COL 380	January 17, 2017

Lab 1

Instructor: Subodh Sharma	Due: January 24, 23:55 hrs

1 Parallel Prefix Sum via Multithreading

You are asked to implement a prefix-sum algorithm using `Pthreads`. Parallel prefix frameworks are incredibly important and are often used as building blocks in solutions to critical problems such as sorting, solving system of linear equations, grep, compiler's lexical analysis, among others. Definition of a prefix-sum operation is as noted below:

Prefix-sum takes a list of elements $[a_1, a_2, \cdots, a_n]$, a binary operator $\oplus$ and returns an ordered set $[a_1, a_1 \oplus a_2, \cdots, (a_1 \oplus a_2 \oplus \cdots \oplus a_n)]$.

Your tasks include:

- Familiarize yourself with Pthreads API.
- Read Prefix-sum algorithm and think about *efficient* parallel implementation that includes choosing the right data-structure, amount of thread coordination, smart handling of the memory hierarchy, avoiding false-sharing, etc.
- A working parallel implementation of parallel prefix-sum.

1.1 Statement of work

- Along with the implementation, prepare a lab report (as .txt file) by clearly explaining your design decisions, parallelization strategy, load-balancing strategy, etc. Make sure that explanations are bulleted and are not long paragraphs. Keep the lab report within 200-400 words (not more than a page; exceeding the page limit will incur negative marking). To compile your programs use the following command:

  ```
  gcc -fopenmp -O2 -pthread -Wall par-prefix.c -o par
  ```

  Optimization `-O3` is not allowed. Make sure that while debugging option `-O2` is replaced by `-g`. In order to debug use DDD, GDB, or IDE-based debuggers. You can also use Valgrind, Kcachegrind to optimize the memory accesses of your program.

- In order to time the program, use API call provided by OpenMP: `omp_get_wtime()`. In order to use the call include the header file `omp.h`. The start time should be taken before the first instruction of the main program is execute and the end time should be taken just before printing the results.

- Output of the program must be in the following format:

  ```
  Threads: %d
  Time:    %f
  Md5-sum: %d
  ```

  You are required to produce an Md5 checksum of your result.

1.2 Grading

Turn in the source code file and the report. The break-down on grading is as follows:

- Correct execution of the program **60 Marks**
- Efficiency/Scalability of the implementation **30 Marks**
- Report clarity **10 Marks**

Note that all submissions will be checked for plagiarism. Write you own code. If any code is taken from the Internet, then acknowledge accordingly. If we discover a code snippet taken from the web without acknowledgement, then the assignment will be awarded zero.