Leveraging RISC-V Vectorization: Accelerating Java Programs with TornadoVM and OCK

Juan Fumero[1], Athanasios Stratikopoulos[1], Colin Davidson[2], Harald van Dijk[2], Uwe Dolinsky[2], Michail Papadimitriou[1], Maria Xekalaki[1], and Christos Kotselidis[1]
[1]Department of Computer Science, The University of Manchester
[2]Codeplay Software Ltd, UK

MOTIVATION

Java and JVM prioritizes stability and backward compatibility. However, this can slow adoption of new hardware features.

So, how can we accelerate modern Java workloads on RISC-V Accelerators?

TOOLCHAIN

TornadoVM: A Java parallel programming framework that offloads data parallel workloads onto heterogenous hardware to increase performance of Java programs. It contains a JIT compiler and a runtime system to run as transparent as possible on GPUs, FPGAs and CPUs.

oneAPI Construction Kit (OCK): A programming framework for the implementation of Open Standards for new hardware accelerators. Runs on RISC-V and it provides an auto-vectorizer.

MORE INFORMATION

Scan & read our paper

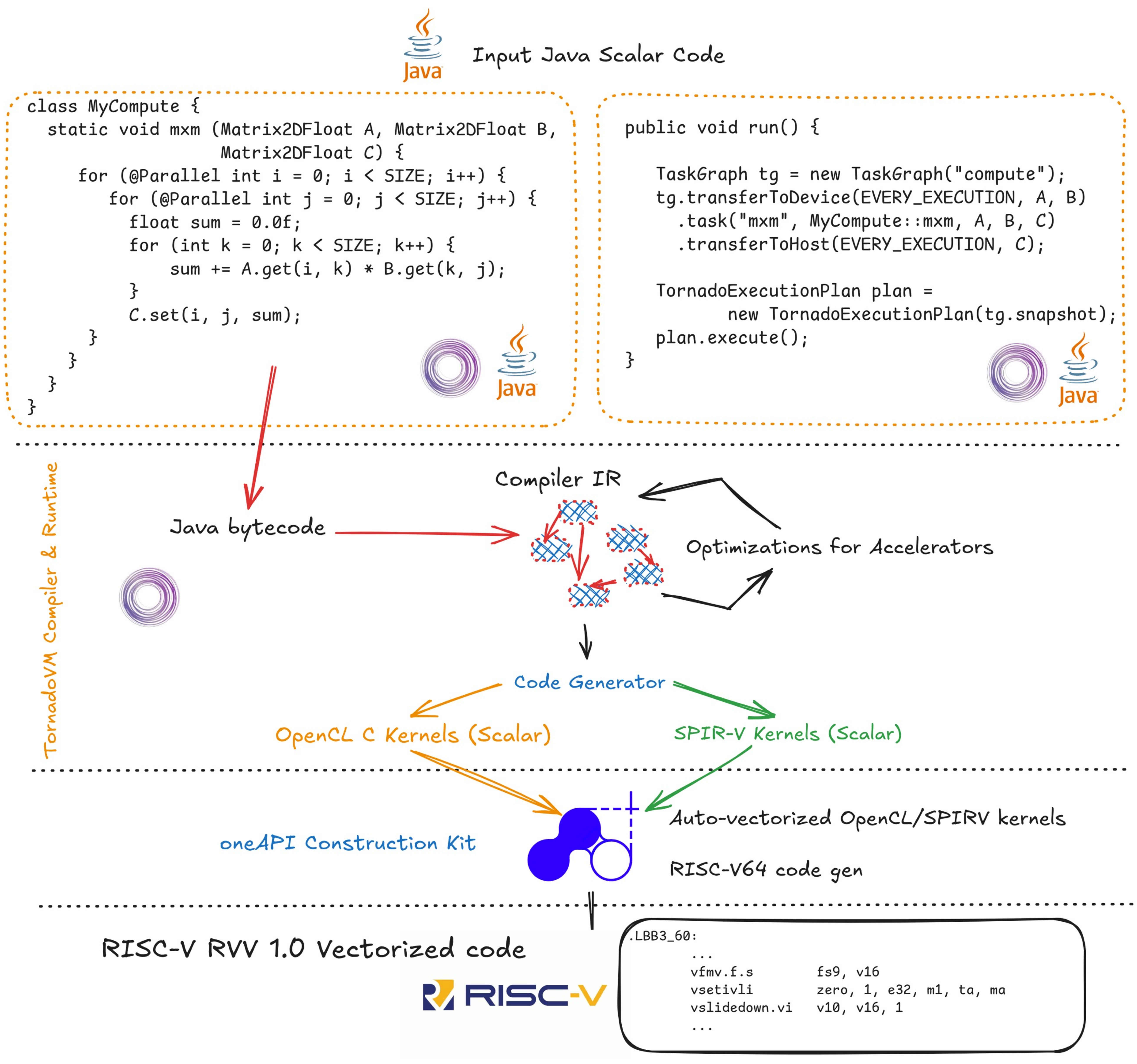
Follow our work in AERO

REFERENCES

[1] Juan Fumero et al. "**Dynamic application reconfiguration on heterogeneous hardware**". VEE 2019. doi: 10.1145/ 3313808.3313819.

[2] Alastair Murray and Ewan Crawford. "**Compute Aorta: A toolkit for implementing heterogeneous programming models**". IWOCL '20. doi: 10.1145/3388333.3388652.

COMPILER TOOLCHAIN: FROM JAVA SCALAR TO VECTORIZED CODE


Input Java Scalar Code
Java
TornadoVM Compiler & Runtime
Java bytecode
Compiler IR
Optimizations for Accelerators
Code Generator
OpenCL C Kernels (Scalar)
SPIR-V Kernels (Scalar)
oneAPI Construction Kit
Auto-vectorized OpenCL/SPIRV kernels
RISC-V64 code gen
RISC-V RVV 1.0 Vectorized code
RISC-V

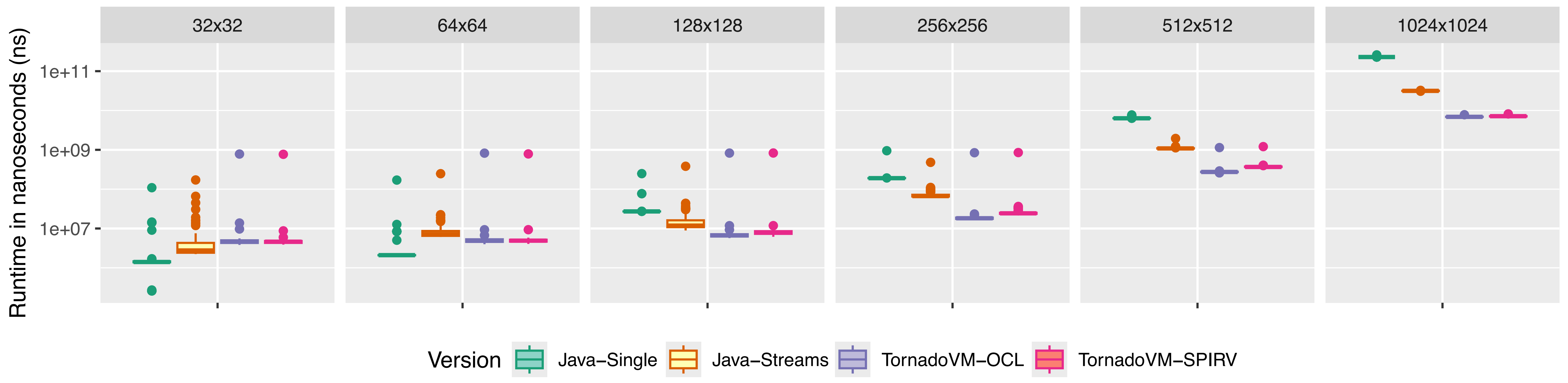
```
class MyCompute {
  static void mxm (Matrix2DFloat A, Matrix2DFloat B,
                   Matrix2DFloat C) {
     for (@Parallel int i = 0; i < SIZE; i++) {
        for (@Parallel int j = 0; j < SIZE; j++) {
          float sum = 0.0f;
          for (int k = 0; k < SIZE; k++) {
              sum += A.get(i, k) * B.get(k, j);
          }
          C.set(i, j, sum);
        }
     }
  }
}
```

```
public void run() {

   TaskGraph tg = new TaskGraph("compute");
   tg.transferToDevice(EVERY_EXECUTION, A, B)
     .task("mxm", MyCompute::mxm, A, B, C)
     .transferToHost(EVERY_EXECUTION, C);

   TornadoExecutionPlan plan =
          new TornadoExecutionPlan(tg.snapshot);
   plan.execute();
}
```

```
.LBB3_60:
        ...
        vfmv.f.s        fs9, v16
        vsetivli        zero, 1, e32, m1, ta, ma
        vslidedown.vi   v10, v16, 1
        ...
```

PERFORMANCE for MATRIX MULTIPLICATION of SIZE NxN

- TornadoVM + OCK is
  - up to 4.6x vs Java-Streams
  - up to 33x vs Java single threaded
- Java-Streams is 7.2x faster than Java single threaded



Evaluated on a BananaPI F3: Spacemit K1 Octacore Processor with 4GB of RAM

Funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the HaDEA. Neither the European Union nor the granting authority can be held responsible for them. Project number: 101092850. In addition, this work is funded by UK Research and Innovation (UKRI) under the UK government's Horizon Europe funding guarantee for grant numbers 10048318 and 10048915.