Renaissance Suite

A Modern Benchmark Suite for Parallel Applications on the JVM

Aleksandar Prokopec [1] Andrea Rosà [2] David Leopoldseder [3] Gilles Duboscq [1]
Petr Tuma [4] Martin Studener [3] Lubomír Bulej [4] Yudi Zheng [1]
Alex Villazón [5] Doug Simon [1] Thomas Würthinger [1] Walter Binder [2]

[1] Oracle Labs, Switzerland [2] Università della Svizzera italiana, Switzerland [3] Johannes Kepler Universität Linz, Austria
[4] Charles University, Czech Republic [5] Universidad Privada Boliviana, Bolivia

Renaissance Suite

- Renaissance is a modern, open, and diversified benchmark suite for the JVM, aimed at testing JIT compilers, garbage collectors, profilers, analyzers and other tools.
- Renaissance contains many modern workloads, comprising various popular systems, frameworks and applications made for the JVM.
- Renaissance benchmarks exercise several programming paradigms, including concurrent, parallel, functional and object-oriented programming.

Motivation

- Many modern programming abstractions (e.g., Java Lambdas, Streams, Futures) are not represented in existing benchmark suites.
- State-of-the-art JIT compilers show similar performance on existing suites. We need new workloads to demonstrate performance gains when exploring compiler optimizations.
- Optimizations in existing JIT compilers rarely focus on concurrency-related primitives. We need new workloads including concurrency- and parallelism-related constructs.

Selection Criteria

- **Use of modern concurrency primitives**: data-parallelism, task-parallelism, streaming and pipelined parallelism, message-based concurrency, software transactional memory, lock-based and lock-free concurrent data structures, in-memory databases, asynchronous programming, network communication.
- **Realistic workloads** using popular frameworks: Java Streams, Apache Spark, Java Reactive Extensions, Java Fork/Join framework, ScalaSTM, Twitter Finagle.
- **Workload diversity**. Benchmarks should exercise different concurrency-related features (e.g., atomic instructions, Java synchronized blocks, thread-park operations, guarded blocks). At the same time, benchmarks should make use of abstractions commonly associated with object-oriented programs, i.e., frequent object allocation and virtual dispatch.
- **Deterministic execution** (as much as possible).
- **Open-source availability** to enable the inspection of the workloads by the community, source-code level analysis, and the evaluation of the actionability of profiler results.
- **Avoid known pitfalls** of other suites, such as lack of benchmark source code, use of timeouts, resource leaks, data races, deadlocks.

Benchmark List

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 structs.
dec-tree	Classification decision tree algorithm using Spark ML.	data-parallel, machine learning
dotty	Compiles a Scala codebase using the Dotty compiler.	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, conc. data structs.
future-genetic	Genetic algorithm function optimization using Jenetics.	task-parallel, contention
gauss-mix	Computes a Gaussian mixture model.	machine learning
log-regression	Performs logistic regression 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 streams.	parallel streaming
philosophers	Dining philosophers using the ScalaSTM framework.	STM, atomics, guarded blocks
reactors	A set of message-passing workloads encoded in Reactors.	actors, msg-passing, critical sect.
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

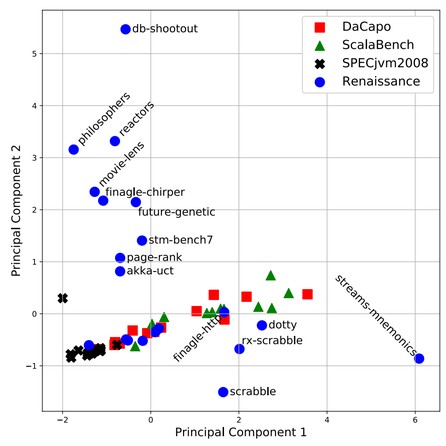
Acknowledgments: This work has been supported by Oracle (ERO project 1332).

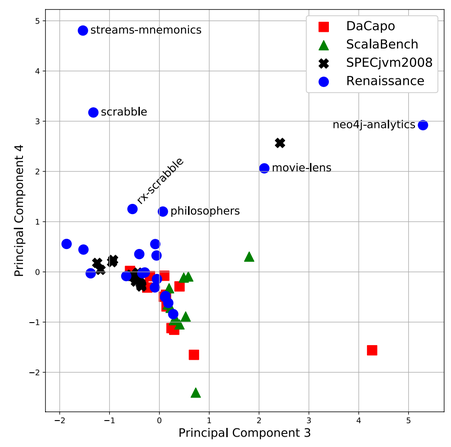
Evaluation

Diversity

- Renaissance represents concurrency primitives better than existing suites, is comparable to DaCapo and ScalaBench in terms of object-allocation rates and dynamic dispatch, and exercises invokedynamic more often.

Principal Component Analysis



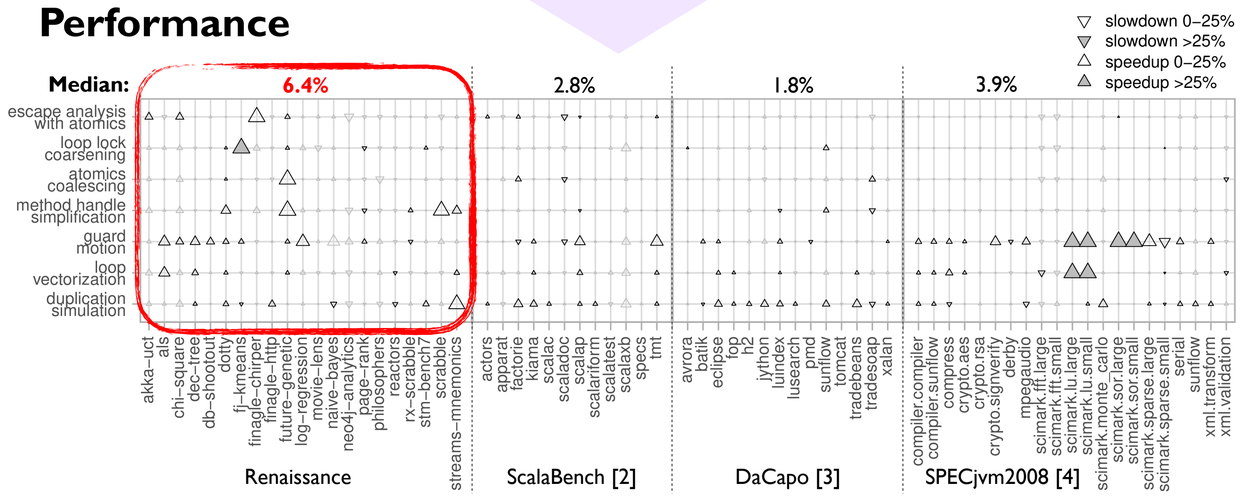


PC1 Metric	PC1 Loading	PC2 Metric	PC2 Loading	PC3 Metric	PC3 Loading	PC4 Metric	PC4 Loading
object	**+0.50**	**atomic**	**+0.67**	**cachemiss**	**+0.58**	**idynamic**	**+0.56**
cpu	**-0.49**	**park**	**+0.65**	**notify**	**+0.50**	array	+0.42
method	**+0.44**	method	+0.20	**wait**	**+0.41**	notify	+0.42
array	**+0.40**	notify	+0.18	cpu	-0.28	method	-0.35
idynamic	+0.27	idynamic	-0.17	synch	-0.25	cachemiss	+0.28
synch	-0.17	cpu	-0.16	park	-0.20	cpu	+0.22
notify	-0.13	cachemiss	-0.08	idynamic	-0.18	atomic	+0.18
atomic	-0.13	object	+0.05	array	-0.13	wait	-0.15
cachemiss	-0.07	array	-0.03	method	+0.10	object	+0.13
park	-0.06	synch	-0.02	object	-0.04	synch	+0.11
wait	-0.02	wait	-0.00	atomic	-0.03	park	+0.05

Compiler Optimizations

- 4 new compiler optimizations (escape analysis with atomic operations, loop-wide lock coarsening, atomic-operation coalescing, method-handle simplification) and 3 existing optimizations (speculative guard movement, loop vectorization, dominance-based duplication simulation) implemented in the Graal compiler [1] have a significant impact on Renaissance benchmarks, while other suites benefit less from the optimizations.

Performance



Additional Information

- Renaissance can be downloaded at:

  `https://renaissance.dev/`

- Renaissance is completely open source. Check out our GitHub repository!
- Renaissance is part of an ongoing, collaborative project in which the community can propose and improve benchmark workloads. Your contribution is welcome!
- More detailed information and analyses can be found in our **PLDI'19** paper [5].

[1] G. Duboscq et al., An Intermediate Representation for Speculative Optimizations in a Dynamic Compiler. VMIL 2013.
[2] S. Blackburn et al., The DaCapo Benchmarks: Java Benchmarking Development and Analysis. SIGPLAN Not. 41, 10 (Oct. 2006).
[3] A. Sewe et al., Da Capo Con Scala: Design and Analysis of a Scala Benchmark Suite for the Java Virtual Machine. OOPSLA 2011.
[4] SPECjvm2008. https://www.spec.org/jvm2008/
[5] A. Prokopec et al., Renaissance: Benchmarking Suite for Parallel Applications on the JVM. PLDI 2019.