2019 19th International Conference on Application of Concurrency to System Design (ACSD)

Lock-free Concurrent Data Structures and How to Model their Performance (Extended Abstract)

Philippas Tsigas
Deapartment of Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
tsigas@chalmers.se

Concurrent data structures provide the means to multi-threaded applications to share data. Common designs of concurrent data structures are based on locks in order to avoid inconsistency due to concurrent modifications. Locks though introduce a sequential component in Amdahls law. Lock-free algorithmic designs of concurrent data structures [1] were introduced in the quest for better performance and scalability and are widely used in practice, e.g. in the Intel's Threading Building Blocks Framework [2], the Java concurrency package [3] and the Microsoft .NET Framework [4]. Lock-free implementations provide indeed a way out of several limitations of their lock-based counterparts, in robustness, availability and programming flexibility. Last but not least, the advent of multi-core processors has pushed lock-freedom on top of the toolbox for achieving scalable synchronization.

Lock-free designs typically employ optimistic conflict control making performance analysis challenging. Lock-free implementations guarantee system-wide progress. In contrast to pessimistic lock-based approaches, processes do not signal their presence before the operation, work independently, break the operation into smaller parts, that gradually modify the data structure to the desired state, and check at the end of each part whether their independent work is invalidated. As a result, delays occur only if there is an actual conflict between concurrent processes. Because lock-free algorithms do not provide termination guarantees of an individual operation there is no worst-case bound on the execution time of an operation.

The common measure of performance for data structures is throughput, defined as the number of operations on the data structure per unit of time. To this end, this performance measure is usually obtained by considering an algorithm that strings together a pure sequence of calls to an operation on the data structure. However, when used in a more realistic context, the calls to the operations are mixed with application-specific code that plays a significant role in the performance of concurrent data structure. In order to cover various contention environments, the time complexity of the algorithms is often parametrized by different contention measures, such as point [5], interval [6] or step [7] contention. However, the worst-case behavior is not enough to express the performance that we observe in practice. A tighter estimate of contention is needed.

A number of promising first steps to derive analytical models that closely describe the performance of lock-free data structures observed in practice appeared recently in the literature [8]–[11]. These results take into consideration the underlying hardware. In doing so, these results start by creating abstractions of the lock-free data structures, the programs that call them and the hardware executing them. These abstractions are characterized by a set of parameters which capture the significant factors that influence the performance of lock-free concurrent data structures. Then, the system is mapped to an execution model, that can be different for different data structures and program settings. Examples of such resulting models include cyclic patterns, Markov chains, Poisson processes, queueing models in steady states under low and high contention. These models retain the initial performance-critical parameters. In a second phase, by analyzing the respective execution model throughput is computed.

These models have been validated in a broad spectrum of data structures implementations that cover queues, stacks, priority queues, hash tables, skip lists, deques, hash tables, binary trees, linked lists. The performance estimations by these models are close to what we observe in practice.

When it comes to practical implementations of lock-free concurrent data structures, there are some specific parameter e.g. back-off, padding, and memory management related ones that are crucial for the performance of the implementation. The value of these parameters is selected carefully after a costly,

Work supported by the Swedish Foundation for Strategic Research, Sweden, proj. Future factories in the cloud (FiC) grant nr. GMT14-0032, by the Swedish Research Council (Vetenskapsrdet), Sweden proj. Models and Techniques for Energy-Efficient Concurrent Data Access Designs grant nr. 201605360 and by the European Research Council under the European Unions Seventh Framework Programme (FP7/2013- 2016) / Grant agreement no. 611183, EXCESS Project.

978-1-7281-3843-5/19/$31.00 ©2019 IEEE
DOI 10.1109/ACSD.2019.00004

extensive, manual exploitation phase. It has been shown that these models can be used to fine-tune such parameters that are important for the performance of lock-free concurrent data structures. More specifically it has been shown how they can be used to: (i) design a new back-off strategy; (ii) optimize memory management mechanisms; (iii) resolve the impact of different memory alignment strategies.

***Index Terms*—Lock-free, Data Structures, Parallel Computing, Performance, Modeling, Analysis**

ACKNOWLEDGMENT

This extended abstract is based on recent joint work with Aras Atalar, Anders Gidenstam and Paul Renaud-Goud.

REFERENCES

[1] Maurice Herlihy and Nir Shavit, "The art of multiprocessor programming, Morgan Kaufmann, 2011.

[2] Intels threading building blocks framework. https://www.threadingbuildingblocks.org/ . Accessed: 2019-05-20.

[3] Java concurrency package. https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/package-summary.html. Accessed: 2019-05-20.

[4] Microsoft .net framework. https://dotnet.microsoft.com/ . Accessed: 2019-05-20.

[5] Hagit Attiya and Arie Fouren, "Algorithms adapting to point contention., Journal of the ACM (JACM), 50(4): 444–468, 2003.

[6] Yehuda Afek, Gideon Stupp, and Dan Touitou, "Long lived adaptive splitter and applications, Distributed Computing, 15(2): 67-86, 2002.

[7] Hagit Attiya, Rachid Guerraoui, and Petr Kouznetsov, "Computing with reads and writes in the absence of step contention, In International Symposium on Distributed Computing (DISC), pages 122-136, 2005.

[8] Aras Atalar, Paul Renaud-Goud and Philippas Tsigas, "Analyzing the Performance of Lock-Free Data Structures: A Conflict-Based Model, In the Proceedings of 29th International Symposium on Distributed Computing (DISC 2015), pages 341–355, Springer 2015.

[9] Aras Atalar, Paul Renaud-Goud and Philippas Tsigas, "How Lock-free Data Structures Perform in Dynamic Environments: Models and Analyses, In the Proceedings of the 20th International Conference on Principles of Distributed Systems (OPODIS 2016), pages 1–17, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[10] Aras Atalar, Paul Renaud-Goud and Philippas Tsigas, "Lock-Free Search Data Structures: Throughput Modeling with Poisson Processes, In the Proceedings of the 21st International Conference on Principles of Distributed Systems (OPODIS 2018), pages 9:1–9:16, Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik.

[11] Aras Atalar, Anders Gidenstam, Paul Renaud-Goud and Philippas Tsigas, "Modeling Energy Consumption of Lock-Free Queue Implementations, In the Proceedings of 29th IEEE International Parallel and Distributed Processing Symposium (IPDPS 2015), pages 229–238, IEEE Press 2015.