OpenMP Homework 2.

==

To get credits for OpenMP topic you should
1) complete the provided code:
BugReduction.c, BugParFor.c, MatMul.c, Pi.c, Car.cpp.
2) write the programs:
Axisb.c/Axisb.cpp and **LeastSquares.c/LeastSquares.cpp**
Note in Axisb.c there are 2 problems (medium and hard (if you can, for additional credits)), NO NEED TO DO BOTH PROBLEMS INSIDE Axisb.c

==

Most complete cheat sheet on OpenMP (some of information from there you will never use): https://www.openmp.org/wp-content/uploads/OpenMP-4.0-C.pdf

Find exercises in the attached folder

There are four types of exercises: **demonstrations**, **program with bugs**, **sequential programs, tasks with an asterisk** for you to parallelize. The order of exercises below roughly corresponds to increasing of complexity. Also, subsequent exercises in the list use primitives and #pragma from previous ones.

You need to download code, compile it with:
$ <cc> <codefile.c> -o <executable> -fopenmp
Here you should specify a compiler (cc), which is, for example, gcc for .c files in Ubuntu and g++ for .cpp files
Run code with:
$ ./<executable>

I. **Demonstrations** – working examples of code for you to understand what and why something is happening.
   1. **Hello.c** – the simplest program to print 'Hello world'.
      Introduction of special omp functions and 'parallel' sections.
      Task: manually explicitly set maximal number of threads for program to use in command line (outside program) by setting:
      $ export OMP_NUM_THREADS=<number you want>.
      Or do it inside your program by uncommenting two omp commands. Try to relocate commands 'omp_…' outside #pragma. Try to set different number of threads.
   2. **OutMes.cpp** – comparison of parallel C and C++ output.

The output of the program should explain to you the notion of ‘threadsafe’ functions. Introduction of private variables and ‘critical’ clause. Compile with C++ compilers, for example, on Ubuntu:
$ g++ OutMes.cpp -o <executable> -fopenmp

3. **PrivateShared.c** – introduction of shared variables, ‘for’ clause.
   Task: by the output of the program try to understand what #pragma are doing exactly.
4. **ParSec.c** – introduction of sections.
   Task: deduce by the output what sections stand for.
5. **SumArray.c** – introduction of ‘reduction’ clause and shortcut for parallel for #pragma.

II. **Programs with bugs** – examples of erroneous code for you to fix and understand.

1. **BugReduction.c** – code for dot product function of two arrays with bugs. **(5)**
2. **BugParFor.c** – This should be a parallel for, it also should print a thread id (tid) for each thread. Google more about ‘schedule’ and ‘shared’ clauses. **(5)**

III. **Sequential programs** – examples of working code for you to parallelize.

1. **MatMul.c** – example of three different orders of indices (ijn, jin, nij) for matrix multiplication from the second lecture.(**10**)
2. **Pi.c** – the simplest integration formula implementation for calculating the value of pi. **(15)** Number pi can be calculated from integration area of a quadrant (1/4th of a circle), i.e.It is necessary to compute integral $\int_0^1 \frac{4}{1+x^2}dx = \pi$ using Monte-Carlo method.
   Detailed info: https://en.wikipedia.org/wiki/Monte_Carlo_integration
   When parallelizing, remember about race conditions (multiple threads will try to write in the same place simultaneously). Use a thread-safe random number generator in C, e.g. rand_r(). To be thread safe, each thread needs to have its own state variable. You don't however want to use the same initial value for each thread's state variable, otherwise every thread will generate the same sequence of pseudo-random values.
3. **Car.cpp** – It is given a photo of a car (car.ppm). Your task is to animate the car. You need to read file ppm, put the obtained data in to a matrix A_{ij}, and move a column j to the position j+1: A_{ij}-> A_{ij+1}. The last column must go the first column position. Save the output into the *.ppm file. See how the frequency of saving influence on the performance time. Use OpenMP. Hint: allocate additional column, thus matrix A is A=N x (M+1).(**20**)

IV. ***Problems with an Asterisk** – examples of working code for you to parallelize.

1. **Axisb.c** – It is suggested to implement linear solver for

$$A\boldsymbol{x} = \boldsymbol{b},$$

Whether Jacobi method (JM) (**20**), OR implement Gauss-Seidel(GSM) (**30**) method for solving linear equation.
Algorithm of Jacobi Method: https://en.wikipedia.org/wiki/Jacobi_method.
JM converges only if matrix A is diagonally dominant.
Gauss-Seidel Method: https://en.wikipedia.org/wiki/Gauss-Seidel_method
GSM converges if A is diagonally dominant OR symmetric and positive definite.

2. **LeastSquares.c** – It needs to solve the Linear regression problem (**25**): the set of data (x_i,y_i) is given. Implement parallel algorithm to find out unknown parameters a, b of the model $f(x, a, b) = ax + b$. To generate samples (x_i,y_i) you can set a, b and calculate: $y_i = ax_i + b + noise()$.
   This problem can be reformulated into optimization problem, i.e. minimizing the sum of squared residuals $r_i = y_i - f(x_i, a, b)$:

   $$\sum_{i=1}^{n} (y_i - f(x_i, a, b))^2 \to \min_{a,b}$$

   Gradient descending method
   Detailed info: https://en.wikipedia.org/wiki/Least_squares

Task: by parallelization of for cycles try to achieve scalability (when double the number of threads, calculation time is divided by two) for at least one order of indices. Read about 'collapse' clause.
Remember about available resources (when you have more threads than cores your computer can provide – you will not achieve speedup).
Note: clock() measures the sum of wall clocks across all threads. Use omp_get_wtime() instead (it returns value in seconds).

==

To learn more about programming with OpenMP, see a good tutorial (some of above exercises are deduced from the ones from this tutorial):
https://computing.llnl.gov/tutorials/openMP/