Splash-4: Improving Scalability with Lock-Free Constructs

Eduardo José Gómez-Hernández*, Ruixiang Shao*, Christos Sakalis†, Stefanos Kaxiras† and Alberto Ros*
* *Computer Engineering Department*, *University of Murcia*, Murcia, Spain
† *Department of Information Technology*, *Uppsala University*, Uppsala, Sweden
eduardojose.gomez@um.es, ruixiang.shaop@um.es, christos.sakalis@it.uu.se, stefanos.kaxiras@it.uu.se, aros@ditec.um.es

***Abstract*—Over the past three decades, the parallel applications of the Splash-2 benchmark suite have been instrumental in advancing multiprocessor research. Recently, the Splash-3 benchmarks eliminated performance bugs, data races, and improper synchronization that plagued Splash-2 benchmarks after the definition of the C memory model.**

In this work, we revisit the Splash-3 benchmarks and adapt them for contemporary architectures with atomic operations and lock-free constructs. With our changes, we improve the scalability of most benchmarks for up to 32 and 64 cores, showing an improvement of up to 9x in actual machines, and up to 5x in simulation, over the unmodified Splash-3 benchmarks. To denote the substantive nature of the improvements in the Splash-3 benchmarks and to re-introduce them in contemporary research, we refer to the new collection as Splash-4.[1]

***Index Terms*—Benchmarks, simulation, synchronization, atomic operations, optimization.**

I. Introduction

Splash-2 [9] is the first major parallel benchmark suite that proved instrumental in the development of shared memory multiprocessors. A significant body of work bases its observations on the behavior of the Splash-2 benchmarks, and this is still relevant and useful today [1], [2].

While Splash-2 shows unexpected and sometimes incorrect behavior when used within contemporary compilers and hardware, a relatively recent update, Splash-3 [8], exposes the data races and performance bugs in Splash-2 and solves these issues by properly synchronizing the benchmarks.

The driving consideration in the development of Splash-2 was to demonstrate shared-memory scalability. Indeed, Splash-2, under the evaluation techniques prevailing at the time (e.g., under a perfect memory system), showed near linear scalability in most of the benchmarks for up to 64 processors [9].

In the more recent Splash-3 work [8], using a more accurate simulation infrastructure, most of the Splash-3 benchmarks reach a speedup between 16x to 64x in a 64-core multicore. However, these results were obtained with an in-order processor model.

On actual processors (AMD EPYC 7702P, 64 Cores), most of the applications exhaust their scaling (i.e., show no further performance improvement) using somewhere between 4 and 16 cores, with a maximum speedup of 10x. Conversely, using the latest out-of-order core simulation models (mirroring the configuration of an Intel Skylake processor), the same Splash-3 benchmarks exhaust their scaling, somewhat higher than in the actual AMD hardware, but still in a range between 16 and 32 cores, and with a maximum speedup of 15x.

One problem is that Splash-2 and Splash-3 benchmarks are crafted using outdated programming techniques and, as a result, are overshadowed by the more recent PARSEC [3]. The later have the benefit of a vastly expanded set of programming tools, e.g., C11 atomics, since the time of Splash-2.

II. Updating Splash-3 Synchronization

We systematically count the shared variables accessed on each critical section using a custom Pintool [6]. Initially, we target critical sections that modify a single shared variable. These critical sections can be easily replaced by an atomic operation. Then, we look at critical sections that access a few shared variables and try to find their lock-free equivalent.

Modern ISAs (e.g., x86, some ARM ISAs) and programming languages (e.g., C, C++, Java) typically provide a basic set of atomic operations that can be used in place of critical sections, offering both atomicity and synchronization. This basic set consists of atomic loads and stores, atomic read–modify–write (RMW) operations (such as fetch–and–add), and some atomic comparisons and exchange operations (such as compare–and–swap, CAS).

Using the CAS construct, a custom RMW fetch–and–add operation [5] for double precision floating point numbers can be implemented. The RMW atomics are typically available only for integer types, especially when only considering lock-free atomics. CAS on the other hand is type-agnostic, so it can be used to implement RMW operations for more complex underlying types.

For the Splash-3 benchmarks, the execution time between barriers is fairly short. To minimize the overhead of the barrier operation, in Splash-4, we provide a sense-reversing barrier optimized for short waiting times [7].

III. Methodology & Evaluation

We use two different environments to execute the applications: (1) A modern machine, AMD Epyc 7702P CPU, with Hyper-threading enabled but running one thread on each physical core and (2) a state-of-the-art full-system simulator, gem5 [4], modeling an Intel Skylake-like processor.

[1] GitHub repository: https://github.com/OdnetninI/Splash-4

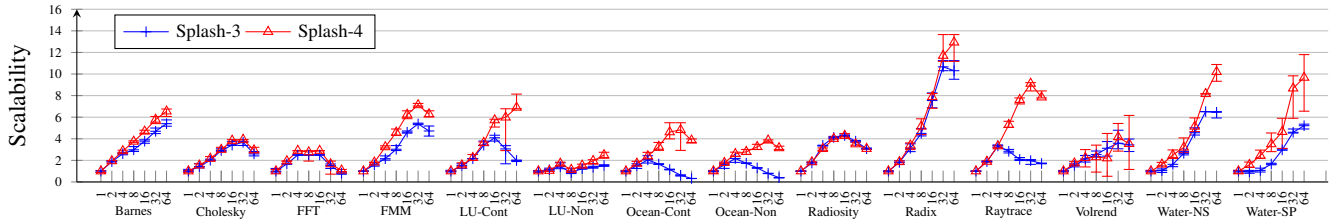

Fig. 1. Splash-3 vs Splash-4 Scalability on an actual processor

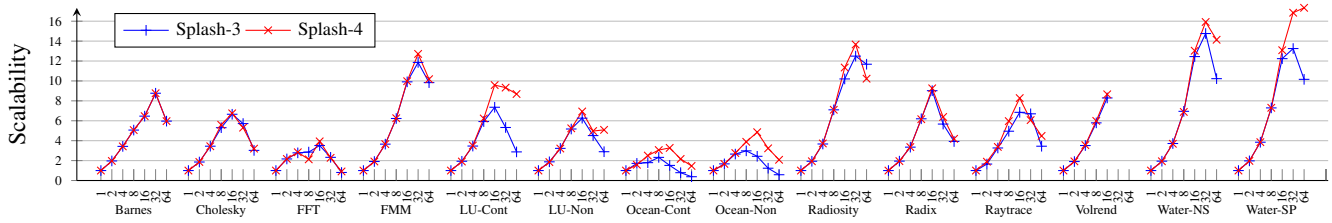

Fig. 2. Splash-3 vs Splash-4 Scalability in simulation

While, ideally, we would like for the results of these two systems to match completely, in practice simulators have inaccuracies that lead to performance differences. However, we expect the overall behavior to be similar in both.

We evaluate Splash-4 (sense-reversing centralized barriers and lock-free) against the original Splash-3. On the real machine, we run ten executions of each application and collect the average, minimum, and maximum speedup for each. These can be seen in Figure 1. The error bars indicate the minimum and maximum speedups observed, relative to the average value. Similarly, Figure 2 shows the results using gem5, but without error bars, as the simulator is deterministic.

When comparing these two, the most obvious difference can be seen with Radiosity. While in the simulator Radiosity is one of the best scaling applications, on the real system it does not scale as well. Otherwise, the rest of the applications exhibit similar behavior in both the simulator and the real system, with many applications scaling better in Splash-4 than in Splash-3, especially on the real system. In particular, Raytrace and both Ocean versions of Splash-3 only scale up to four cores, while the Splash-4 versions scale up to 32 cores.

IV. Conclusions

In this work, we focus on the performance of the synchronization operations used in the Splash-3 benchmark suite, with the goal of modernizing and improving the scalability of the applications. We transform critical sections into lock-free constructs, while maintaining the applications' behavior.

We evaluate the new version of the suite, named Splash-4, both in a state-of-the-art simulator and in a current processor. Splash-4 exercises the atomic support of the hardware and scales better than Splash-3 (from 8 and 16 cores up to 32 and 64 cores) for most of the applications under current hardware.

V. Future Work

While modifying the Splash-3 benchmarks, replacing lock-based synchronization constructs with lock-free atomic-based ones, we kept all the algorithms exactly the same. However, some of the applications (e.g. FFT, LU) can be further modified to remove barriers and replace them with signal and wait operations instead. Other algorithmic improvements can also be made to further improve scalability. For example, in Barnes, many of the critical sections could not be replaced with lock-free equivalents without modifying the algorithm.

Acknowledgments

This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 819134).

References

[1] N. Barrow-Williams, C. Fensch, and S. Moore, "A communication characterisation of splash-2 and parsec," in *2009 IEEE International Symposium on Workload Characterization (IISWC)*, 2009, pp. 86–97.

[2] C. Bienia, S. Kumar, and Kai Li, "Parsec vs. splash-2: A quantitative comparison of two multithreaded benchmark suites on chip-multiprocessors," in *2008 IEEE International Symposium on Workload Characterization*, 2008, pp. 47–56.

[3] C. Bienia, "Benchmarking modern multiprocessors," Ph.D. dissertation, Princeton University, Jan. 2011.

[4] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, May 2011.

[5] H. Gao and W. Hesselink, "A general lock-free algorithm using compare-and-swap," *Information and Computation*, vol. 205, no. 2, pp. 225–241, 2007.

[6] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," in *2005 Conf. on Programming Language Design and Implementation (PLDI)*, Jun. 2005, pp. 190–200.

[7] J. M. Mellor-Crummey and M. L. Scott, "Algorithms for scalable synchronization on shared-memory multiprocessors," *ACM Trans. Comput. Syst.*, vol. 9, no. 1, pp. 21–65, Feb. 1991.

[8] C. Sakalis, C. Leonardsson, S. Kaxiras, and A. Ros, "Splash-3: A properly synchronized benchmark suite for contemporary research," in *Int'l Symp. on Performance Analysis of Systems and Software (ISPASS)*, Apr. 2016, pp. 101–111.

[9] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," in *22nd Int'l Symp. on Computer Architecture (ISCA)*, Jun. 1995, pp. 24–36.