Project Proposal for Research on Oblivious Algorithms

15-300, Fall 2020

Mike Xu
https://mkpjnx.github.io/ObliviousSort/

November 6, 2020

1 Project Description

I will be working with Professor Elaine Shi and two other students from this class, Marcia Dai and Yuheng Guo, to develop a reference implementations for a novel oblivious sorting algorithm, and investigate its application as a building block for other algorithms.

Studies have shown that the memory access patterns of memory can reveal information about the accessed data, even if the data itself is encrypted [5, 10]. For the sake of user data privacy in the age of cloud computation, where data processing is done remotely, it is crucial to not only encrypt the data, but also obfuscate the access patterns of programs [4]. Goldreich and Ostrovsky [3] proposes using an **Oblivious Random-Access Machine**, or ORAM, to simulate memory accesses while obfuscating them by continuous rewriting and reshuffling. In their paper, Goldreich and Ostrovsky prove that obfuscating memory access pattern to be statistically indistinguishable to an adversary must incur a $\Omega(\log m)$ overhead, where m is the size of memory [3].

Much research has gone into developing and ORAM algorithms [6, 2], but a general purpose solution to make a program oblivious - and by extension the incurred overhead - is not necessary for specific tasks that a program may do. **Oblivious Data Structures**, such as oblivious trees, stacks and queues [9, 7]. What is important to note is that they are able to obtain better asymptotic performance than simply simulating the non-oblivious data structures and algorithms on ORAM.

The focus of our research will be implementing the oblivious algorithm for sorting outlined in Asharov et al. named **Bucket Oblivious Sort** [1]. In short, the algorithm shuffles the input obliviously, and sorts the shuffled input non-obliviously. This is still oblivious, because any input has equal chance of being shuffled to any another order, so the probability that any particular access pattern occurs during the subsequent sorting is still equal among all inputs. Furthermore, bucket oblivious sort achieves the optimal asymptotic performance of $O(n \log n)$ on input sizes of n.

The plan is for all three of us to work collaboratively on the reference implementation, so that we familiarize ourselves with the algorithm, and then diverge to focus on objectives that extend our research. One objective is to compare our implementation with other oblivious sorting algorithms, most notable one being bitonic sort. These algorithms are also able to achieve the optimal asymptotic performance, but the complexity incurred on these algorithms lead to "enormous constants that render them completely impractical" [1]. Bucket oblivious sort, on the other hand, is at most $3\times$ slower than merge sort, albeit with some caveats. In short, the shuffling portion of the algorithm has a failure probability based on the amount of local, secure memory (i.e. CPU registers) we can work with.

Like merge sort, Bucket Oblivious Sort is also quite parallelizable. Our second objective is to create a binary fork-join version of bucket oblivious sort as outlined in [8]. This primitive is a powerful tool for building oblivious *parallel* RAM (OPRAM), which can then serve as a general purpose solution for making other applications oblivious. Furthermore, the sorting primitive can be used in graph algorithms to make them oblivious [4, 8], and we aim to provide sample code for this, with some evaluation to investigate its overhead as well.

Finally, we aim to extend our findings in the project by exploring other algorithms that can be made oblivious, and propose approaches for these problems. One particular area of interest for professor Shi is that of machine learning. It would be a stretch goal to develop some ideas on how to make algorithms such as K-nearest neighbor oblivious.

2 Project Goals

2.1 75% Goal

Even if we are behind schedule, we will aim to accomplish the following

- Create a reference implementation for Bucket Oblivious Sort as described in [1]
- Produce performance evaluation and compare against existing oblivious sorting algorithms, such as bitonic sort
- Create the binary fork-join version as described in [8]

2.2 100% Goal

If we are on track, we will aim to accomplish the following

- All of the above
- Apply oblivious sort to the graph algorithms described in [8]
- Prove the obliviousness of the resulting algorithms
- Evaluate the incurred overhead

2.3 125% Goal

If we are on track, we will aim to accomplish the following

- All of the above
- Apply parallel oblivious sort to build a reference implementation of OPRAM
- Propose approaches for oblivious machine learning algorithms

3 Milestones

3.1 First Technical Milestone

By the end of the semester, we would like to create a preliminary reference implementation for bucket oblivious sort. If it is not functional, it will become next week's goal.

3.2 Biweekly Milestone

The tasks from these milestones will be split among the three of us, and are subject to change based on actual progress and potential changes in direction for the project.
February 15th: Test and evaluate bucket oblivious sort implementation and design tests to compare against bitonic sort.
March 1st: Investigate Cilk or equivalent parallel framework and begin planning on binary fork-join version, deliver preliminary results for comparison with bitonic sort. Begin work on using oblivious sort in graph algorithms.
March 15th: Complete parallel version and design evaluative tests, continue work on graph algorithm implementation.
March 29th: Begin evaluation of parallel version of oblivious sort. Complete graph algorithm implementations.
April 12th: Begin evaluation of graph algorithms, using both the sequential and parallel versions as primitives.
April 26th: Produce evaluations for graph algorithms and their span analysis
May 10th: Investigate stretch goals such as building an OPRAM implementation.

4 Literature Search

As shown below, we have collected a number of background papers as well as related works to guide our research. As we continue, we will dive further into different areas and consult relevant literature - many of which are cited by the papers we are currently reading.

5 Resources Required

To the best of my knowledge, no further resource is required beyond a computer with a C++ compiler and support for a parallel programming framework such as Cilk or OpenMP. To evaluate performance, we may require more computational power. We will consult professor Shi regarding access to a computing cluster such as the Latedays cluster in CSD.

References

[1] Gilad Asharov, T.-H. Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. Bucket oblivious sort: An extremely simple oblivious sort.

[2] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. OptORAMa: Optimal oblivious RAM.

[3] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious RAMs. 43(3):431–473.

[4] Michael T. Goodrich, Olga Ohrimenko, and Roberto Tamassia. Graph drawing in the cloud: Privately visualizing relational data using small working storage. In Walter Didimo and Maurizio Patrignani, editors, *Graph Drawing*, Lecture Notes in Computer Science, pages 43–54. Springer.

[5] M. S. Islam, M. Kuzu, and Murat Kantarcioglu. Access pattern disclosure on searchable encryption: Ramification, attack and mitigation. In *NDSS*.

[6] Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. On the (in)security of hash-based oblivious RAM and a new balancing scheme.

[7] Daniele Micciancio. Oblivious data structures: applications to cryptography. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 456–464. Association for Computing Machinery.

[8] Vijaya Ramachandran and Elaine Shi. Data oblivious algorithms for multicores.

[9] Xiao Shaun Wang, Kartik Nayak, Chang Liu, T-H. Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pages 215–226. Association for Computing Machinery.

[10] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. ISSN: 2375-1207.