CS 133 Spring 2011

Semester Project

(presentations due in class/discussion Friday May 27th, Wednesday June 1st, and Friday June 3rd – implementation due to CourseWeb Friday June 3rd at 11:59pm)

Arbitrary-precision integer arithmetic operations (or "big-integer" arithmetic) are mathematical calculations that can be performed on integer numbers of any size (limited only by the memory constraints of the system). For instance, families of algorithms such as public-key cryptography often perform arithmetic operations on numbers with hundreds or thousands of digits. Such calculations are often parallelizable, and can thus be accelerated on parallel architectures.

In this semester project, you will work in groups of 4-5 students total to implement a small library of operations for arbitrary-precision integers. You are welcome to choose groups however you like, subject to this constraint: there shall be no more than 2 graduate students in any given group. This way the graduate students will be spread among the groups more evenly.

You will implement both a sequential and parallel version of these parallel operations, as well as the data structure which represents the arbitrary-size integers. You will be given much flexibility in this assignment, including the underlying integer representation and the parallel programming framework/platform you wish to use.

Your library must support the following operations:

- Addition
- Subtraction
- Multiplication
- Division
- Left shift
- Right shift
- Bitwise AND
- Bitwise OR
- Bitwise XOR
- Bitwise Complement (aka "NOT")
- Equality test ("==")

Choices of parallel programming framework/platform include…

a) OpenMP (on lnxsrv machines or other)
    - You may use any platform that supports OpenMP to measure performance improvement between parallel/sequential implementation.

b) POSIX threads (on lnxsrv machines or other)
    - You may use any platform that supports pthreads to measure performance improvement between parallel/sequential implementation.

c) CUDA or OpenCL (on **your own machine!**)

- You may use any programmable GPU platform available to you to implement this library in CUDA or OpenCL. There is no official class support for programmable GPUs, so **ONLY TAKE THIS OPTION IF YOU HAVE SUCH A PLATFORM HANDY!** Be forewarned…your teammates might need access to this platform also, depending how work is divided.

Finding the parallelism (where available) in each of these operations is an exercise for your team, and you may use any available books or web resources to assist your solution, provided that you cite them in your report.

The deliverables for this project are broken down into three categories, as follows…

1: PRESENTATION

Using either MS Powerpoint, OpenOffice, PDF, or similar, you will be designing a 10-15 minute talk discussing the results of your project. It is suggested that you follow the outline below, but all sections mentioned are required parts of your presentation (see Grade Criteria further below).

1. Introduction: mention all team members
2. How it is going (or how it went, if you are finished)? How did you divide tasks among team members and schedule work? Where did you deviate from the schedule?
3. What kind of parallel programming framework/platform did you use? Why did you select it?
4. Where did you get stuck? Bugs/challenges?
5. Data comparing parallelized/sequential versions of code – along with analysis of results… why did you encounter the speedup (or lack thereof) that you did?

Following this talk, there will be a brief (~5 minute) Q&A where your peers and instructor may ask questions for clarity.

It is suggested that you time your presentation to make sure it fits within the 15 minute constraint. Guidelines often state that you should plan to spend 1.5 to 2 minutes per slide (not counting outline or title slides) but this should be customized to the way you present.

You may make use of the chalkboard as appropriate. Animations on the slides are neither required nor necessary, but it is suggested that you use any visual aide that is appropriate to clarify your presentation.

Presentation Dates:

Friday May 27th (9th week), Wednesday May 5th (10th week) and Monday May 10th (10th week) are candidate days on which your team may present.

Your team must email me by Monday May 23rd (9th week) and let me know which day you wish to present! Slots will be allocated on a first-come, first-serve basis. There should be time for roughly 5 or 6 team presentations on each day.

On the same day of your presentation, please email me your slides (at kaplan@cs.ucla.edu). You may use your laptop, my laptop, or the computer in the classroom to give your presentation. If you are using my laptop, please bring presentation on memory stick or email to me beforehand. Also, if you are using my laptop or the room computer, you will want to either export your presentation to Powerpoint (.ppt, .pptx) or PDF format.

NOTE

All team members' names must be on the first slide (or cover slide) of the talk. All team members must contribute to this presentation, even if they are not talking. All team members should be present during their group's presentation.

2: SOURCE CODE SUBMISSION

Please submit to CourseWeb a tarball (.tar.gz) file containing

- Your source code in addition to compiled executables for the platform of your choice
    - please maintain the directory structure of your source code as it was organized on the machine it was compiled on
    - please include any input files necessary to test your code
- In the root directory ***two additional files*** are needed:
    - **Writeup.doc (or .docx, .txt, .html, .htm, .pdf)**: see Part 3 below…
    - **README.txt**: a small ASCII file explaining…
        - the directory structure of the tarball (i.e. which files are located where)
        - how to compile this source code… step by step instructions using the compiler and platform of your choice
            - What compilers are needed? What type of platform will compilation work on? What command-line arguments are needed, if any? (This explanation should be similar to those on the lab assignments.)
        - where the executable files are located (for instance, are they in a directory named …/bin/ ?)
        - where the input files are located
        - how to execute this program from command-line

3: PROJECT WRITEUP

This section describes the file Writeup.doc alluded to in Part 2 above. As mentioned there, this file may be an MS Word document (or prepared in another word-processing application and exported to MS Word). This file may also be an ASCII (.txt) or HTML (.html or .htm) file. Please make sure it has one of these extensions: .doc, .docx, .txt, .html, .htm, .pdf

This project writeup is the last chance for your team to address the project implementation. Even though its contents are similar to the presentation you'll be performing in class, the status of the implementation that it describes is considered final. For example, even though there may be some tasks left to finish at the end of the presentation you give in class during Week 9 or 10, any tasks which are described as "not finished" in the project writeup will simply not be finished in this class.

The project writeup should contain the following discussion, which may be concise, but should include enough detail for the reader to understand exactly what was implemented and how. Here are the sections required by the project writeup…

- **Guide to source code**
    - Does the source code compile or not?
    - Which sections of the code work (i.e. are complete) and which don't?
    - Where in the code is the thread code?
        - If using pthreads, where are the thread functions?
        - If using OpenMP, where are the parallel pragmas located?
        - If using CUDA or OpenCL, where are the kernels?
- **Project breakdown**
    - Why did you choose to parallelize the code in this way?
    - Was synchronization needed (in the form of locks/mutexes, condition variables, critical sections, barriers, etc)? If so, where was it needed?
    - Which sections of the source code did each team member work on?
    - What features of the implementation needed to be sacrificed? Which were completed and which were not?
- **Bugs & Challenges**
    - What bugs did you run into, if any? Did your team get stuck on any part of the implementation?
    - What challenges existed in using this framework?
        - For instance, did you have to learn an API not covered in class?
        - Did you run into difficulty extracting good performance from the threads?
    - What have you learned about parallel programming from this project?
        - For instance, how long does it take to debug this program versus a typical sequential program?
        - (…any other observations as you desire)
- **Final Results**
    - How did the parallelized code compare to a sequential (or single-threaded…or non-GPU-enhanced…) version of the code in performance?
    - Please include sample output of your code.
    - How fast can your parallel code process inputs of different sizes?
    - How does the performance of your parallel code scale with different numbers of threads? (This will depend on the platform you use, but you should be able to use at least up to 8 threads on the lnxsrv machines.)

GRADE CRITERIA

- **Presentation (17 points out of 100)**
    - You showed up and gave a talk in class with your team: **2%**
        - Attendance during team talks is mandatory even if you're not talking. You (the individual) will miss these points if you are not present for your team.
    - The slideshow is clear and informative. Slides reflect information in a well-organized fasion: **6%**
    - Presentation ability. Team members speak clearly and audibly, and are able to convey facts and answer questions: **7%**
    - Each member of the team contributed to the project: **2%**

- **Implementation (83 points out of 100)**
    - **Source Code Submission (50 pts)**
        - Tarball was submitted containing the files and directory structure requested: **10%**
            - Yes once again, you reap rewards for following directions. ☺
        - README.txt sufficiently describes directory structure, compilation method, and how to test code: **10%**
        - Source code compiles successfully: **10%**
        - Project can be tested/executed successfully by following instructions in README.txt and using inputs included in the tarball: **20%**

    - **Project Writeup (33 pts)**
        - Guide to source code: **8%**
            - All information requested is presented
        - Project breakdown: **5%**
            - All information requested is presented
        - Bugs & challenges: **5%**
            - A short discussion of how this project went for your team is presented
        - Final result: **15%**
            - Graph, quantitative time comparison table, or any other type of result comparing the parallel implementation to the sequential version. **Note: show results for different input sizes as well as different thread-counts!**