Freestyle Assignment 20P+5P

submission date (proposal): 11 Jan (12:00)
submission date (implement.): 13 Feb (23:59)
window (presentation): 16-25 Feb

For this assignment, each of you will construct their own exercise. This works in the following way:

1. Decide on a problem.
2. Design an exercise sheet (similar to the ones that we have had).
3. Send us the exercise sheet for feedback, and to sign off on it (until 11 Jan, preferably before).
4. We will reply to you with some mandatory changes.
5. You send us an improved version (until 16 Jan).
6. You implement your exercise (until 13 Feb).
7. There will be a presentation of your projects sometime between 16 and 25 February (will be finalized later). This presentation is part of your grading.

5.1 Defining your Exercise

Try to mimic the style of our assignments:

- Use either a PDF or a plain text format (*.txt), if you want to use pictures, just append them to the email, and reference them in the exercise.
- Each exercise should contain an interesting parallelization.
  - nothing you can do with a simple OpenMP pragma
  - there should be interaction between threads
  - think about atomicity vs. locks (you should at least need one)
- Make sure to have specific goals. This way you can easily show us that your implementation satisfies your goals.
- Split your task in logical steps, and distribute points evenly (otherwise not achieving one goal might cost you a lot of points).
- Each task should be very specific in what you expect to do to solve it.
- You should be able to argue why you expect your improvements to work. We will not accept unrealistic proposals.
- If possible, point out different possible solutions. It might be advisable to implement several solutions and compare them.
- Creating interesting problem instances for your tests might be worth some points. If you do this:
  - define characteristics of interesting instances
  - implement a pseudo-random instance-generator that can create interesting instances
  - do not check in the datasets themselves, but give the used inputs for your generator
  - make sure that you can generate instances with varying characteristics (as defined above)
- Include some points for interesting evaluations.
- The points in your exercise should add up to 20 points.

Send us your exercise before 16 Jan so we can sign off on it. Expect us to make some changes or veto the first version of your proposal.

5.2 Implementation (20)

This is what you get your points for: Implement your exercises and evaluate your implementation. When something does not work as expected, talk to us/write an email and specifically argue why your original idea does not work. Are there other options? Something not working as expected might be a result in and of itself?

5.3 Short Presentation (5)

Prepare a 15 minute presentation, in which you explain the problem, show one or two implementation highlights, and explain your evaluation to the others. These presentations should give everyone an overview of what techniques you used and which techniques gave you the best results.

Some Inspiration

just look at papers that are out there We would advise you to either look for papers about concurrent data-structures or parallel algorithms. Possible conferences might include *Princilples and Practices of Parallel Programming* (PPoPP), *Algorithm Engineering and Experiments* (ALENEX), and to a lesser extent Euro-Par and *International Parallel and Distributed Processing Symposium* (IPDPS; they do a lot of MPI stuff which is not part of our lab course). Pay attention to the implementability of your algorithms/papers.

ref: PPoPP 2021 list of papers
ALENEX 2020 list of papers
Euro-Par 2021 list of papers
IPDPS 2021 list of papers

look at common libraries Another good inspiration is to look for libraries and see what is implemented there. In particular, Intel's TBB library offers a wide range of functionality that could be reimplemented. This is especially thankful, because you can use their implementation as a competitor (for experiments). Additionally, some parts of the library might be open source, giving you an idea of how to implement your plans.

Good ideas might be a concurrent memory allocator, one of the concurrent datastructures, a concurrent algorithm (especially the parallel pipeline), or a simplified task scheduler.

hazard pointer library. Deallocating shared resources in a lock-less parallel setting. Think about a growing concurrent data-structure. Once all elements have been copied to the new structure the old one should be replaced and deallocated. In the worst case, there is still one sleeping thread, which will access the data-structure using an old pointer, which it has read before the change. Most solutions to this problem use a common counter for each data-structure, but this can be a huge bottleneck limiting speedups to constant factors.

One difficulty is to design an interface that is easy to use and hard to misuse. Another difficulty is to also construct a meaningful benchmark that can show the performance of your solution. This benchmark should be discribed, in detail, within your first proposal.

ref: hazard pointer
Fast and Robust Memory Reclamation for Concurrent Data Structures

concurrent caching data-structure A data-structure that you can concurrently access files over, evicting old files when too many files are loaded (too much memory). Think about the use of shared pointers, to make sure no file is deallocated while it is accessed. You should also implement and benchmark different eviction schemes. Similar to the hazard pointer task there is some necessity to create meaningful benchmarks.

As an optional task you could try different scalable memory allocators (e.g. TBBmalloc, jemalloc, tcmalloc), because the basic malloc implementation can be to slow to handle many small allocations.

ref: article on scalable allocators

one parallel algorithm - to compare different parallelization techniques Take an algorithm that is not trivially parallelizable, but also not too hard; maybe look for complex multi-dimensional dynamic algorithms. Parallelize this algorithm using multiple techniques such as OpenMP and different methods of using `std::thread`s. Each implementation should fully utilise its capabilities, and they should be significantly different. Then benchmark all variants against each other. Which variants scale the best? Do the differences depend on the inputs?

parallel data-structures Concurrent data-structures are always a source for assignments, but make sure that there are interesting subtasks to your exercise. Many data-structures require global locks, or globally shared variables that lead to repeated false sharing. Make sure your plans are possible and there are no obvious bottlenecks that could hide your improvements. Maybe find a paper that you can implement; this simplifies the creation of experiments/inputs.