Project Seminar:
Parallel and Distributed Systems

Assignment 3 (Submission deadline: Nov 30th 2015, 23:59 CET)

General Rules

The assignment solutions have to be submitted at:

https://www.dcl.hpi.uni-potsdam.de/submit/

Our automated submission system is intended to give you feedback about the validity of your file upload. A submission is considered as accepted if the following rules are fulfilled:

- You did not miss the deadline.
- Your file upload can be decompressed with a zip / tar decompression tool.
- Your submitted solution contains only the source code files and a Makefile. Please leave out any Git / Mercurial repository clones or SVN / CVS meta-information.
- Your solution can be compiled using the "make" command, without entering a separate sub-directory after decompression.
- You program runs without expecting any kind of keyboard input or GUI interaction.
- Our assignment-specific validation script accepts your program output / generated files.

If something is wrong with your submission, you will be informed via email (console output, error code). Re-uploads of corrected solutions are possible until the deadline.

All tasks must be submitted accordingly in order to pass the assignment.

Assignment 3

The third assignment covers OpenMP in shared memory systems. OpenMP-enabled compilers should be available in all modern development systems, such as with the default compiler under Linux and MacOS X (gcc –fopenmp)

Task 3.1: Decrypt with OpenMP

Develop an OpenMP-based command line tool that performs a brute-force dictionary attack on Unix crypt(3) passwords. One of the users has a password exactly matching one dictionary entry. A second user has a password build from one of the dictionary entries plus a single number digit (0-9) attached, e.g. “Abakus5”.

It is recommended to start with a serial version of your program, and add the OpenMP parallelization as the last step.

Please note that the first two characters of the encrypted password string in the *pw.txt* are the salt string used in the original encryption process. A correct solution therefore splits the encrypted password string into salt and encryption payload, calls some *crypt(3)* implementation with the salt and all of the dictionary entries, and checks if one of the crypt results matches with an entry from the user list.

Input

Your program has to be named “decrypt” and has to take two arguments, the name of the *password file* as the first and the name of the *dictionary file* as the second command line argument.

```
Example: ./decrypt ../../pw.txt ./../dict.txt
```

Output

The program must terminate with exit code 0 and produce an output file with the name “output.txt” in the same directory. This file has to contain nothing but the users whose passwords could be decrypted with the dictionary. Each line of the result file has to be a combination of username and decrypted password, separated by semicolon:

```
User01;pass
User02;Abakus5
```

Submit a compressed archive with the OpenMP sources as a solution. Beside the source code, the archive can also contain a file named “output.txt” with the cracked users for the example data. In this case the validation step will tell you if you found the right ones. Please do not let the validation machine perform the cracking of the example data, since this may take several hours.

Task 3.2: IEEE Floating Point standard

Make yourself familiar with the main cornerstones of the IEEE Standard for Floating-Point Arithmetic (IEEE 754). What aspects of the standard should highlighted for applications in parallel computing use cases? Please submit a text file and list 5 aspects that are covered by the standard. Write 2-3 sentences and reason the relevance with respect to parallel computing.

Task 3.3: Matrix Multiplication with OpenMP

Matrix multiplication is an elementary operation for many algorithms and use cases. As a matter thereof, a large portion of the Basic Linear Algebra Subprograms (BLAS) is built around the operation. The goal of this task is to compute the matrix product C (*n* x *p*) from the two input matrices A (*n* x *m*) and B (*m* x *p*) using OpenMP. Although several specialized approaches exist for sparse matrices and matrices of odd dimensions, your task assumes balanced dimensions.

Input

Your program has to be named "matmul" and has to take five arguments, the *file containing matrix A*, the *file containing matrix B*, as well as the dimensions n, m and p.

```
Example: ./matmul a.csv b.csv 3 3 3
```

Matrices are stored in CSV files (delimiter: ","), using the decimal point notation:

```
a.csv

0.929089,0.927045,0.325209
0.860091,0.848562,0.0269771
0.91075,0.395736,0.593213

b.csv

0.909713,0.17103,0.521874
0.363113,0.617162,0.973297
0.613226,0.0434021,0.309842
```

Output

The program must terminate with exit code 0 and produce an output file with the name "output.csv" in the same directory. The file "output.csv" must adhere to the format of the input files.

```
output.csv

0.845204,0.158553,0.169718
0.31231,0.5237,0.0262567
0.558496,0.0171758,0.183802
```

Further Remarks

For the purpose of comparability, please use single precision floating point data types. If you want to add low-level optimizations, note that the execution platform uses ARM v7l CPUs, not x86_64:

https://www.dcl.hpi.uni-potsdam.de/submit/machine/36/