Accelerating Java Programs on RISC-V with Vector Instructions via TornadoVM and OCK

Juan Fumero

Research Fellow & Intel Innovator

TornadoVM's Software Architect

@jfumero.bsky.social

MANCHESTER
1824

The University of Manchester

J Extension Task Group

27th February 2025

TORNADOVM

Outline

1. Motivation
2. Overview of TornadoVM
3. Java Acceleration on RISC-V
   - TornadoVM and oneAPI Construction Kit (OCK)
4. OCK for RISC-V
5. Performance Numbers on RISC-V RVV 1.0 – Feb 2025
6. Conclusions & Discussions

Who am I?

Dr. Juan Fumero

Research Fellow @ University of Manchester

Architect and Developer of TornadoVM

oneAPI **Intel Innovator**

- oneAPI Lang SIG
- oneAPI Hardware SIG

juan.fumero@manchester.ac.uk ✉

@jfumero.bsky.social

Background:

PhD: Java JIT Compilers for GPUs

Oracle Labs *GraalVM/Truffle*

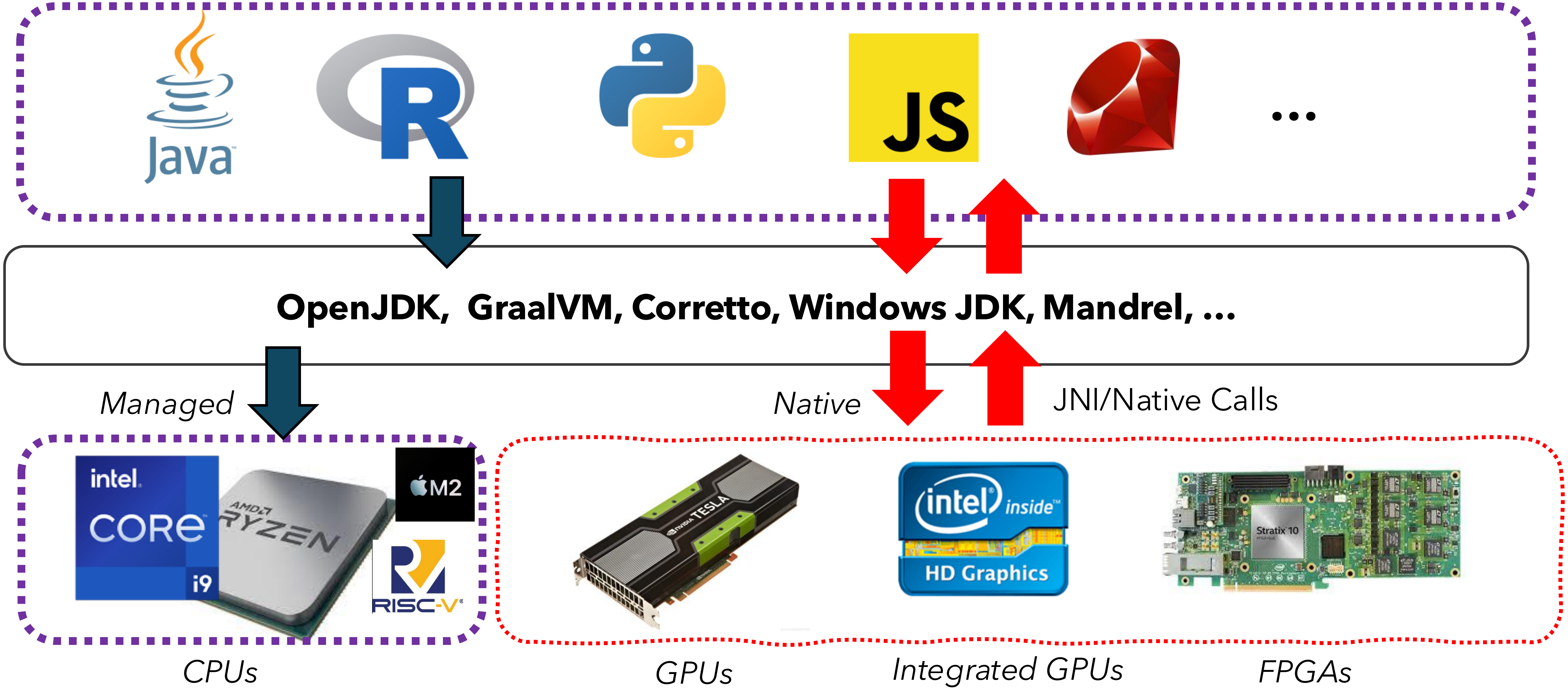
Intel CilkPlus Vectorization for Root and GeantV

TORNADOVM

www.tornadovm.org

Motivation

Enabling Acceleration for Managed Runtime Languages



OpenJDK, GraalVM, Corretto, Windows JDK, Mandrel, …

Managed

Native

JNI/Native Calls

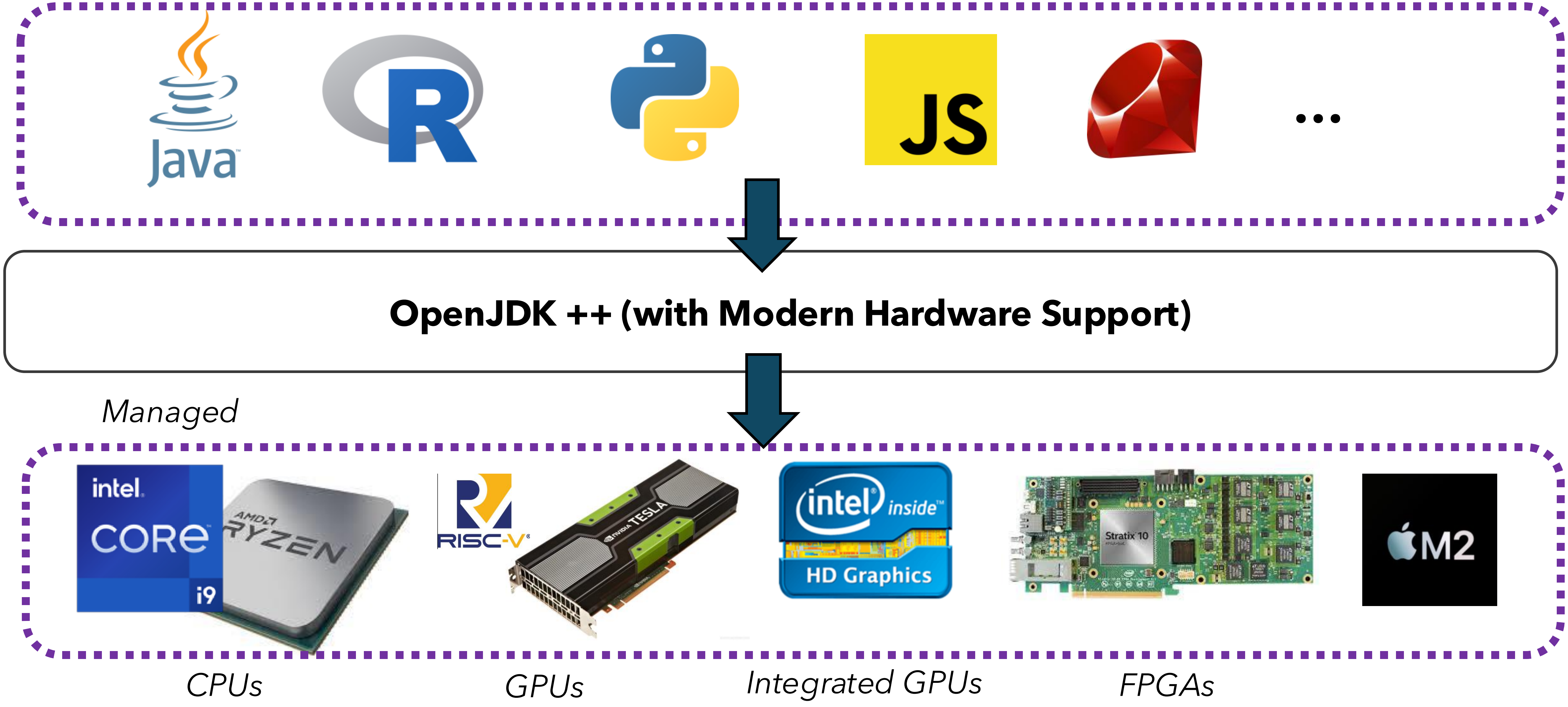
CPUs

GPUs

Integrated GPUs

FPGAs

But what if?



OpenJDK ++ (with Modern Hardware Support)

Managed

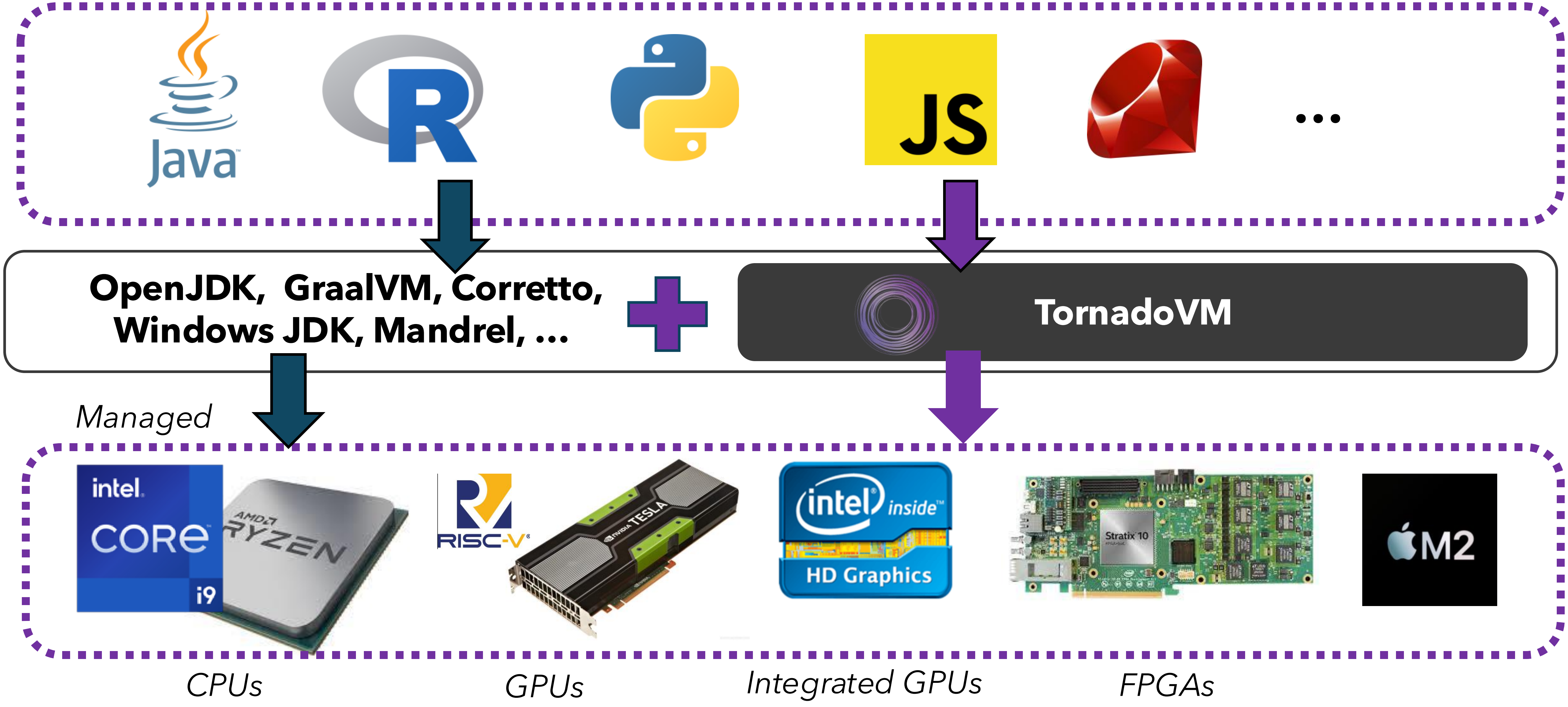
CPUs

GPUs

Integrated GPUs

FPGAs

Enabling Acceleration for Managed Runtime Languages



OpenJDK, GraalVM, Corretto, Windows JDK, Mandrel, ...

TornadoVM

Managed

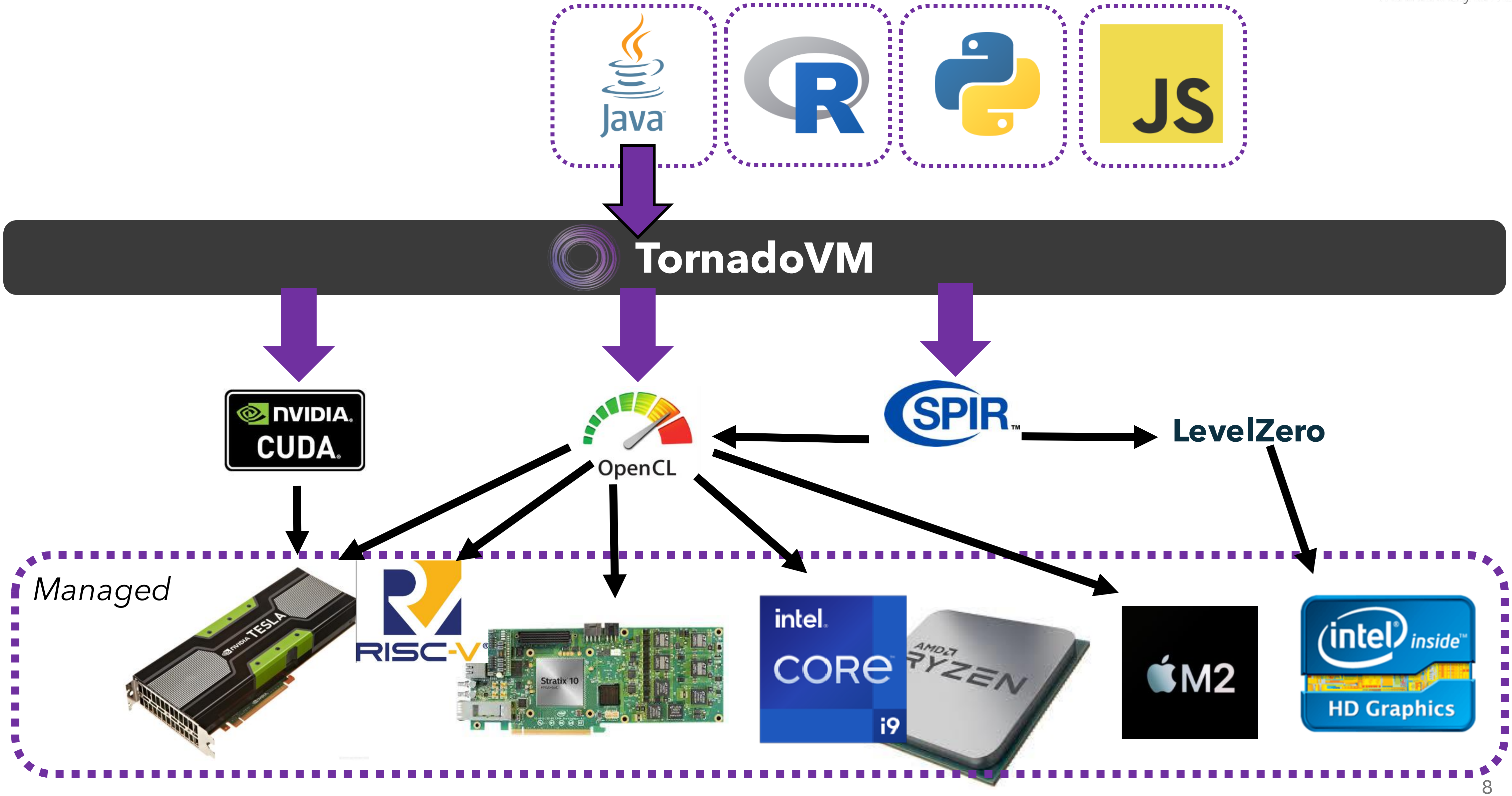
CPUs

GPUs

Integrated GPUs

FPGAs

Enabling Acceleration for Managed Runtime Languages



TornadoVM

The University of Manchester

Parallel Programming Framework

for

Accelerating

Java Data Parallel Workloads

on

Heterogeneous Hardware

Parallel Programming API

Three JIT Compilers

- *OpenCL C*
- *CUDA PTX*
- *SPIR-V*

Optimizer

- *100+ optimization phases*

Optimising Runtime System

Runs on JDK 21 & JDK 23

Dynamic Task Reconfiguration

Muti-device

Automatic Batch Processing

Automatic Data Management

Multi-backend

and it is **Open Source**:

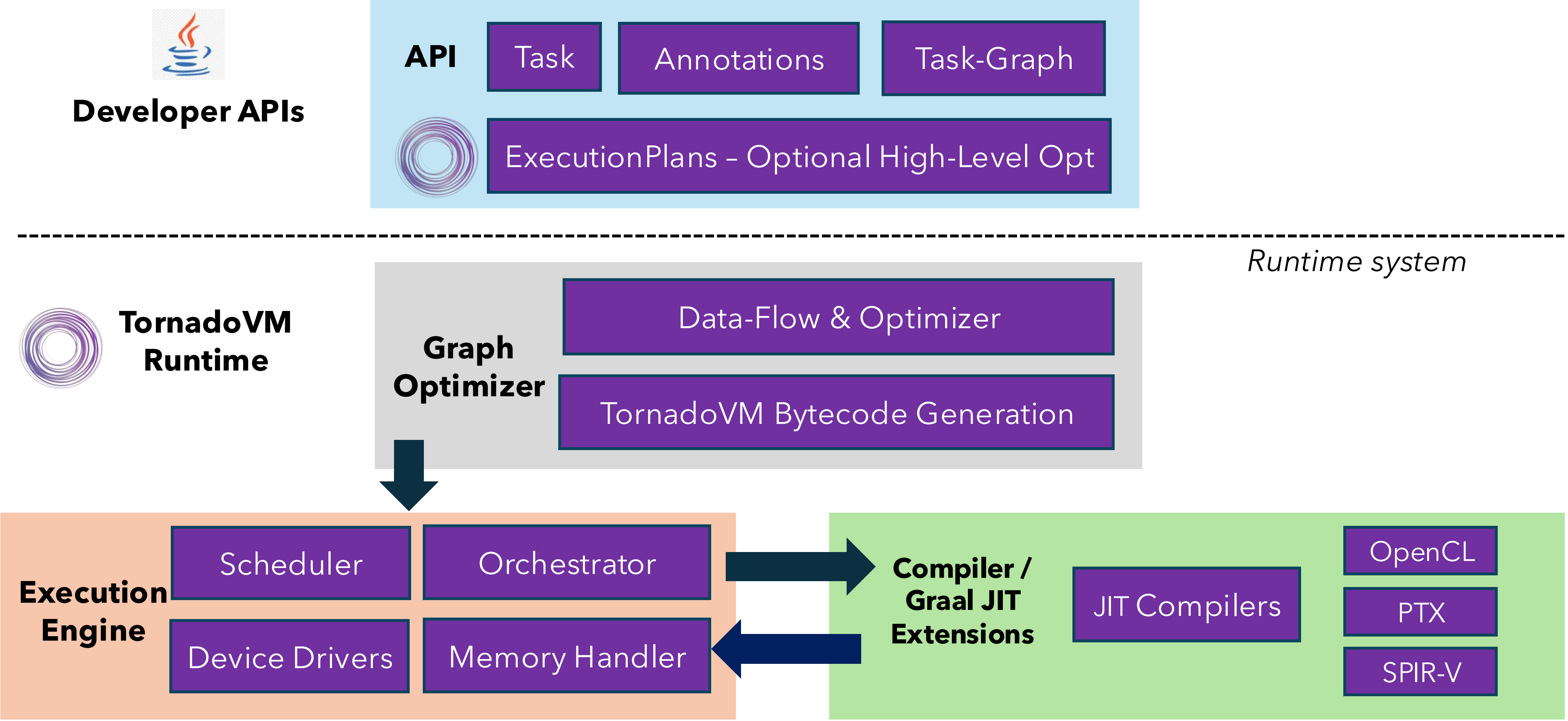
https://github.com/beehive-lab/TornadoVM

TORNADOVM

www.tornadovm.org

TornadoVM APIs

TornadoVM's Software Stack



Different components of the User APIs

a) How to represent parallelism within functions/methods?

- A.1: Java annotations for expressing parallelism (**@Parallel, @Reduce**) for Non-Experts

- A.2: Kernel API for GPU experts (use of **kernel context** object)

b) How to define which methods to accelerate?

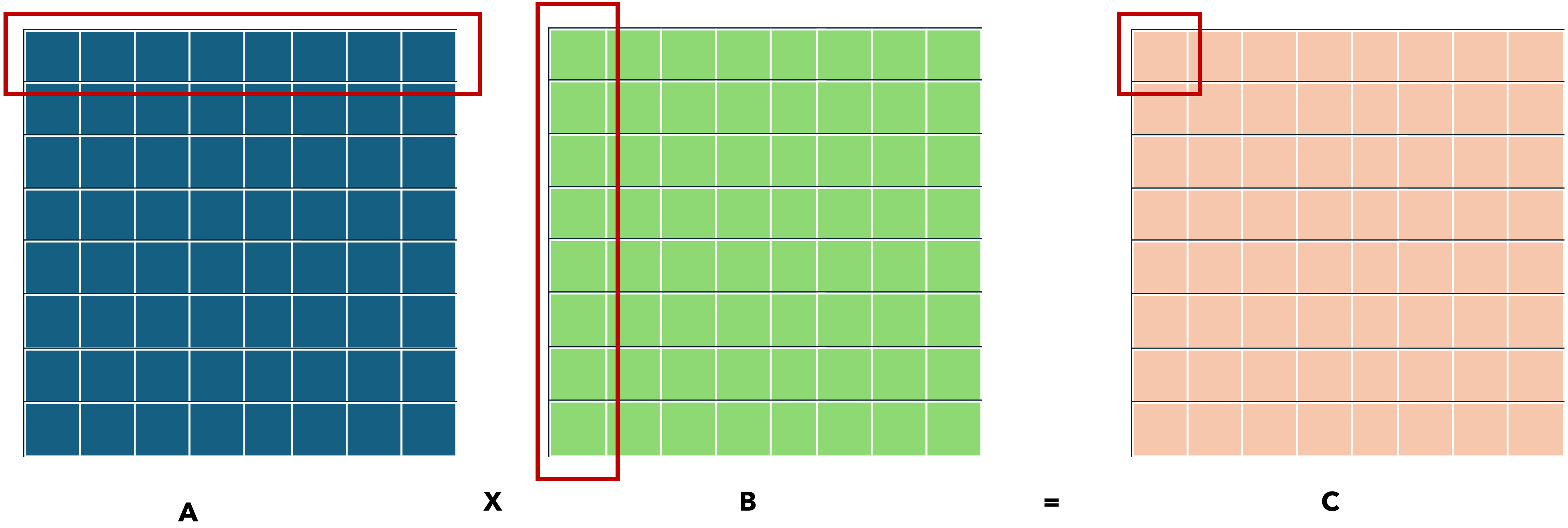
Build a **Task-Graph API** to define data In/Out and the code to be accelerated

c) How to explore different optimizations?

Execution Plan

Let's Learn the Different API Components with an Example

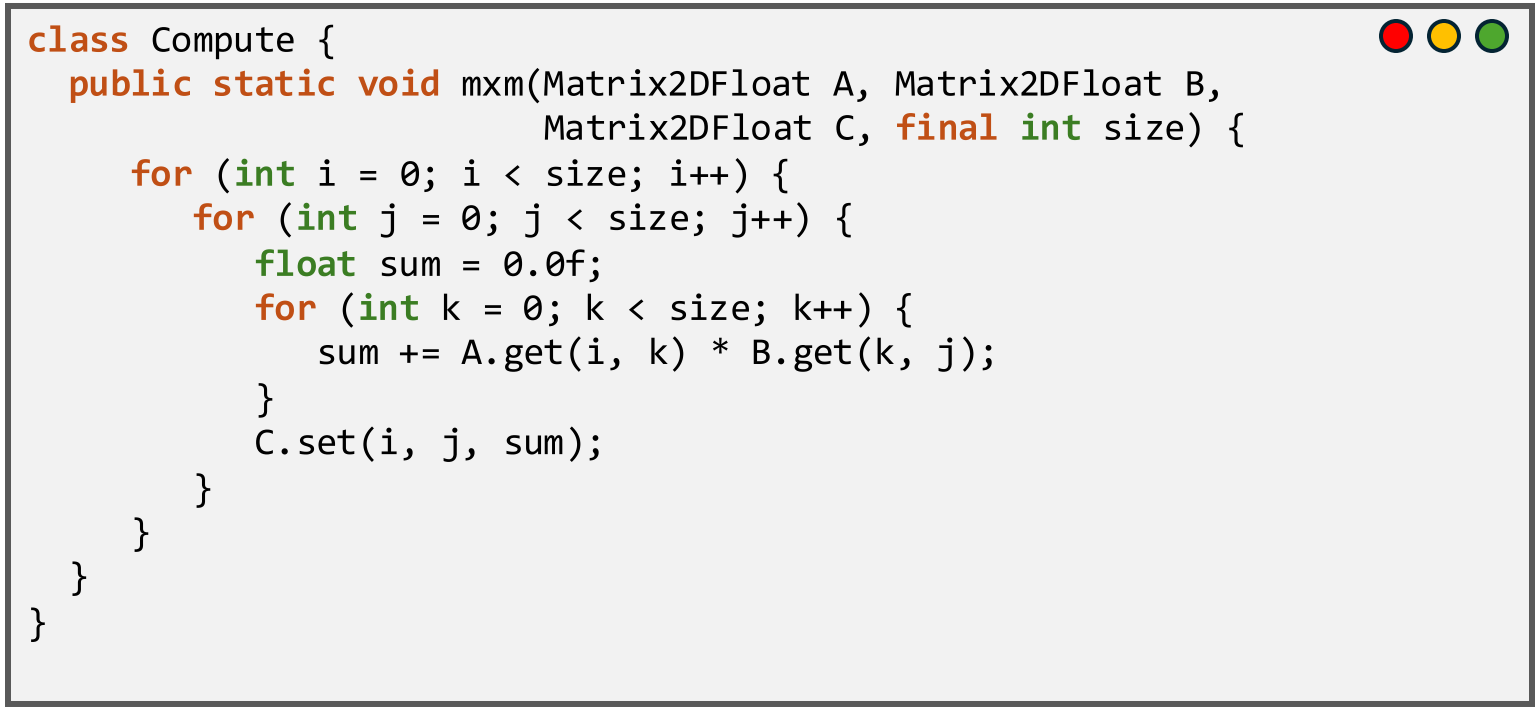
Matrix Multiplication



- Widely used on ML and AI
- Linear Algebra Kernel

https://github.com/beehive-lab/TornadoVM/blob/master/tornado-examples/src/main/java/uk/ac/manchester/tornado/examples/compute/MatrixMultiplication2D.java

Example using the TornadoVM Loop Parallel API

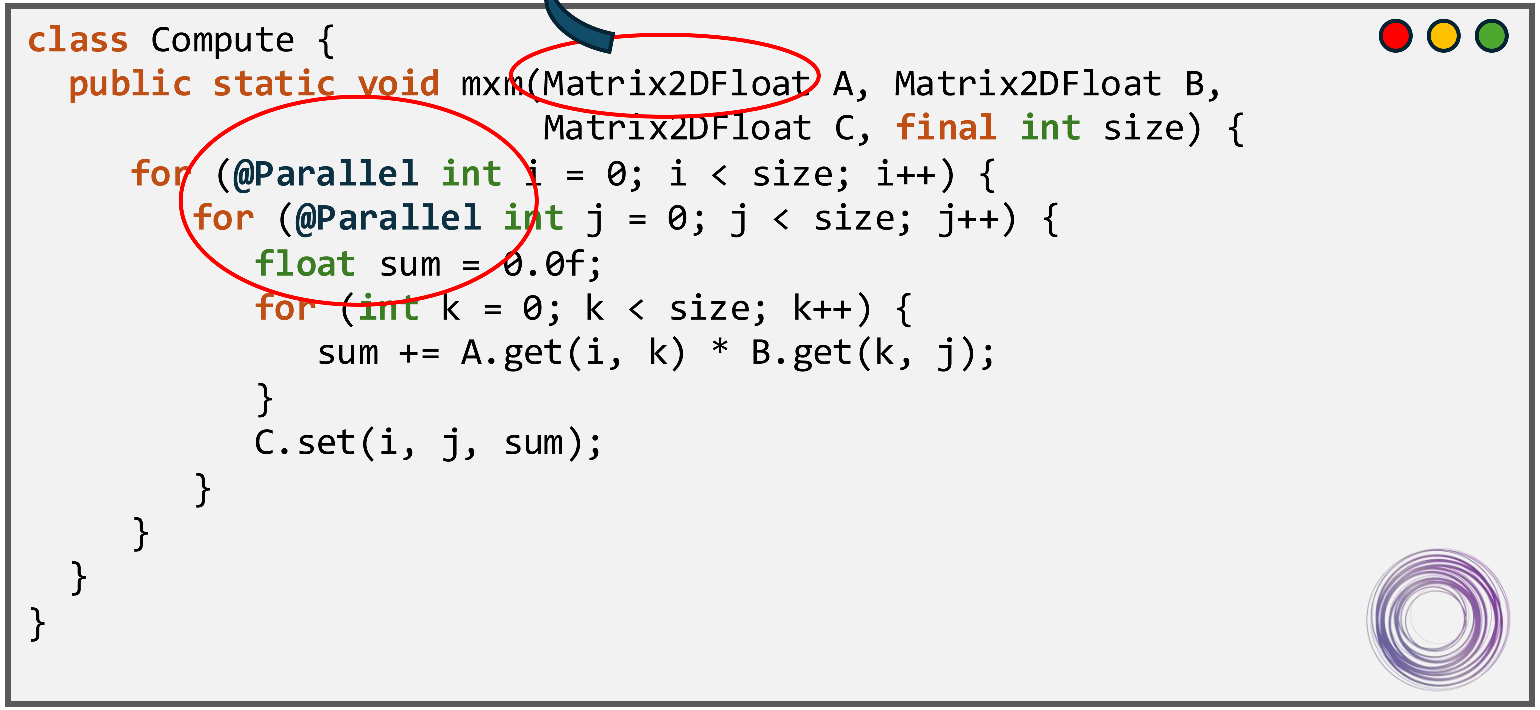


```java
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                         Matrix2DFloat C, final int size) {
     for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
           float sum = 0.0f;
           for (int k = 0; k < size; k++) {
              sum += A.get(i, k) * B.get(k, j);
           }
           C.set(i, j, sum);
        }
     }
  }
}
```

https://github.com/beehive-lab/TornadoVM/blob/master/tornado-examples/src/main/java/uk/ac/manchester/tornado/examples/compute/MatrixMultiplication2D.java

Tornado API – example using Annotations

Panama-Based Memory Segments for Off-Heap Data (*)



```
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                         Matrix2DFloat C, final int size) {
     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);
        }
     }
  }
}
```

We add the @Parallel Annotation

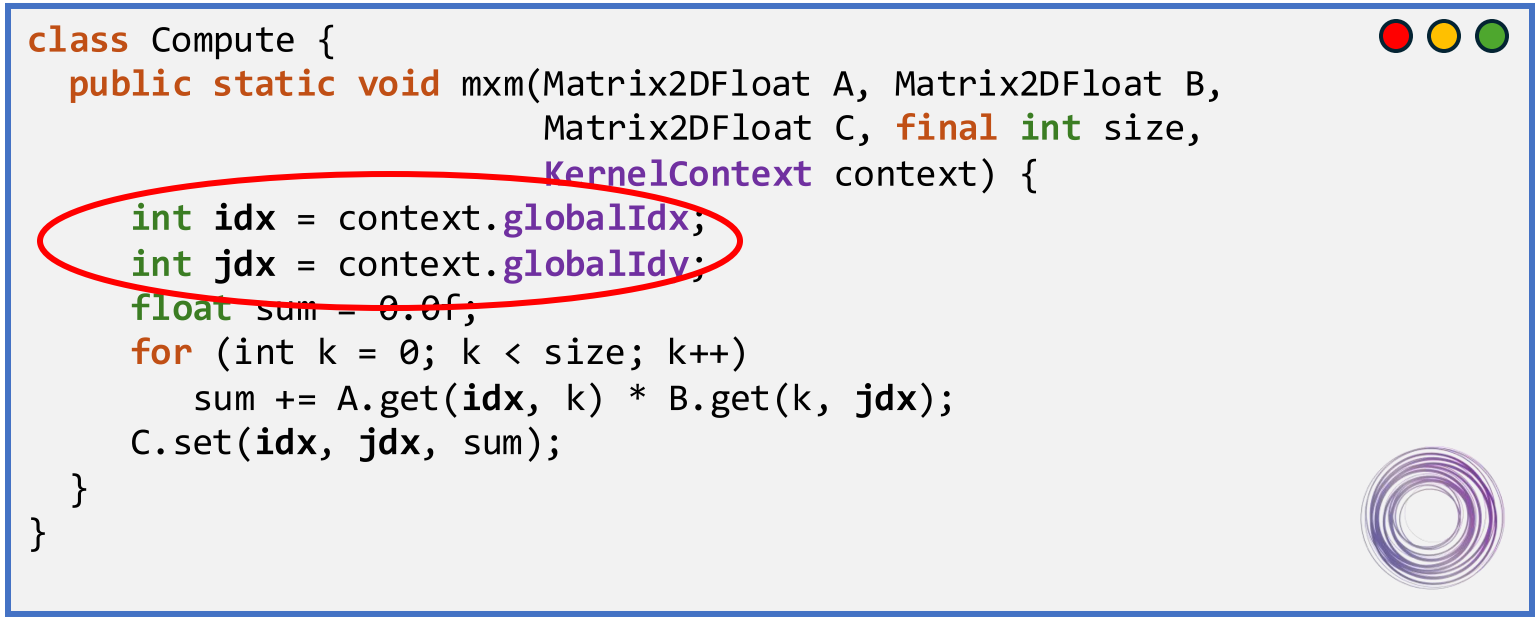
We add the parallel annotation as a hint for the compiler

We only have 2 annotations:

@Parallel

@Reduce

Tornado API – example using Kernel Context



```
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloat B,
                          Matrix2DFloat C, final int size,
                          KernelContext context) {
     int idx = context.globalIdx;
     int jdx = context.globalIdy;
     float sum = 0.0f;
     for (int k = 0; k < size; k++)
        sum += A.get(idx, k) * B.get(k, jdx);
     C.set(idx, jdx, sum);
  }
}
```

Alternative API for expert programmers. It offers more control

Kernel-Context accesses thread ids, local memory and barriers

It needs a **Grid of Threads** to be passed during the kernel launch

Tornado API – example

How to identify which methods to accelerate? --> TaskGraph

```
TaskGraph taskGraph = new TaskGraph("myComputeGraph")

   .transferToDevice(DataTransferMode.EVERY_EXECUTION , matrixA, matrixB)

   .task("parallelTask", Compute::mxm, matrixA, matrixB, matrixC, size)

   .transferToHost(DataTransferMode.EVERY_EXECUTION, matrixC);
```

Host Code

TaskGraph is a new TornadoVM object exposed to developers to define :

a) **The code to be accelerated** (which Java methods?)

b) **The data (Input/Output)** and how data should be streamed

Defining and Running Execution Plans

How to run/explore different optimizations? --> `ExecutionPlan`

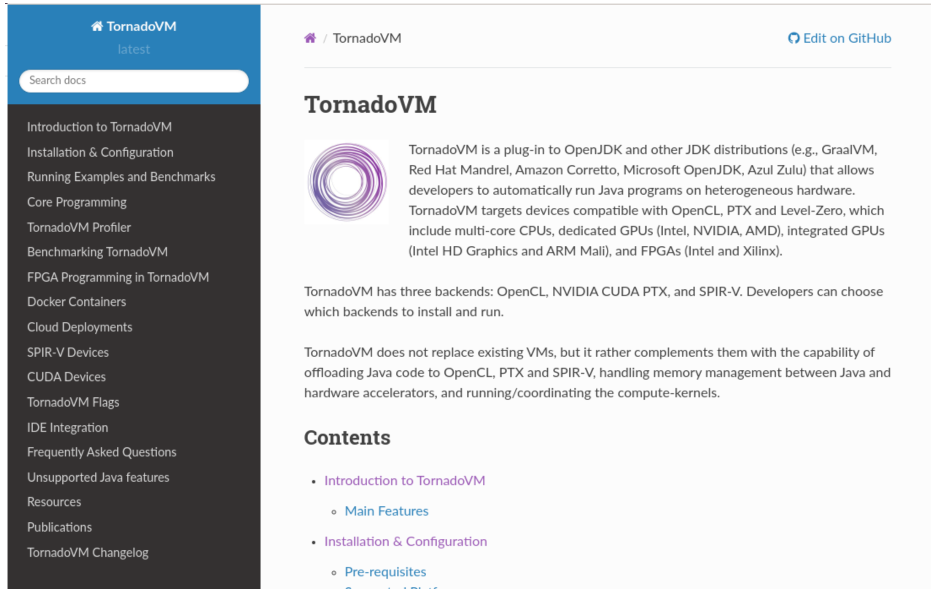
```java
TornadoExecutionPlan executionPlan = new TornadoExecutionPlan(taskGraph.snapshot());

// optional: change runtime parameters/optimize the runtime
executionPlan.withWarmUp()
             .withProfiler(...)
             .withDynamicReconfiguration(PERFORMANCE, ...);

// blocking call: the execute is where JIT compilation + execution happen
TornadoExecutionResult result = executionPlan.execute();

long elapsedKernelTime = result.getProfiler().getDeviceKernelTime(); // in ns
```

Host Code

Optional High-Level Optimization Pipelines:

- Enable/Disable Profiler
- Enable Warmup
- Enable Dynamic Reconfiguration
- Enable Batch Processing
- Enable Thread Scheduler (no need for recompilation for different grids schedulers)

To learn more about the APIs



TornadoVM

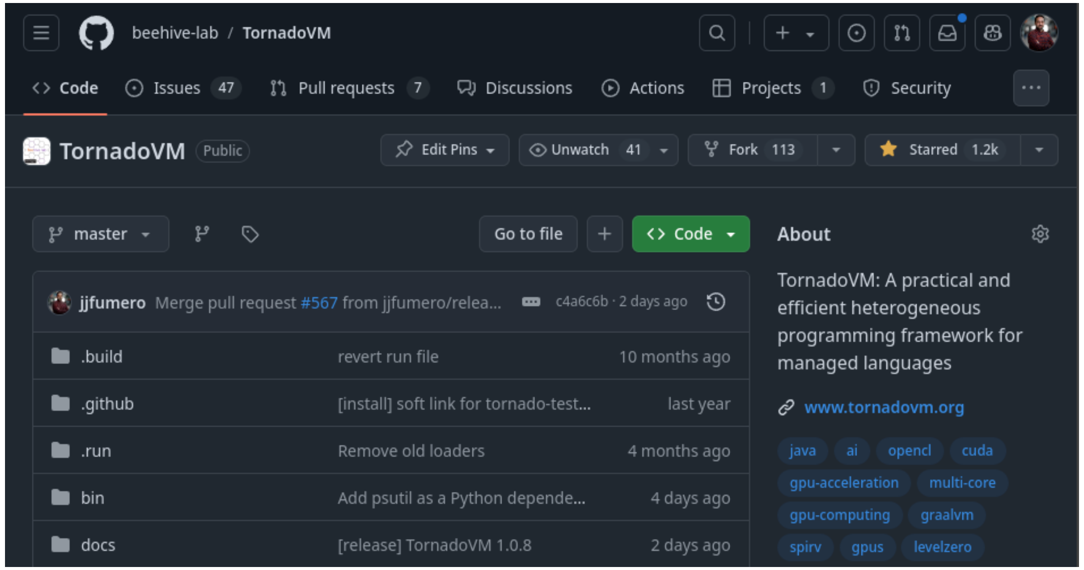
TornadoVM is a plug-in to OpenJDK and other JDK distributions (e.g., GraalVM, Red Hat Mandrel, Amazon Corretto, Microsoft OpenJDK, Azul Zulu) that allows developers to automatically run Java programs on heterogeneous hardware. TornadoVM targets devices compatible with OpenCL, PTX and Level-Zero, which include multi-core CPUs, dedicated GPUs (Intel, NVIDIA, AMD), integrated GPUs (Intel HD Graphics and ARM Mali), and FPGAs (Intel and Xilinx).

TornadoVM has three backends: OpenCL, NVIDIA CUDA PTX, and SPIR-V. Developers can choose which backends to install and run.

TornadoVM does not replace existing VMs, but it rather complements them with the capability of offloading Java code to OpenCL, PTX and SPIR-V, handling memory management between Java and hardware accelerators, and running/coordinating the compute-kernels.

Contents

- Introduction to TornadoVM
  - Main Features
- Installation & Configuration
  - Pre-requisites



jjfumero Merge pull request #567 from jjfumero/relea... c4a6c6b · 2 days ago

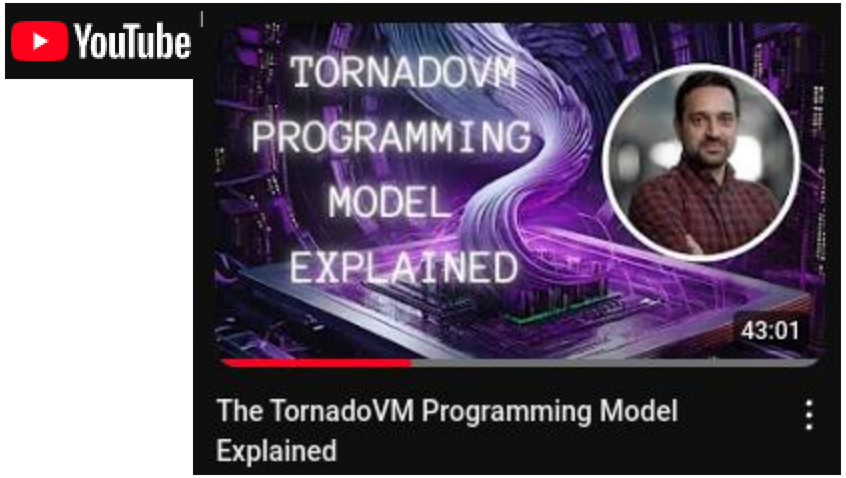
.build	revert run file	10 months ago
.github	[install] soft link for tornado-test...	last year
.run	Remove old loaders	4 months ago
bin	Add psutil as a Python depende...	4 days ago
docs	[release] TornadoVM 1.0.8	2 days ago

TornadoVM: A practical and efficient heterogeneous programming framework for managed languages

www.tornadovm.org

https://tornadovm.readthedocs.io/en/latest/

https://github.com/beehive-lab/TornadoVM



The TornadoVM Programming Model Explained

TORNADOVM

www.tornadovm.org

How TornadoVM can be used on RISC-V for RVV 1.0?

oneAPI Construction Kit (OCK)

https://github.com/uxlfoundation/oneapi-construction-kit

"Programming Framework to enable hardware platforms to access open standards (e.g., SYCL, OpenCL, etc) and it can be used to extend the oneAPI software ecosystem to custom compute architectures [1]".

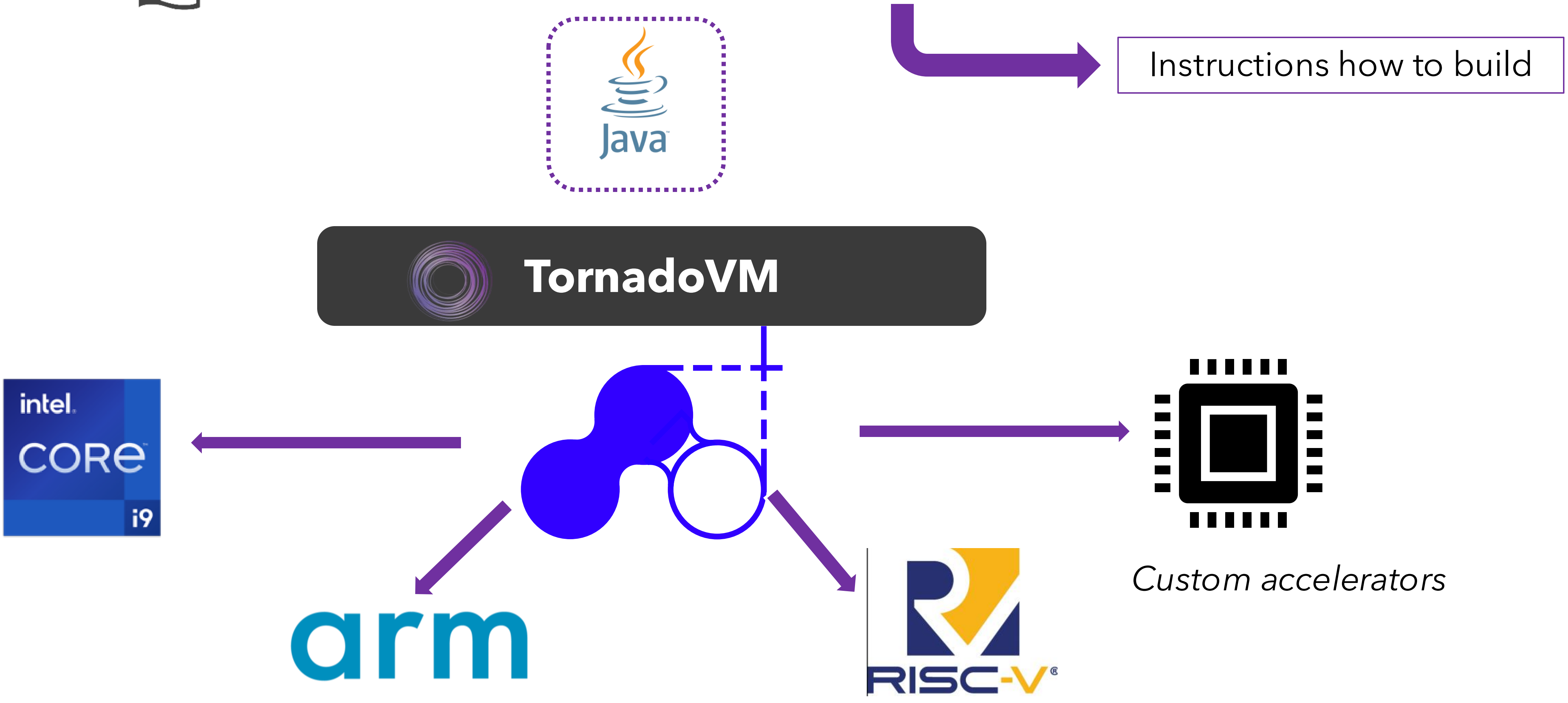
Part of the UXL Foundation: https://uxlfoundation.org/

Working with Codeplay in the AERO EU Project [2]

[1] https://developer.codeplay.com/products/oneapi/construction-kit/home/
[2] https://aero-project.eu/

Running on Multi-Vendor CPUs: ARM, RISC-V, Intel
Using the **oneAPI Construction Kit (OCK)**

https://jjfumero.github.io/posts/2024/09/10/tornadovm-ock



Instructions how to build

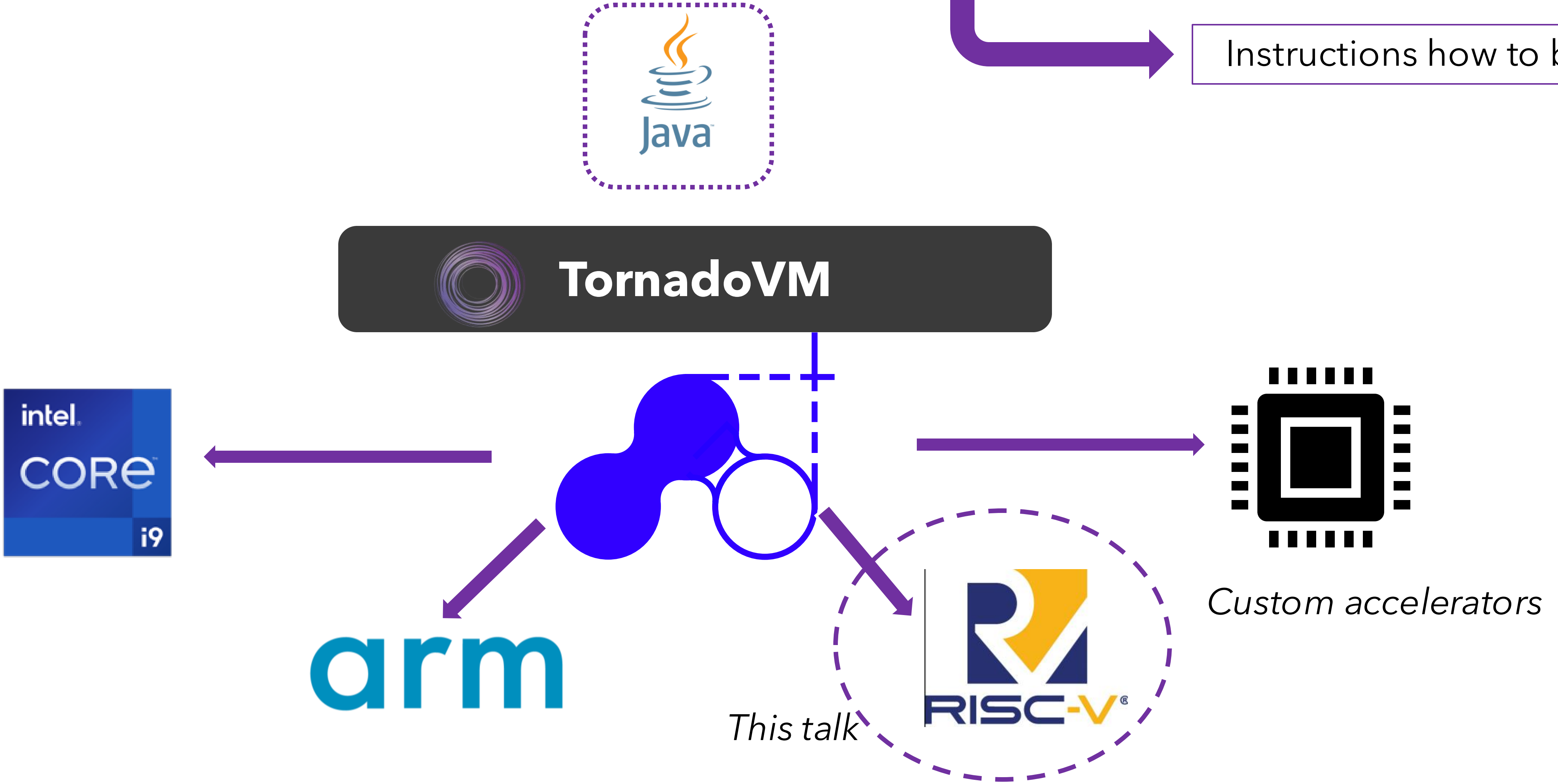
TornadoVM

Custom accelerators

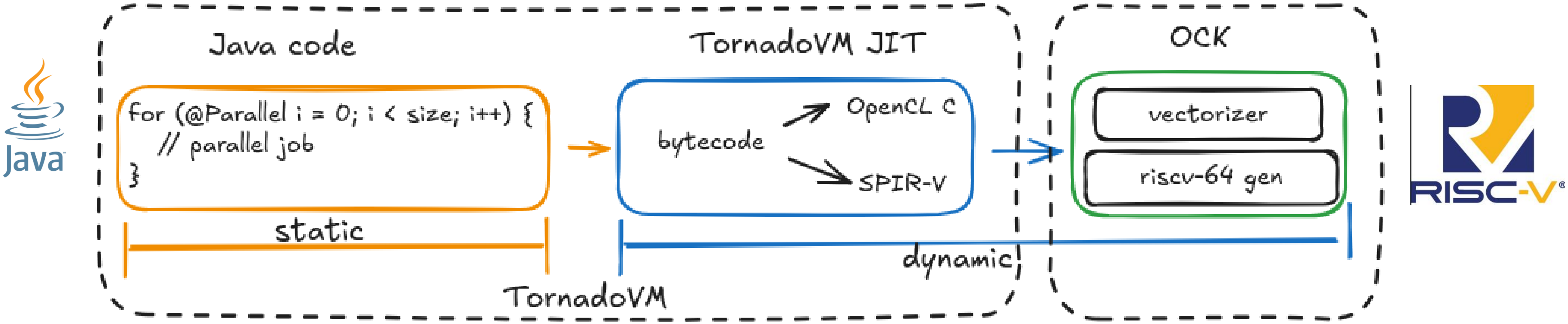
Running on Multi-Vendor CPUs: ARM, RISC-V, Intel
Using the **oneAPI Construction Kit (OCK)**

https://jjfumero.github.io/posts/2024/09/10/tornadovm-ock

Instructions how to build



JIT Compilation Process



TornadoVM contains more than 50 compilation phases
Compilation is per-device, per architecture. This is crucial to get performance portability

TORNADOVM

www.tornadovm.org

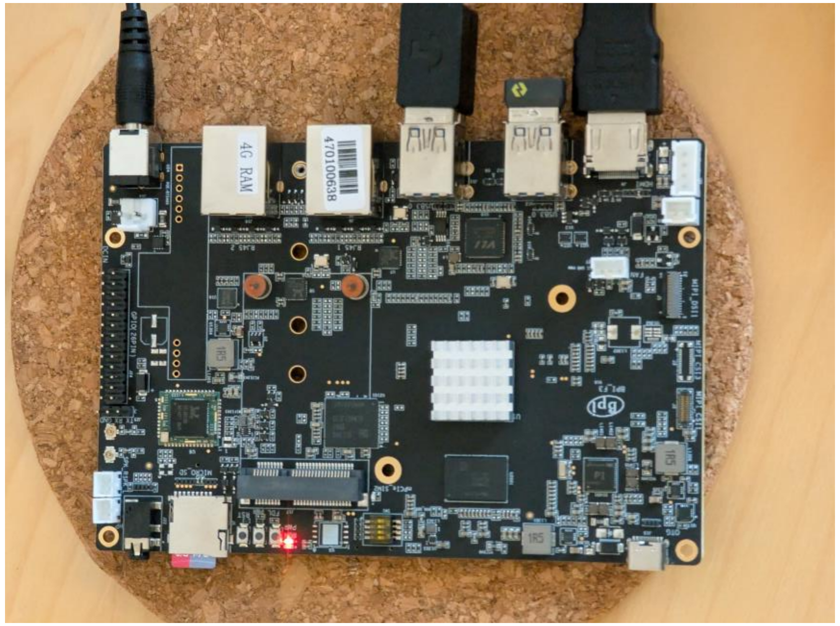
Performance

SBC used

RISC-V Spacemit K1 Processor

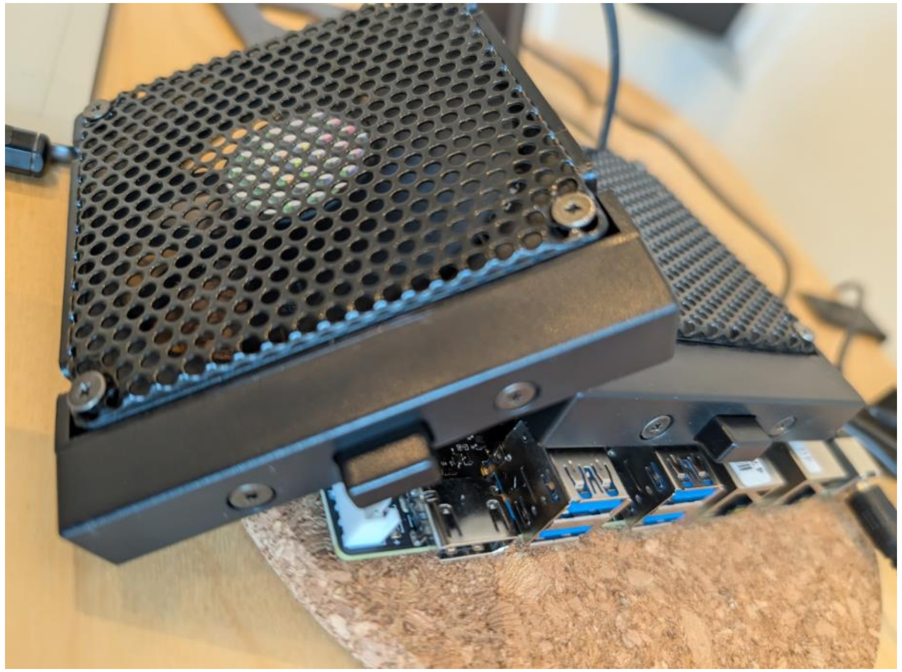
- Banana PI BPI F3
- Sipeed Lichee PI 3A

Octacore RISC-V @ 1.6GHz
RVA22 and 256-bit RVV 1.0
1MB shared L2 Cache
Max 16GB of RAM

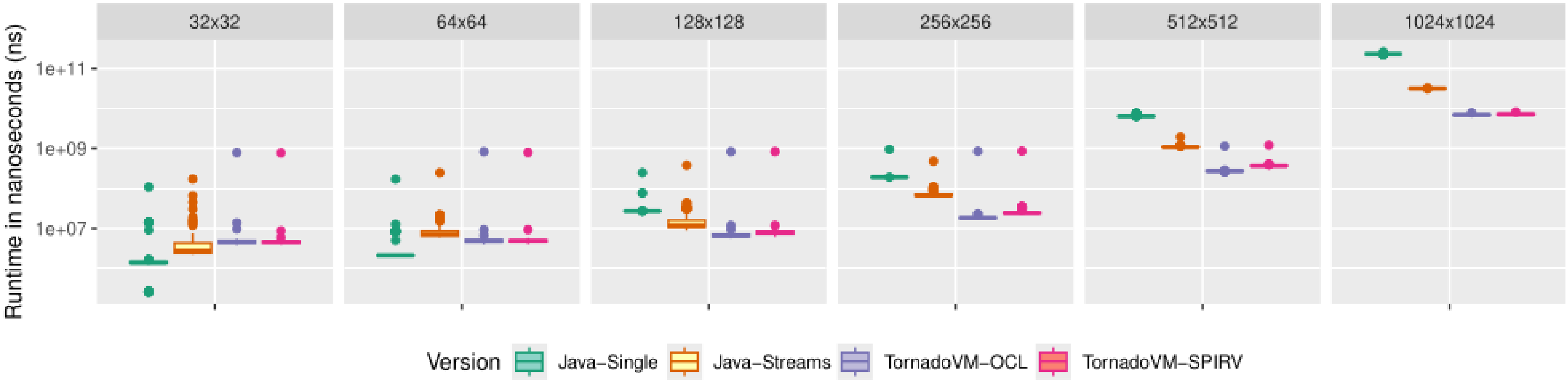


Things to be aware of

- Need to be extremely patient. This development board is extremely slow.
- ~4 days to compile LLVM on the RISC-V
  - Single thread compilation to avoid swapping
- ~2 days to compile OCK
  - Multiple configurations:
    - ***Refsi*** (RISC-V Emulator on RISC-V)
    - No vectorization
    - **With auto-vectorization**
- Use active cooling to avoid throttling



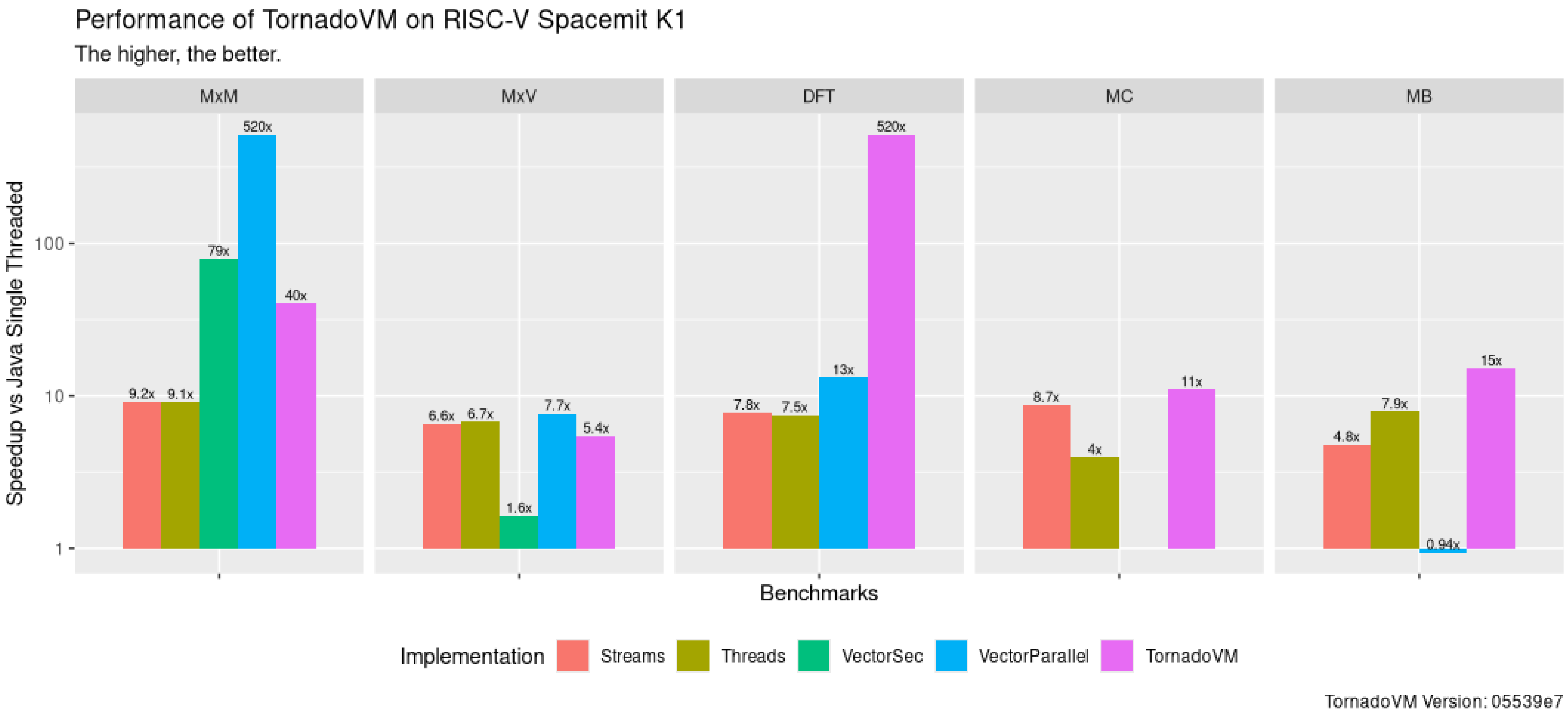
Performance of MxM on RISC-V (Banana PI 4GB)



TornadoVM 1.0.10-dev
(ec667bd65)
LLVM 19.1.5
GCC 13.2
OpenJDK 21.0.5
Banana PI F3

- For small matrices, stay on single core.
  - No auto-vectorization for Java on RISC-V as in FEB 2025 (we checked the generated Assembly code)
- Java streams is 2.3x-7.2x faster than single core for larger matrices
- TornadoVM + OCK achieves 4.1x-33x faster than Java seq, and up to 4.6x vs Streams
- SPIR-V backend is bit slower compared to OpenCL C

Performance on RISC-V Spacemit K1 (Lichee PI3A)



Explicit Vector API is quite competitive
Hard to express Vector API in some applications

TornadoVM: 0553ae7 (1.0.11-dev)
LLVM 19.1.5
GCC 13.2
OpenJDK 21.0.5
Lichee PI 3A

Vector API uses Intrinsics

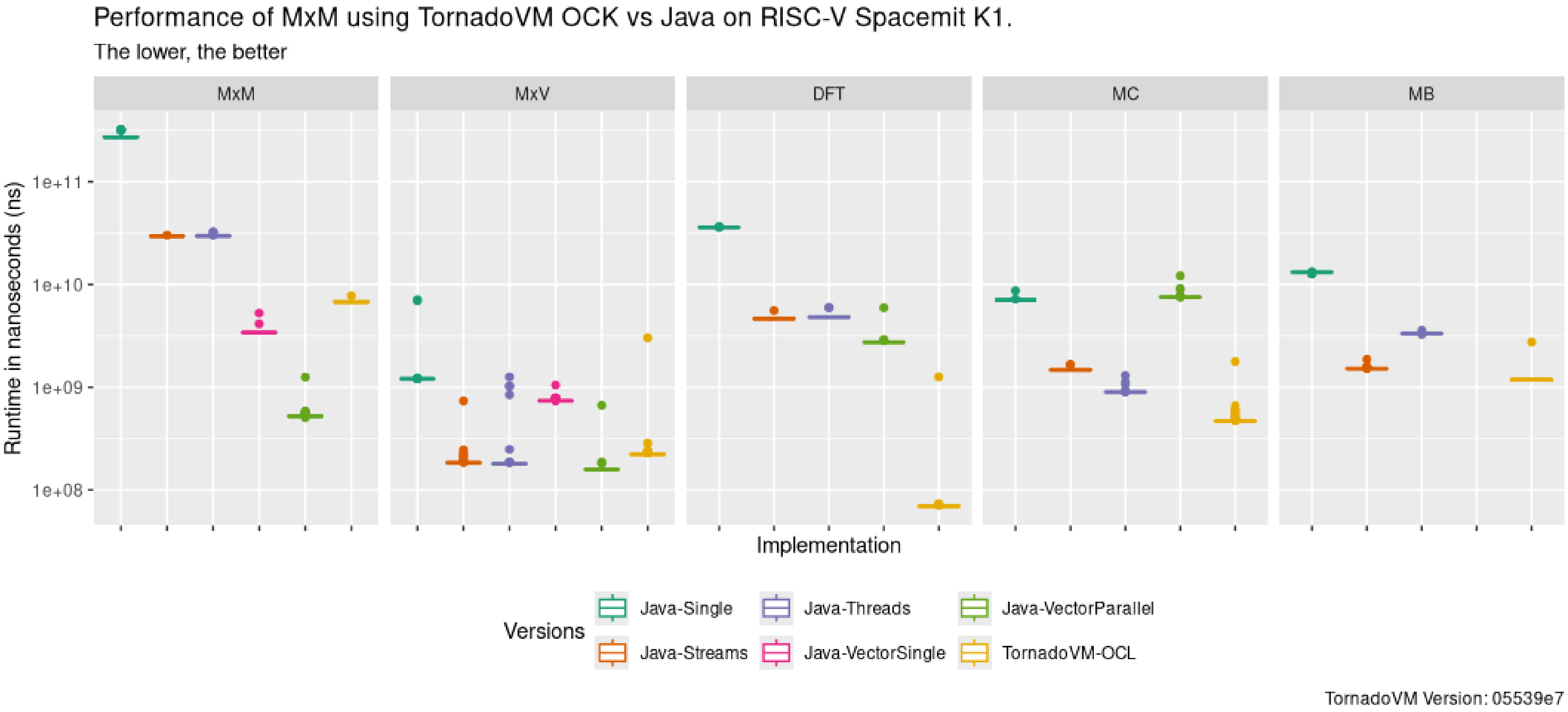
Java JIT Compiled Code (C2), using `-XX:+UnlockDiagnosticVMOptions -XX:+PrintAssembly`

```
0x00007f469cdb7a0f:   cmp    %r11,%r10
0x00007f469cdb7a12:   jne    0x00007f469cdb8ad4
0x00007f469cdb7a18:   mov    %rcx,%rax    ;*invokestatic reductionCoerced {reexecute=0 rethrow=0 return_oop=0}
                                          ; - jdk.incubator.vector.FloatVector::reduceLanesTemplate@78 (line 2644)
                                          ; - jdk.incubator.vector.Float256Vector::reduceLanes@2 (line 324)
                                          ; - tornadovm.benchmarks.DFT::lambda$computeWithParallelVectorAPI$1@263 (line
  136)
```

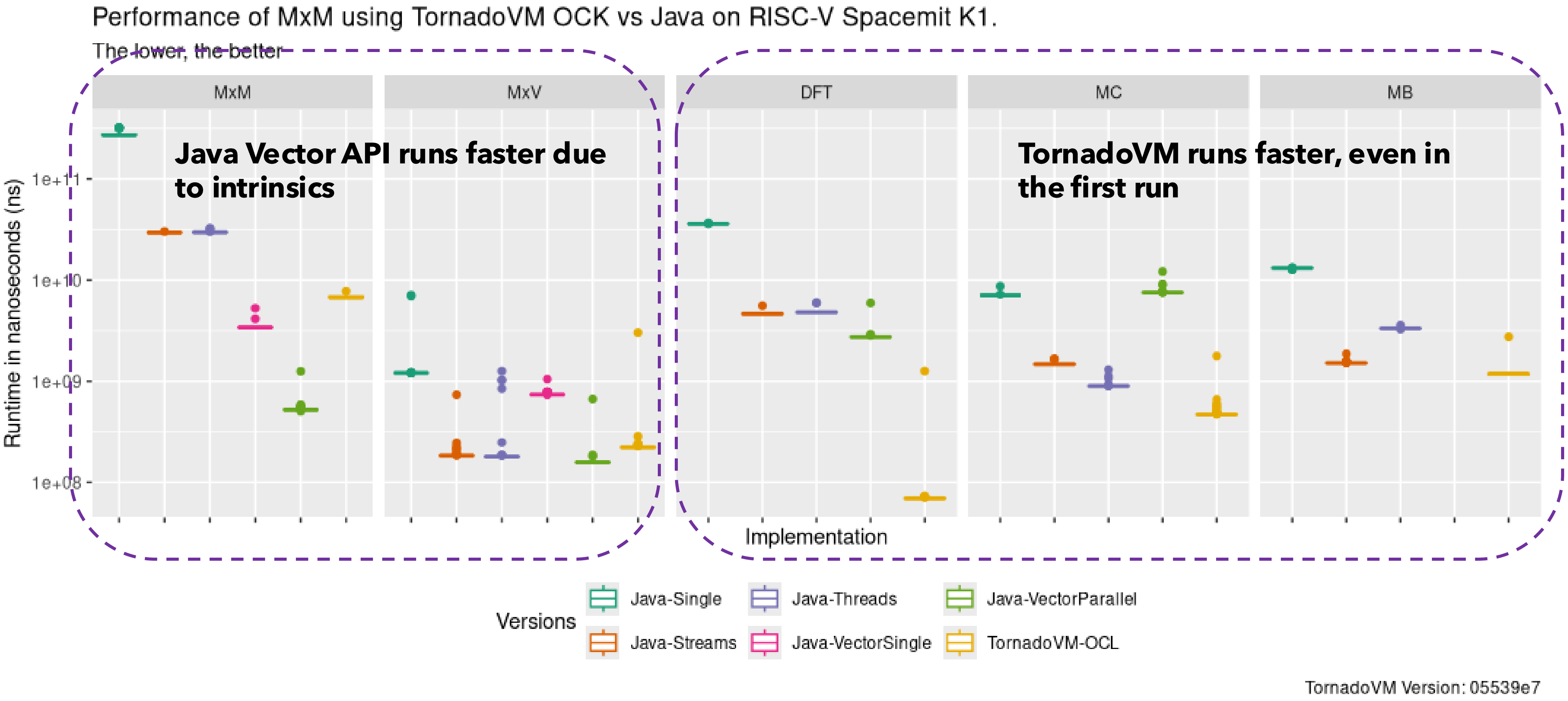
Java Source Code:

```
136  sumimag += -1 * vInReal.mul(FloatVector.fromArray(species, sinAngles, 0))
                    .add(vInImag.mul(FloatVector.fromArray(species, cosAngles, 0)))
                    .reduceLanes(VectorOperators.ADD);
```

Runtime Distribution on RISC-V Spacemit K1 (Lichee PI3A)



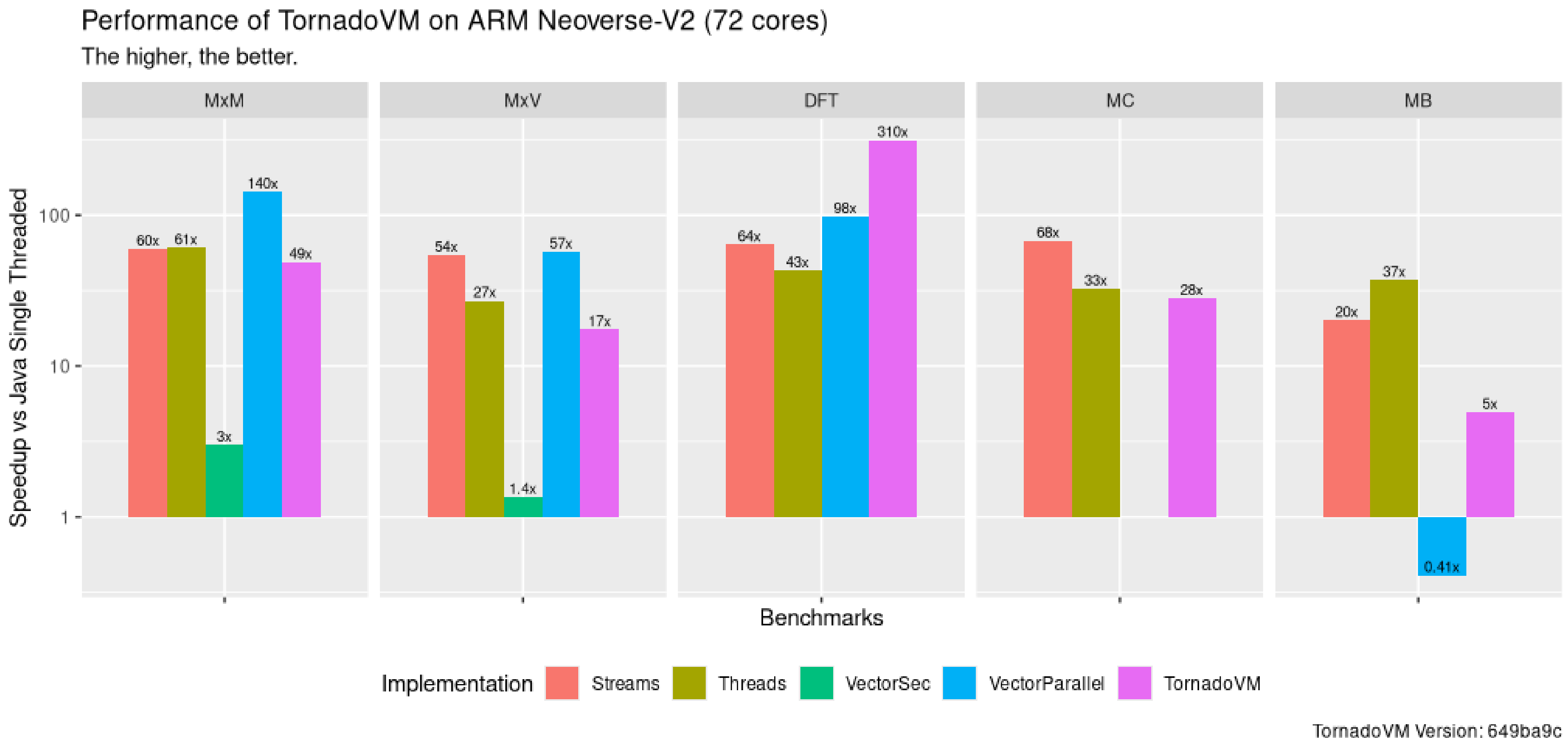
Runtime Distribution on RISC-V Spacemit K1 (Lichee PI3A)



Java Vector API runs faster due to intrinsics

TornadoVM runs faster, even in the first run

[**ARM CPUs**] ARM Neoverse V2 (Grace Hopper)


Performance of TornadoVM on ARM Neoverse-V2 (72 cores)

The higher, the better.

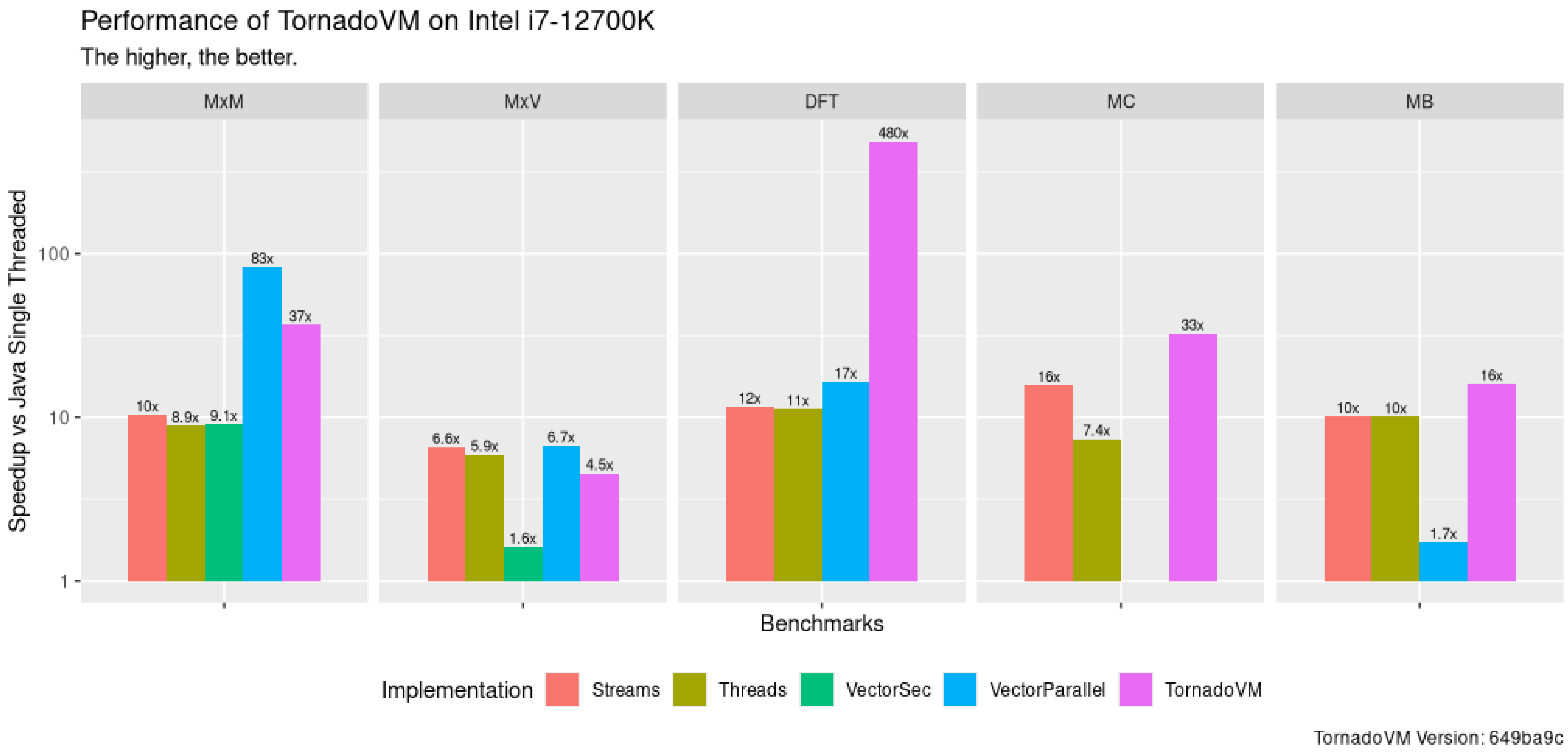
TornadoVM Version: 649ba9c

ARM Neoverse V2 CPU (NVIDIA Grace Hopper)
Ubuntu 22.04.4 LTS
Kernel 6.2.0-1015-nvidia-64k
OCK 4.0: 5be5a8d
TornadoVM 1.0.11-dev: 649ba9c
OpenJDK 21.0.3+7-LTS

Java multi-threaded is very competitive in this platform, since OCK does not use SVE instructions. Explicit vectorization is a mixed: if auto-vectorization works, there is no major benefit, at least for these benchmarks.

[**INTEL CPUs**] TornadoVM + OCK/oneAPI

MANCHESTER 1824
The University of Manchester


Performance of TornadoVM on Intel i7-12700K
The higher, the better.

Intel i7-12700K
Fedora 41
Kernel 6.2.15-200
OCK 4.0: a537ec99
TornadoVM 1.0.11-dev: 649ba9c
OpenJDK 21.0.6

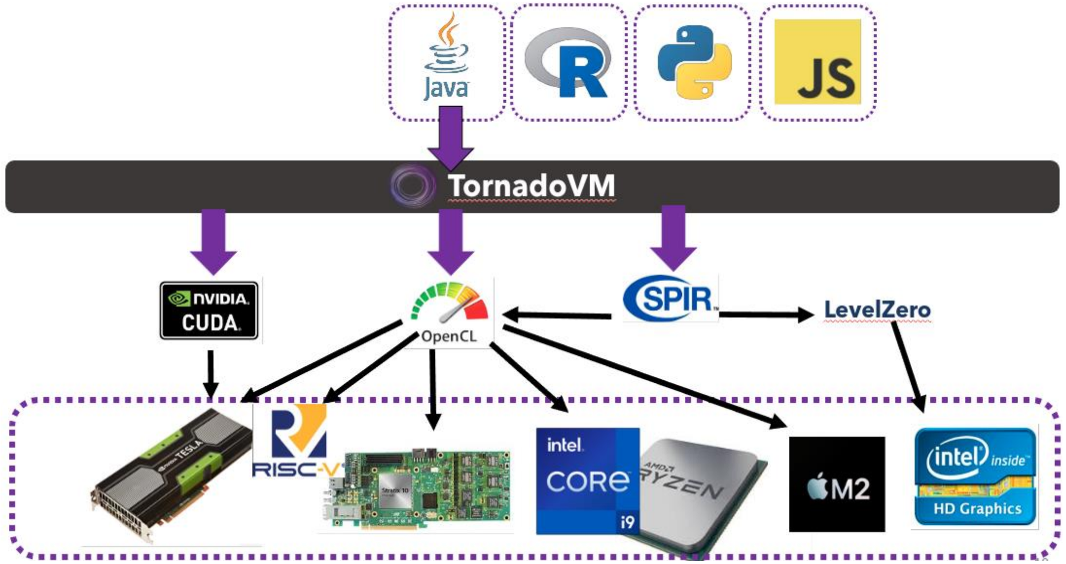
For Intel CPUs, the Java JIT compiler generates vector instructions
TornadoVM + OCK generates efficient code just for the kernel of interest

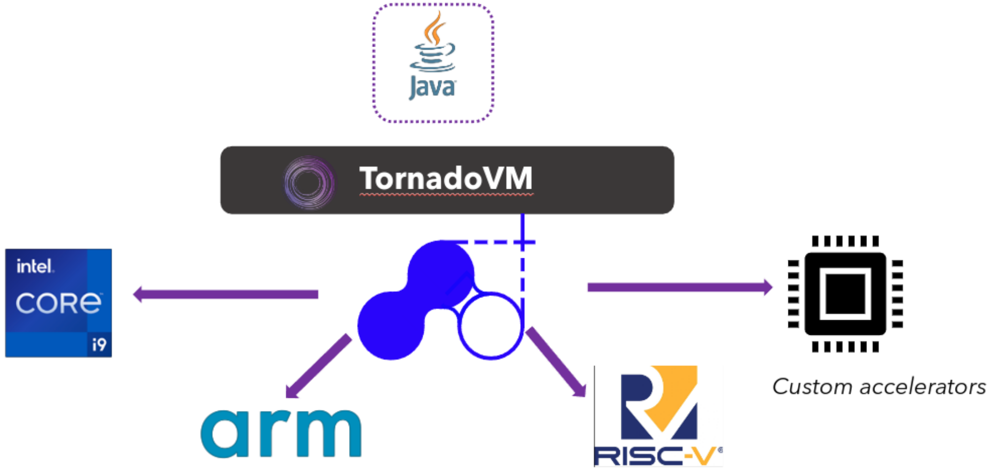
TORNADOVM

www.tornadovm.org

Conclusions

Key Takeaways

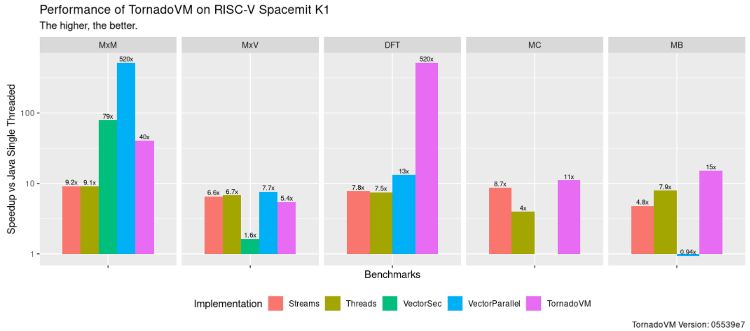




Custom accelerators

Faster than Java multi-core

Usually slower than explicit Java vectorization


Performance of TornadoVM on RISC-V Spacemit K1

The higher, the better.

TornadoVM Version: 05539e7

tornadovm.org

Collaborations and Projects

AERO

Project Number: 101092850

INCODE

Project Number: 101093069

encrypt

Project Number: 101070670

TANGO

Project Number: 101070052

INTEL oneAPI

UK Research and Innovation

MANCHESTER
1824

The University of Manchester

AERO

European
Commission

Thank you!

juan.fumero@manchester.ac.uk

@jfumero.bsky.social

@snatverk

Discussions