Project Descriptions for Team 1 and Team 2 - CSE 436/536, Winter 2016

For final projects for team 1 and team 2, each of you is given one application and you will need to provide total 6 parallel versions of the application by using OpenMP, Cilkplus and C++multithread. The initial sources files for each application will be given, which also contains the OpenMP version using parallel for. So you will implement the other 5 versions and perform analysis for the their performance, speedup and efficiency on multicore machines. The descriptions of the 6 versions are as follows:

**OpenMP:** parallel for and task in a loop

**Cilk:** cilk\_for and spawn/sync

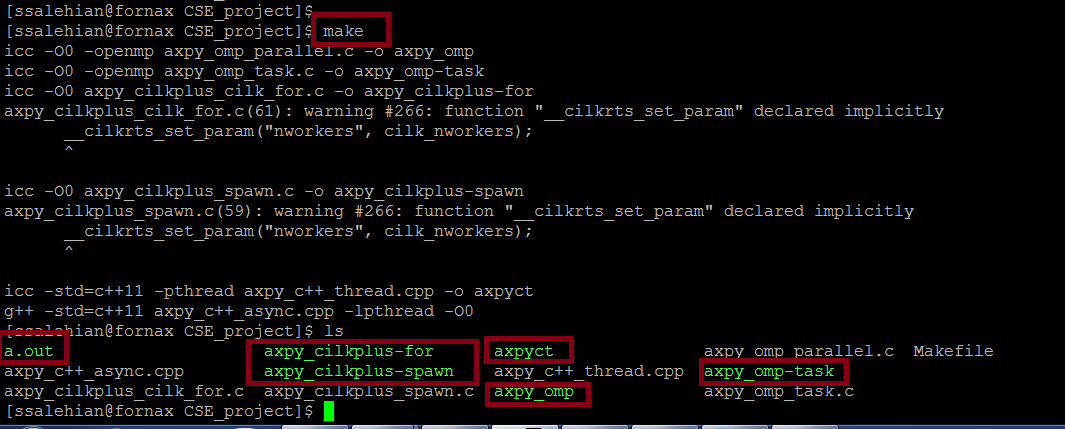
**C++:** thread/join and async/future

This application is taken from Rodinia Benchmark you can find more information about application in the link below:

https://www.cs.virginia.edu/~skadron/wiki/rodinia/index.php/Rodinia:Accelerating\_Compute-Intensive\_Applications\_with\_Accelerators

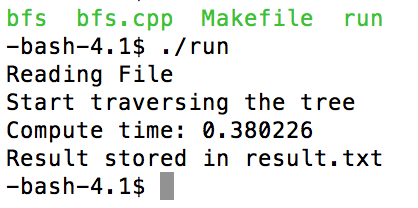
One sample application (axpy) is provided that includes all the six versions for your reference. You can download the source files and input data files from github (<https://github.com/passlab/CSE536_project_applications>). The data folder provides input data for some applications. Each application folder has makefile, source codes and a run script. makefile is the receipt to compile the source file and you can customize it to use with your implementation. The run script includes the command to execute the code. You may need to copy the data file to the right location to make the run script working (or change the run script).

You should compile and run your program using the command given in the following screenshots. –O0 is meant to tell compiler to turn off optimization so all our performance gains are from parallelism. Make sure you set up the environment by sourcing the ‘ . /opt/intel/bin/iccvars.sh -arch intel64 -platform linux’ script before you use icc (you only need to source it once each time you login to the system). For each version creates a separate file and follow the steps bellow to compile and run the codes:

1. There is a makefile in the application folder, in order to compile all six codes use this make file as below:

When you create a new version of the program you need to modify the makefile for the new version. It will help you to automatically compile your code. However, you still can use icc –O0 -fopenmp axpy.cpp axpy as before from commandline.

1. The run file inside the application folder is used to set the no of thread and problem size automatically. So, after compiling your code use ./run to run the cod.



Feel free to search for and design the best implementation you can, however you need to explain in details in your report why one of your version is better than the other in terms of performance and scalability. You can run your program on either of the two machines (lennon.secs.oakland.edu or paul.secs.oakland.edu) for up to 32 threads in a sequence of 1, 2, 4, 8, 16, 32 and collect your performance results. You need to submit a graph which compares runtime of all versions.

To ensure the project success, you should follow the following timeline and submit the required files on time to moodle.

|  |  |  |  |
| --- | --- | --- | --- |
| Phase | Deadline | Tasks | Submission |
| 1 | 03/16 | OpenMP implementation, both parallel and tasking, and performance report and application descriptions, and performance analysis using Intel vtune performance tool. For application description: max 1 page covers the description of the application, its domain usage, the challenges of parallelizing it, and related work. | Source files including Makefile and at least 3 page report |
| 2 | 03/30 | Cilkplus implementation, both spawn and cilk\_for, and performance report, performance comparison and analysis with OpenMP version using Intel vtune performance tool | Source files including Makefile and at least 2 page report |
| 3 | 04/18 | C++ implementation: both thread/join and async/future, and performance report, performance comparison and analysis with OpenMP and Cilkplus version using Intel vtune performance tool. | Source files including Makefile and at least 2 page report |
| 4 | 04/25 | Presentation and final project report due. | All source files and final report and presentation |

**Submission:** Your final submission should include all the sources and a report in Word editable format. The report should include:

1. One page description of the application, its domain usage, the challenges of parallelizing it, and related work.
2. Description on how you implement each of the six versions functions
3. Performance report should include one table that collects all the performance numbers of each of the six versions, and two figures when running with the problem size you choose. Figure 1 report the execution times in ms for each of the three implementations with 1, 2, 4, 8, 16 and 32 threads. X-axis will be for the number of threads, and Y-axis will be the execution time in ms and your figure should have six lines. Figure 2 report the speedup and efficiency when running 1, 2, 4, 8, 16 and 32 threads (X-axis will be for the number of threads and Y-axi will be the speedup and efficiency). Speedup is calculated as the ratio of sequential execution time (1 thread) to the parallel execution time (2, 4, 8, 16 and 32 threads) and efficiency is calculated as the ratio of speedup to the number of threads. You can use Excel sheet to record and plot the results, and generate figures for your report (see below a sample figures of speedup and efficiency). *In your report, you should explain the reasons of the performance differences between different versions, and the speedups of each program when increasing the number of threads.*
4. While the development can be done from your laptop or any other computers, the results in the report should be collected from one of the two machine we listed before. For information to access the machine, please refer to <http://cto.secs.oakland.edu/docs/pdf/linuxServers.pdf> , and let me know if you need help to access (you need VPN to access those machine from home, check <http://secs.oakland.edu/docs/pdf/vpn.pdf> ). Please indicate in your report, which machine you use. Please be noted that the machine is shared resource, overloaded use of the machine the last day of the machine may cause inaccurate performance results.
5. Explanation of the performance results shown in your figures and draw meaningful conclusions.

**Grading:**

**Functions implementations: 60 points (20 for each of the three program).**

**Report: 40 points.**

**For non-compliable code, you only receive max 60% of function implementations points. For compliable, but with execution errors and incorrectness, you receive max 70% of function implementation points.**

Assignment policy:

Programming assignments are to be done individually. You may discuss assignments with others, but you must code your own solutions and submit them with a write-up in your own words. Indicate clearly the name(s) of student(s) you collaborated with, if any. Although homework assignments will not be pledged, per se, the submitted solutions must be your work and not copied from other students' assignments or other sources.

You may not transmit or receive code from anyone in the class in any way--visually (by showing someone your code), electronically (by emailing, posting, or otherwise sending someone your code), verbally (by reading your code to someone), or in any other way.

You may not collaborate with people who are not your classmates, TAs, or instructor in any way. For example, you may not post questions to programming forums.

You may search the web and use any information that you find. However, you cannot take more than two lines of code from an external resource and actually include it in one of your assignments. Changing variable names or rewriting code you find does not void the "two line rule."

Any violations of these rules will be reported to the honor council. Check the syllabus for the late policy and academic conduct.