make plot`. Does the plots make sense? Why?

2 Numerical Integration (20 pts)

Question: Take your sequential code to compute numerical integration from the previous assignment. Modify it to do the numerical integration in parallel with the OpenMP loop construct. Output the value of the integration to `stdout` and the time taken to compute it to `stderr`. Use the template provided in `numint/numint.cpp`. You can test your code with `make test`.

Question: Run the code on mamba, in the `numint/` directory, using `make bench`. Plot charts with `make plot`. Does the plot make sense? Why?

3 Prefix Sum (30 pts)

Here is a sequential Prefix Sum:

```
void prefixsum (int* arr, int n, int* pr) {
  pr[0] = 0;

  for (int i=0; i<n; ++i) {
    pr[i+1] = prefix[i] + arr[i];
  }
}
```

Question: Describe the structure of the parallel algorithm for Prefix Sum. (Highly recommend the "cut in P and fix it" approach".)
Question: Implement a parallel function using OpenMP parallel loop constructs to compute the prefix sum of an array. Pick the scheduling policy you feel is appropriate. Output the time it took on `stderr`. Use the template provided in `prefixsum/prefixsum.cpp`. Note that the data is generated by function `generatePrefixSumData` and the correctness of the result is checked by function `checkPrefixSumResult`.
Question: Run the code on mamba, in the `prefixsum/` directory, using `make bench`. And then plot the results using `make plot`. Does the plot make sense? Why?

4 Merge Sort (30 pts)

Question: Describe the structure of the parallel algorithm for Merge Sort that does not try to make Merge parallel. How would you map that parallelism to OpenMP's looping construct?
Question: Implement a parallel function using OpenMP parallel loop constructs to perform merge sort on an array of integer. Use the template of `mergesort/mergesort.cpp`. Output the time it took on `stderr`. Note that the data is generated by function `generateMergeSortData` and the results is checked by `checkMergeSortResult`.
Question: Run the code on mamba, in the `mergesort/` directory, using `make bench`. And then plot the results using `make plot`. Does the plot make sense? Why?
Question: (Extra Credit) Still using only OpenMP loops, make Merge Sort more parallel by making Merge parallel.