CMPUT 481 LEC A1 – CMPUT 681 LEC A1 – Fall 2022

Assignment #1: Sh (2022)

CMPU arallel and Distributed Systems

Paul Lu

Fall 2022

Assignment: Shared-Memory Programming:

Parallel Sorting by Regular Sampling

Due Date: Friday, Oct. 7, 2022, 11:55 PM, via eClass

(Penalty-free extension to Monday, October 10, 11:55 PM, via eClass)

Overview:

There are three parts to this assignment: implementation, experimentation, and exposition.

1. Implement Parallel Sorting by Regular Sampling (PSRS) using C/C++ and shared-memory pthreads (POSIX threads) library on any Linux or Unix-based system of your choice.
2. Experiment with the basic algorithm. For example, I recommend that you try sorting an array of integers of at least 32M keys or larger (possibly much larger, depending on the speed of your cores).
3. Describe your implementations and experiments in a **3-page report** (excluding any graphs, diagrams, or source code listings). The goal is to present a **concise, coherent, and insightful** description of the lessons learned from your experimentation; do **not** exceed 3 pages on the report.

NOTE: Please do **not** put your code on a public git repository. This assignment is largely the same as the one given to my class in previous years. I have complete confidence that you will **not** seek out a solution from a previous student of the course. Please do not compromise this learning opportunity (and commit an Academic Offense) by not doing the work by yourself!

Details on Implementation:

Again, you should implement PSRS using shared-memory pthreads.

When debugging your programs, you will likely create a lot of runaway processes. Learn about `ps`! **Be sure to clean up your threads and processes,** since they will interfere with your benchmarking, or interfere with other users (if any).

If you search hard enough, you will likely be able to find existing implementations of PSRS on the Internet. Please do the implementation yourself in order to get the hands-on experience.

HINT: You may find the `pthread_setconcurrency()` function to be useful to achieve maximum speedup.

HINT: Use command-line arguments to parameterize the number of threads used (e.g., p) and the number of keys to sort (e.g., n). This will make experimentation much easier.

Details on Experimentation:

1. When testing your code, do **not** use input files for test data. Instead, randomly generate the data within the program.
   If you use the same seed value (i.e., `srandom()`) for each run, then `random()` will **deterministically** generate the exact same sequence of pseudo-random numbers. Notice that `random()` returns a value of type `long int`. **NOTE: Do NOT use `srand()` and `rand()`. The range of values generated is considerably lower than from `srandom()` and `random()`.**

   And, by generating the data within your program, it will run faster.

2. One should never report performance based on single runs. Although imperfect, I recommend you do 5 runs for each datapoint in your results and use the average of those runs. As well, there are usually process or experiment start-up overheads, so you might consider doing 7 runs and average the last 5 of those runs.

3. Vary the size of the array of integers to be sorted.

4. You should use the Unix function `gettimeofday()` or something similar to measure time.

5. Speedups of up to p=4 are acceptable. Speedups for p > 4 are appreciated.

Details on Expositi eport):

This should be a well-written, clear, our implementations and experiments. Use appropriate graphs, figures, and source code listings. For example:

1. Speedup graphs
2. As appropriate, breakdown o o phases
3. Effect of varying the input da

The report is **not** just a description of your implementation. The main purpose is to communicate the lessons learned from your hands-on experience with the implementations and experimentation. Think of this as a mini-paper where you have a thesis (i.e., a point that you would like to make) and you present data and justification for your thesis.

What to hand in:

1. A title page for your assignment with: (1) your name, (2) student number, (3) CCID
2. A 3 page typed report **in PDF format** (single spaced, 11 point font, all margins at least 1 inch). Figures, graphs, and diagrams do not count as part of the 3 pages. Do NOT submit Microsoft Word or other word processing file formats; PDF please.
3. Your source code, Makefile, and any relevant test data.

Marking:

The assignment is worth 15% of your final mark in the course. This is an individual assignment. Do not work in groups. Do not share source code. However, you may freely discuss this assignment at a ``high level'' with your fellow students.

Other comments and hints:

1. Please do **not** put your code on a public git repository.
2. Be careful about speculating and giving theories as to why performance is poor (or good) without some evidence. Wild, unsupported (and wrong) theories undermine the writer's credibility.
3. Be careful of poor implementations of sequential sorting; they tend to make your speedups look overly high. Complicated array indexing and dereferencing can **really** slow down the code.
4. How did you validate the correctness of your sequential and parallel codes?
5. Use at least `-O` compiler optimization when benchmarking. This reduces the chance that a silly bit of code (which a compiler could optimize out) can distort the timings. You should never publish (or put in your thesis) numbers generated with `-g` specified to the compiler.
6. Give me speedups graphs!!! (In addition to other kinds of graphs.)
7. Include listings of your souce code!!
8. In addition to speedups, give me the real time (e.g., in seconds) of your runs. Knowing the actual timings helps to put the experiments in perspective.
9. For Phase 1, use Quicksort, which is a library function.
10. For Phase 4, do **NOT** use Quicksort; do **NOT** use Mergesort. Phase 4 must be a merge (not even a Mergesort), since a merge of partially ordered lists is much faster than either Quicksort and Mergesort. A k-way merge or successive 2-way merge would be fine.
11. Do **NOT** draw too many performance-related conclusions based on **short runs**. If your timing measurements are for, say, less than 2 seconds, then non-determinism and other overheads can greatly skew your results and conclusions. To be sure, time measurements under 2 seconds can be done, but they must be done carefully.

Useful Resources

1. Your recommended textbooks.
2. Google for online tutorials on Pthreads.

Last modified: Wednesday, 7 September 2022, 2:51 PM

◀ Paper for Discussion, September 13: Parallel Sorting by Regular Sampling (1993)

Jump to...

Lecture Slides Part 2 (PDF, 350 KB) ►