Renaissance: A Modern Benchmark Suite for Parallel Applications on the JVM

Aleksandar Prokopec
Oracle Labs
Switzerland
aleksandar.prokopec@oracle.com

Andrea Rosà
Università della Svizzera italiana
Switzerland
andrea.rosa@usi.ch

David Leopoldseder
Johannes Kepler Universität Linz
Austria
david.leopoldseder@jku.at

Gilles Duboscq
Oracle Labs
Switzerland
gilles.m.duboscq@oracle.com

Petr Tůma
Charles University
Czech Republic
petr.tuma@d3s.mff.cuni.cz

Martin Studener
Johannes Kepler Universität Linz
Austria
martinstudener@gmail.com

Lubomír Bulej
Charles University
Czech Republic
bulej@d3s.mff.cuni.cz

Yudi Zheng
Oracle Labs
Switzerland
yudi.zheng@oracle.com

Alex Villazón
Universidad Privada Boliviana
Bolivia
avillazon@upb.edu

Doug Simon
Oracle Labs
Switzerland
doug.simon@oracle.com

Thomas Würthinger
Oracle Labs
Switzerland
thomas.wuerthinger@oracle.com

Walter Binder
Università della Svizzera italiana
Switzerland
walter.binder@usi.ch

CCS Concepts • **General and reference → Empirical studies**; **Evaluation**; *Metrics*; • **Software and its engineering → Just-in-time compilers**; **Dynamic compilers**; **Runtime environments**; *Object oriented languages*; *Functional languages*; • **Computing methodologies →** *Parallel programming languages*; *Concurrent programming languages.*

Keywords benchmarks, JIT compilation, parallelism, concurrency, JVM, object-oriented programming benchmarks, functional programming benchmarks, Big Data benchmarks

ACM Reference Format:
Aleksandar Prokopec, Andrea Rosà, David Leopoldseder, Gilles Duboscq, Petr Tůma, Martin Studener, Lubomír Bulej, Yudi Zheng, Alex Villazón, Doug Simon, Thomas Würthinger, and Walter Binder. 2019. Renaissance: A Modern Benchmark Suite for Parallel Applications on the JVM. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Extended Abstract

To demonstrate that a compiler optimization, a memory management algorithm, or a synchronization technique is useful, researchers need benchmarks that demonstrate the desired behavior and, at the same time, capture representative aspects of real-world applications. During the last decade, multiple new programming paradigms have appeared on the Java Virtual Machine (JVM), including functional programming, big-data processing, parallel and concurrent programming, message passing, stream processing, and machine learning. The JVM has evolved as a platform, too. New features—such as method handles, variable handles, the `invokedynamic` instruction, lambdas, atomic and relaxed memory operations—present new challenges for just-in-time (JIT) compilers and runtime environments.

Workloads exercising the above features potentially present new optimization opportunities for compilers and virtual machines. Unfortunately, existing benchmark suites (such as DaCapo [2], ScalaBench [5], or SPECjvm2008 [1]) do not capture these new features, because they were made in a time when such workloads did not exist. Moreover, such suites do not specifically focus on concurrency and parallelism.

To bridge this gap, we propose Renaissance [4], a new representative set of benchmarks that covers modern JVM concurrency and parallelism paradigms. Renaissance consists of 25 benchmarks (summarized in Table 1) representative of common modern patterns (including, but not limited to, big data, machine learning, and functional programming) relying on multiple existing state-of-the-art Java and Scala

Table 1. Summary of benchmarks included in Renaissance (version 0.10.0, latest at time of writing).

Benchmark	Description	Focus
akka-uct	Unbalanced Cobwebbed Tree computation using Akka.	actors, message-passing
als	Alternating Least Squares algorithm using Spark.	data-parallel, compute-bound
chi-square	Computes a Chi-Square Test in parallel using Spark ML.	data-parallel, machine learning
db-shootout	Parallel shootout test on Java in-memory databases.	query-processing, data structures
dec-tree	Classification decision tree algorithm using Spark ML.	data-parallel, machine learning
dotty	Compiles a Scala codebase using the Dotty compiler for Scala.	data-structures, synchronization
finagle-chirper	Simulates a microblogging service using Twitter Finagle.	network stack, futures, atomics
finagle-http	Simulates a high server load with Twitter Finagle and Netty.	network stack, message-passing
fj-kmeans	K-means algorithm using the Fork/Join framework.	task-parallel, concurrent data structures
future-genetic	Genetic algorithm function optimization using Jenetics.	task-parallel, contention
gauss-mix	Computes a Gaussian mixture model using expectation-maximization.	machine learning
log-regression	Performs the logistic regression algorithm on a large dataset.	data-parallel, machine learning
mnemonics	Solves the phone mnemonics problem using JDK streams.	streaming
movie-lens	Recommender for the MovieLens dataset using Spark ML.	data-parallel, compute-bound
naive-bayes	Multinomial Naive Bayes algorithm using Spark ML.	data-parallel, machine learning
neo4j-analytics	Analytical queries and transactions on the Neo4J database.	query processing, transactions
page-rank	PageRank using the Apache Spark framework.	data-parallel, atomics
par-mnemonics	Solves the phone mnemonics problem using parallel JDK streams.	parallel streaming
philosophers	Dining philosophers using the ScalaSTM framework.	STM, atomics, guarded blocks
reactors	A set of message-passing workloads encoded in the Reactors framework.	actors, message-passing, critical sections
rx-scrabble	Solves the Scrabble puzzle using the RxJava framework.	streaming
scala-kmeans	Runs the K-Means algorithm using Scala collections.	machine learning
scala-stm-bench7	STMBench7 workload using the ScalaSTM framework.	STM, atomics
scrabble	Solves the Scrabble puzzle using Java 8 Streams.	data-parallel, memory-bound
streams-mnemonics	Computes phone mnemonics using Java 8 Streams.	data-parallel, memory-bound

frameworks. The suite can be useful to optimize just-in-time (JIT) compilers, interpreters, garbage collections, as well as tools such as profilers, debuggers, or static analyzers.

To obtain these benchmarks, we gathered more than 100 candidate workloads, both manually and by scanning an online corpus of GitHub projects. We then defined and collected a set of metrics able to capture the use of concurrency-related features as well as the use of code patterns commonly associated with abstractions of object-oriented programs. We used these metrics to detect potentially interesting workloads, and to ensure that the selection is sufficiently diverse. Thanks to PCA analysis, we revealed that the set of benchmarks selected for inclusion covers the metric space differently than the existing benchmark suites.

To confirm that the selected benchmarks are useful, we analyzed them for performance-critical patterns. We demonstrated that the proposed benchmarks reveal new opportunities for JIT compilers by implementing four new optimizations in the Graal JIT compiler [3]. These optimizations have a considerably smaller impact on existing suites, indicating that Renaissance helped in identifying new compiler optimizations. We also identified three existing optimizations whose performance impact is prominent. Furthermore, by comparing two production-quality JIT compilers, Graal and HotSpot C2, we determined that performance varies much more on Renaissance than on other benchmark suites.

We also compared the complexity of the Renaissance workloads with those of other suites, by evaluating six Chidamber and Kemerer metrics, and by inspecting the compiled code size and the hot method count of all the benchmarks. Our results show that the proposed benchmark suite is as complex as DaCapo and ScalaBench, and much more complex than SPECjvm2008.

Renaissance is intended to be an open-source, collaborative project, in which the community can propose and improve benchmark workloads. Renaissance is publicly available at https://renaissance.dev. More information on the suite and on the described analyses can be found in our previous publication [4].

References

[1] 2008. SPECjvm2008. https://www.spec.org/jvm2008/.

[2] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. 2006. The DaCapo Benchmarks: Java Benchmarking Development and Analysis. *SIGPLAN Not.* 41, 10 (Oct. 2006), 169–190.

[3] Gilles Duboscq, Thomas Würthinger, Lukas Stadler, Christian Wimmer, Doug Simon, and Hanspeter Mössenböck. 2013. An Intermediate Representation for Speculative Optimizations in a Dynamic Compiler. In *VMIL*. 1–10.

[4] Aleksandar Prokopec, Andrea Rosà, David Leopoldseder, Gilles Duboscq, Petr Tůma, Martin Studener, Lubomír Bulej, Yudi Zheng, Alex Villazón, Doug Simon, Thomas Würthinger, and Walter Binder. 2019. Renaissance: Benchmarking Suite for Parallel Applications on the JVM. In *PLDI*. 31–47.

[5] Andreas Sewe, Mira Mezini, Aibek Sarimbekov, and Walter Binder. 2011. Da Capo Con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine. In *OOPSLA*. 657–676.