Parallel Programming Concepts WS 2011 / 2012

Assignment 2

(Submission deadline: Jan 15th 2012, 23:59 CET)

General Rules

Each of your solutions has to fulfill the described application functionality (‚correctness') regardless of the underlying hardware. In order to test that, we will run your software on a FutureSOC lab machine with > 12 cores. A solution is correct if your program runs with the (directly or indirectly) configured number of threads and terminates without errors. After finishing, your program must terminate with exit code 0.

Two out of four tasks must be solved correctly in order to pass the assignment. Your submitted solution must contain only the **source code C files, a Makefile and the requested ready-to-execute binaries** for Windows 64-bit or Linux 2.6 64-bit. GUI applications are not allowed. Programs expecting any kind of keyboard input at any point of the execution are not allowed. Documentation should be done inside the source code. Students can submit solutions **in the established teams from assignment 1**.

Please submit your solution before the given deadline at https://www.dcl.hpi.uni-potsdam.de/parProg/. Incorrect submissions (missing source code, missing inline documentation, missing binaries, non-functional binaries) are rejected, with one possibility for re-submission.

For the OpenCL tasks, you may use thread-based implementations of the API if you do not own the neccessary hardware. OpenMP is by default supported in all major compiler products.

Task 2.1

Develop an OpenMP-based command line tool that searches a file for a given string. OpenMP-enabled compilers should be available in all modern development frameworks, for example with the default compiler under Linux and MacOS X (`gcc -fopenmp`).

Your program should take two arguments, a search string and a file path:

```
./pargrep Foo ./input.txt
```

The program should first read the complete file into memory (yes, normally you wouldn't do that). After that, the program should scan the buffer for the given string, and count the number of occurrences. Beside that, the program should also count the number of lines in the document, and output both numbers at the end of execution.

It is recommended to start with a serial version of the program, and add the parallelization with OpenMP afterwards. Test both versions of the application with different input file sizes. What is the investigated performance improvement / decrease with the OpenMP version ?

Submit the OpenMP version of the sources and the according binaries as solution. Add your short analysis summary as comment lines in the source code file.

Task 2.2

Develop an OpenCL version of the program described in Task 2.1.

Task 2.3

Develop an OpenMP - based command line tool that can be used to perform a brute-force dictionary attack on Unix `crypt(3)` passwords. The password file to be attacked is available at http://www.dcl.hpi.uni-potsdam.de/teaching/parProg/task2pw.txt. Each line of the password file contains the user name and the encrypted password, separated by the character „**:**“.

Your program should use the dictionary file available at http://www.dcl.hpi.uni-potsdam.de/teaching/parProg/task2dict.txt. One of the users has a password exactly matching to 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*“.

The submitted binary should take the password file as first, and the dictionary file as second command-line argument.

It is recommended to start with a serial version of your program, and add the OpenMP parallelization as last step. Only the OpenMP-based source code and binary must be submitted as solution. If feasible on your development machine, find both users and their clear-text passwords, and add them as comment to your submitted source code file.

Please note that the first two characters of the encrypted password string 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 the `crypt(3)` system function 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.

Task 2.4

Develop an OpenCL version of the program described in Task 2.3. In order to solve the task, you need to port the `crypt(3)` function to a kernel implementation.