TornadoVM: Multi-Backend Hardware Acceleration Framework for Java

Juan Fumero, PhD

Research Fellow at The University of Manchester, UK

@snatverk

MANCHESTER 1824

The University of Manchester

oneAPI Language SIG

14th March 2023

TORNADOVM

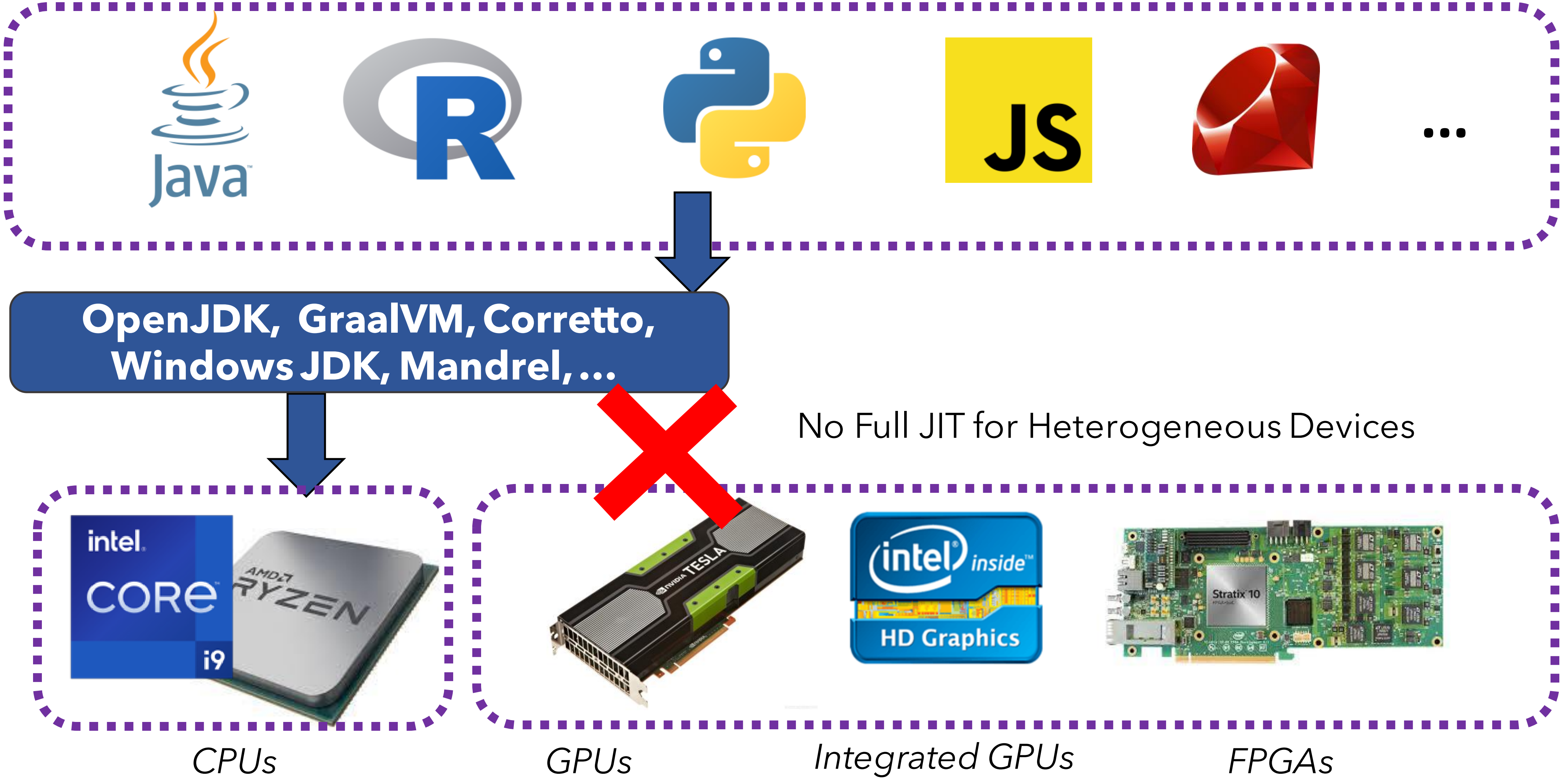
Outline

1. Motivation
2. Quick Overview of TornadoVM
3. TornadoVM's main abstractions for CPUs, GPUs & FPGAs
    1. User API abstractions
    2. Abstractions in the Runtime System
    3. Abstractions in the JIT Compiler
4. Feedback to the oneAPI/LevelZero Software Stack

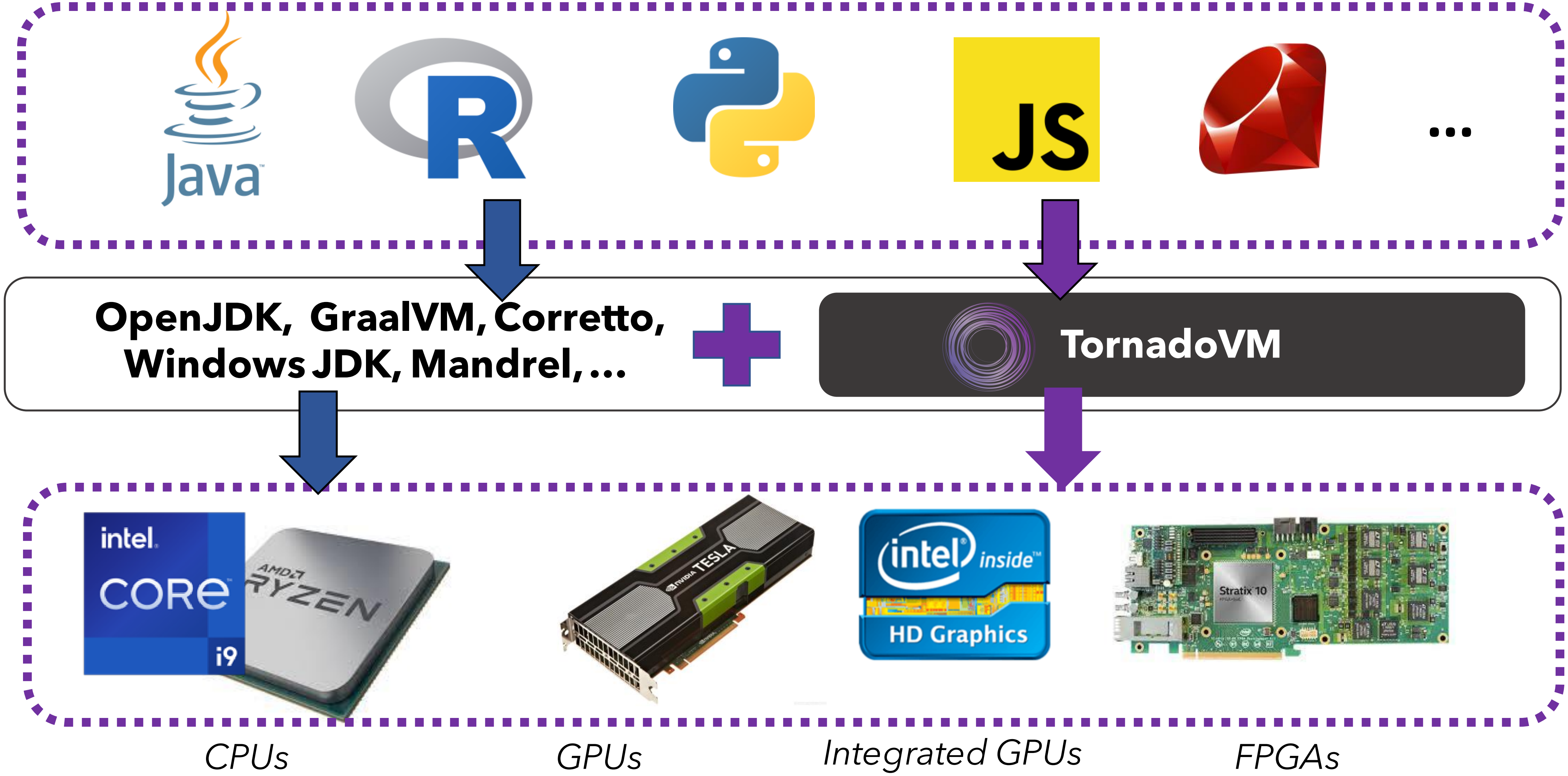
Motivation

Fast Path to GPUs and FPGAs

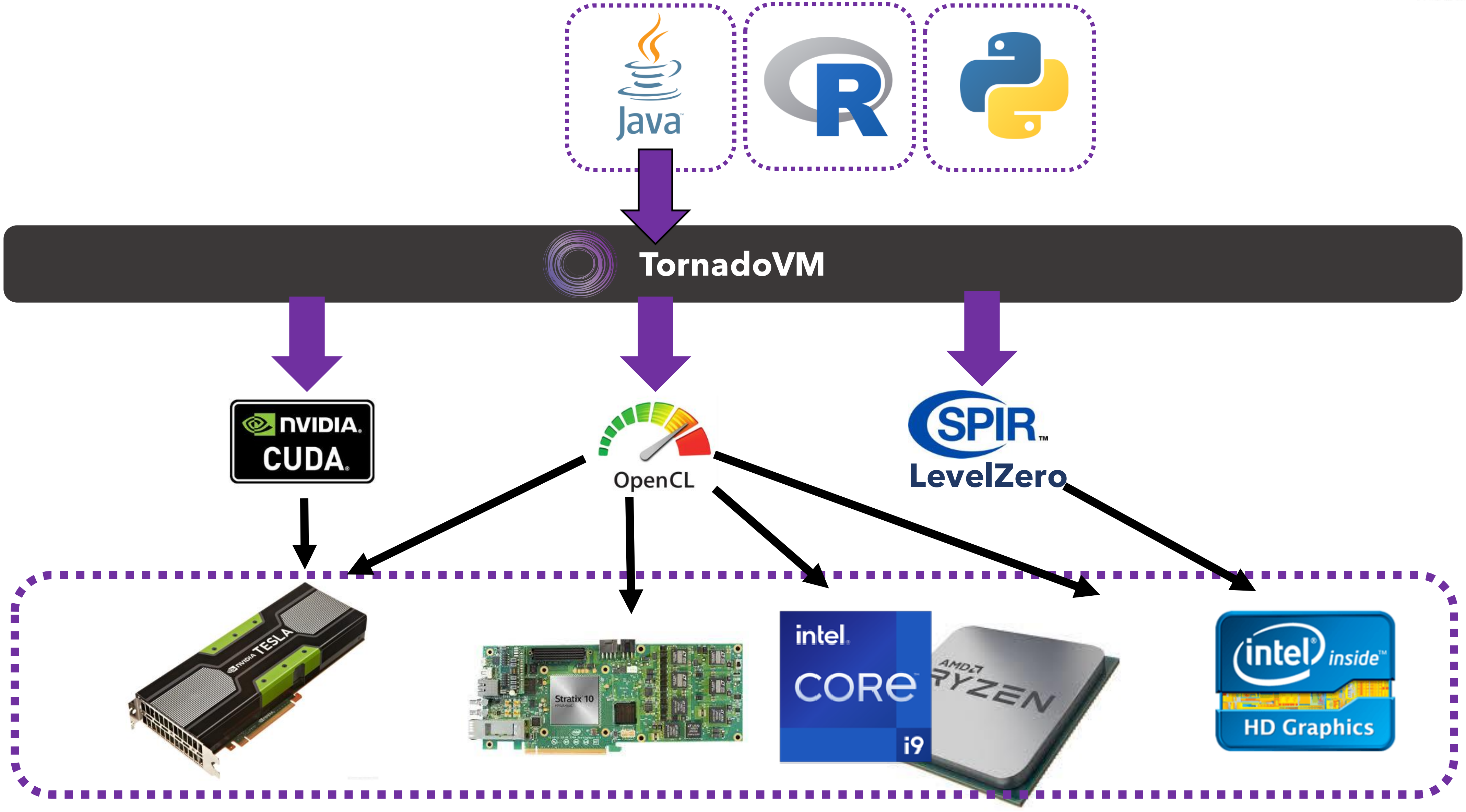
MANCHESTER 1824
The University of Manchester



Fast Path to GPUs and FPGAs



Enabling Acceleration for Managed Runtime Languages



TORNADOVM

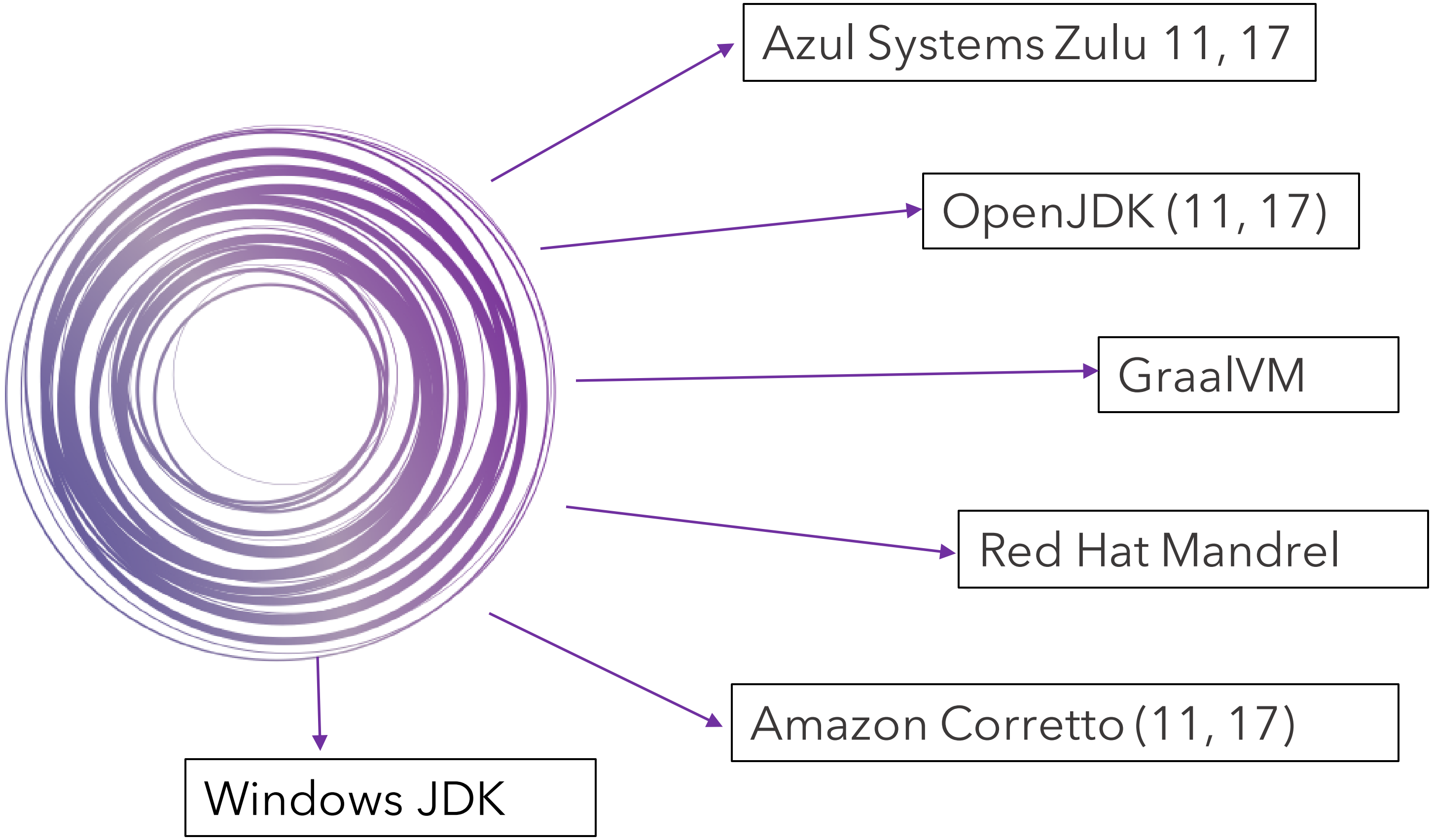
www.tornadovm.org

TornadoVM Overview

MANCHESTER 1824
The University of Manchester

www.tornadovm.org

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

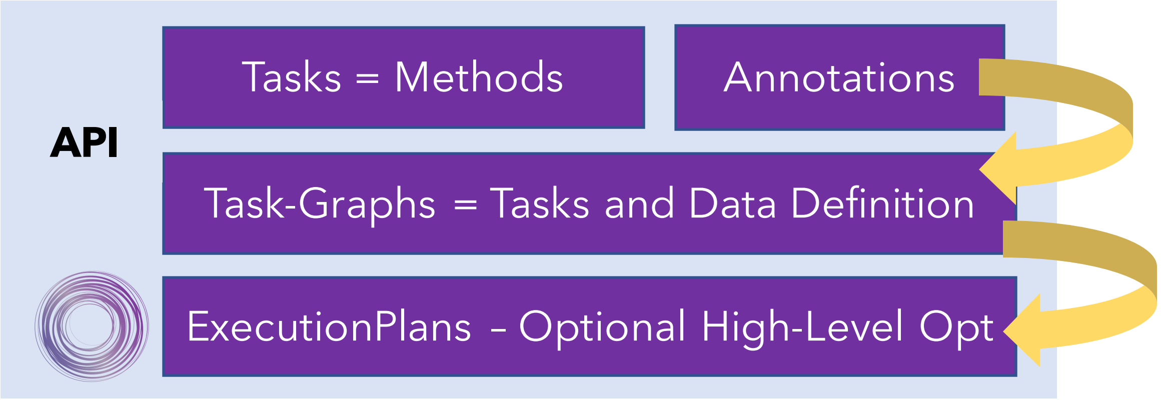


Features such as:
- Dynamic Reconfiguration
- Cloud deployment | AWS
- Multiple devices | CPU, GPU, FPGA
- JIT compilation specialization
- Specialized optimizations
- Hardware Agnostic

- And more …

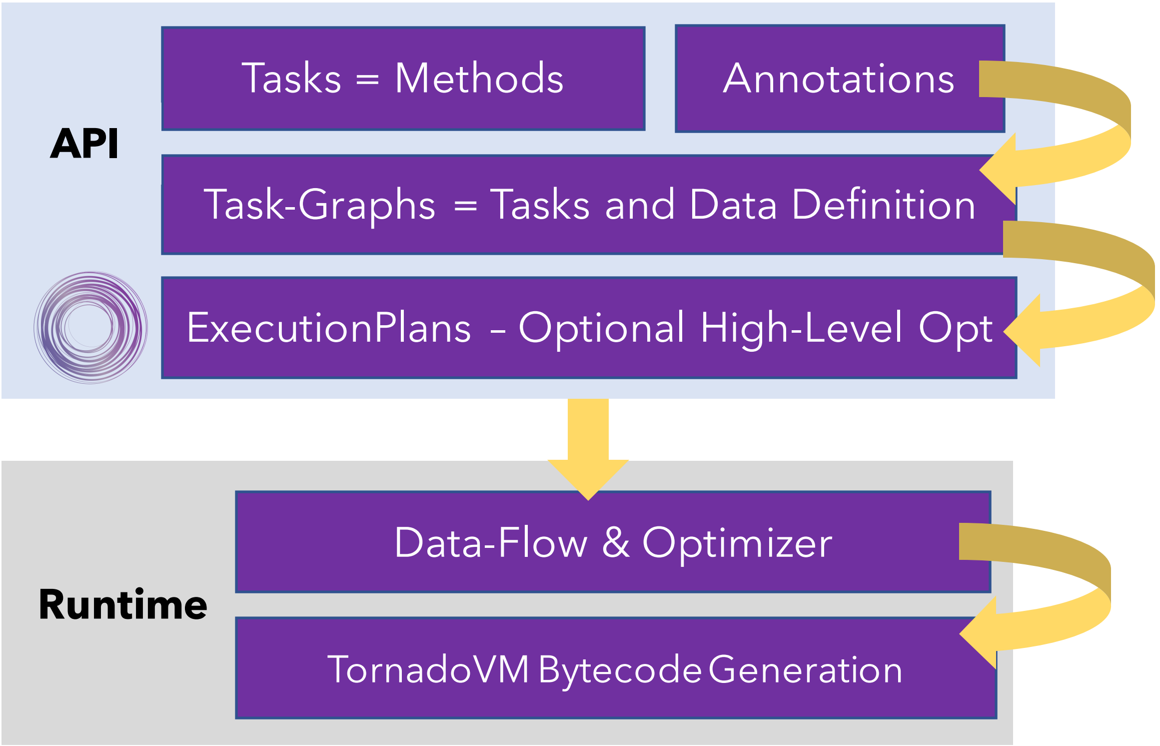
License: GPLv2 + CE

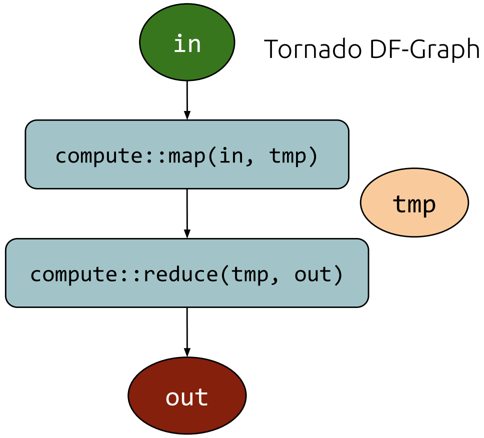
TornadoVM Software Stack



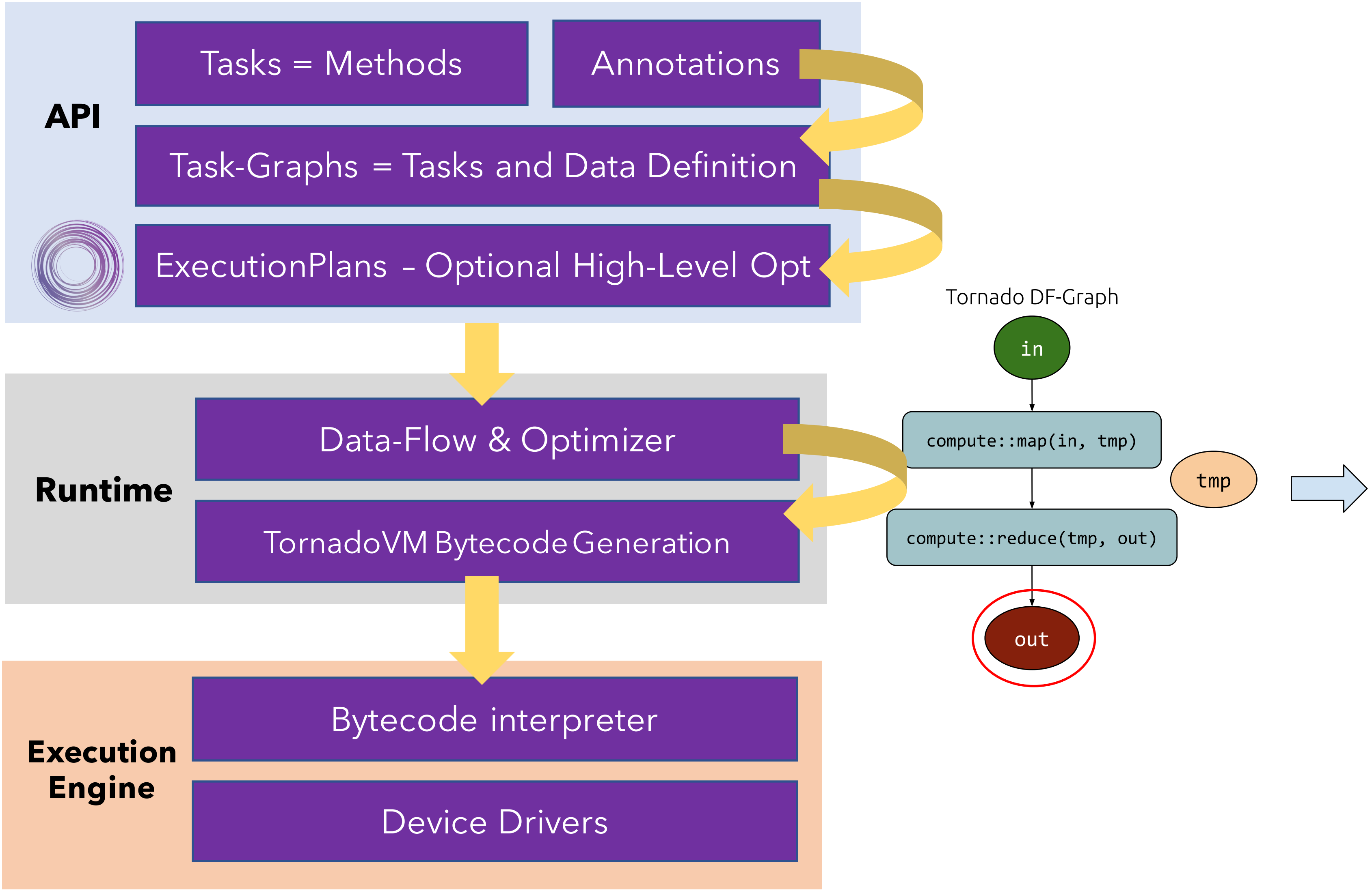

```
void map(params ... ){
  for (@Parallel int r = 0; r < numRows; r++) {
    for (@Parallel int c = 0; c < numCols; c++){
      compute(. . . );
    }
  }
}

void reduce(@Reduce params ... ){
  for (@Parallel int r = 0; r < size; r++){
    ...
  }
}
```

TornadoVM Software Stack





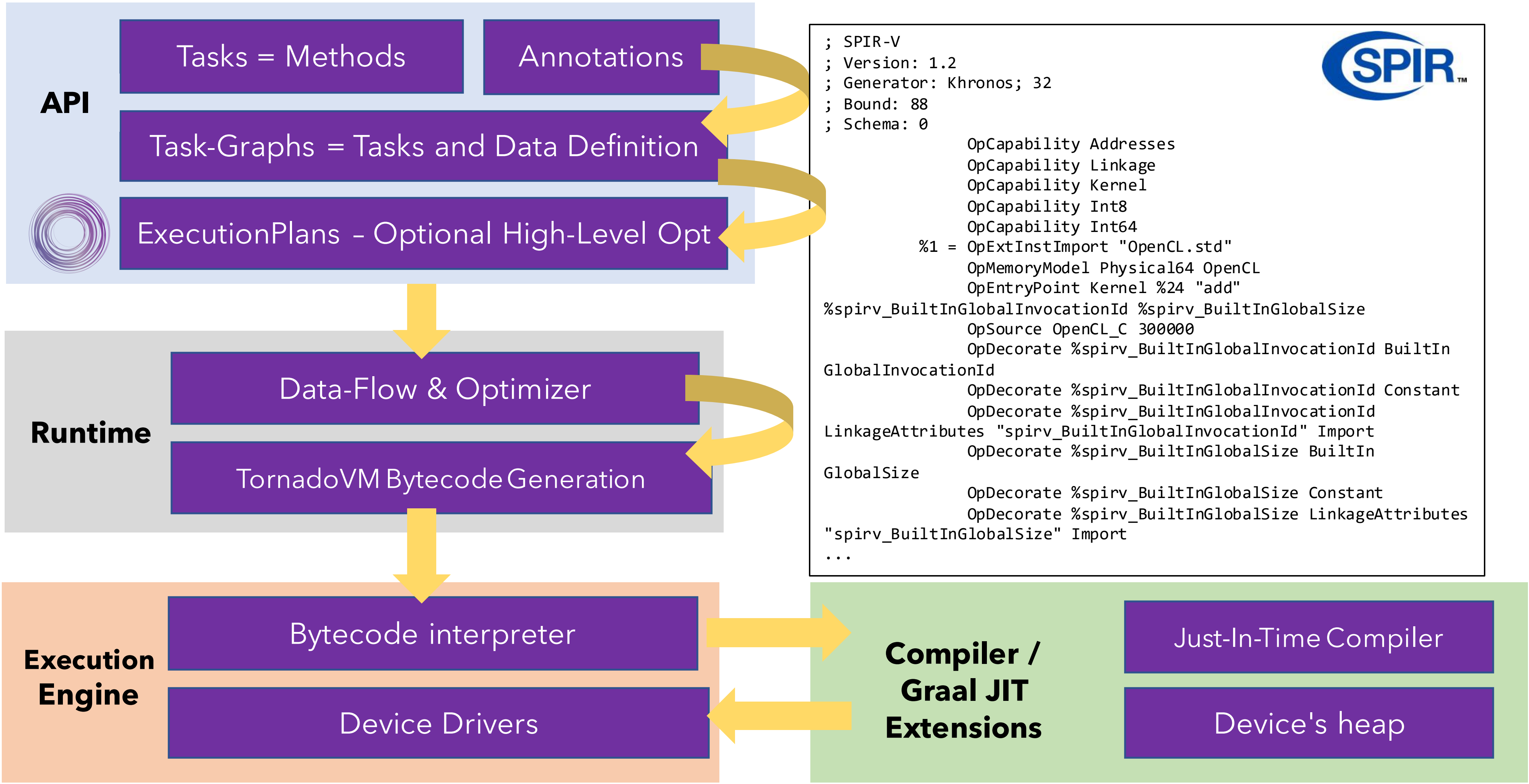
TornadoVM Software Stack



Persist tmp on the GPU

TornadoVM Software Stack

MANCHESTER 1824
The University of Manchester




```
; SPIR-V
; Version: 1.2
; Generator: Khronos; 32
; Bound: 88
; Schema: 0
               OpCapability Addresses
               OpCapability Linkage
               OpCapability Kernel
               OpCapability Int8
               OpCapability Int64
          %1 = OpExtInstImport "OpenCL.std"
               OpMemoryModel Physical64 OpenCL
               OpEntryPoint Kernel %24 "add"
%spirv_BuiltInGlobalInvocationId %spirv_BuiltInGlobalSize
               OpSource OpenCL_C 300000
               OpDecorate %spirv_BuiltInGlobalInvocationId BuiltIn
GlobalInvocationId
               OpDecorate %spirv_BuiltInGlobalInvocationId Constant
               OpDecorate %spirv_BuiltInGlobalInvocationId
LinkageAttributes "spirv_BuiltInGlobalInvocationId" Import
               OpDecorate %spirv_BuiltInGlobalSize BuiltIn
GlobalSize
               OpDecorate %spirv_BuiltInGlobalSize Constant
               OpDecorate %spirv_BuiltInGlobalSize LinkageAttributes
"spirv_BuiltInGlobalSize" Import
...
```

Points of Parallel Programming Models Abstractions

1. User API:
    - Fully Hardware Agnostic Programming Interface
2. Runtime GPU/CPU/FPGA Program Orchestration
    - TornadoVM Bytecodes
    - Device Abstraction
3. Compiler IR abstraction
    - Common IR for all backends (Java bytecode -> Graal IR -> TornadoVM Common IR)

1. User APIs

Different components of the User API

MANCHESTER 1824
The University of Manchester

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

Tornado API – example using Annotations

MANCHESTER 1824
The University of Manchester

```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);
        }
     }
  }
}
```

Tornado API – example using Annotations

MANCHESTER 1824
The University of Manchester

```java
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 as a hint for the compiler

We only have 2 annotations:

@Parallel
@Reduce

+ A light API to identify which methods to accelerate

Tornado API – example using Kernel Context

```java
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(i, k) * B.get(k, j);
     C.set(i, j, sum);
  }
}
```

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

MANCHESTER 1824
The University of Manchester

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

```
TaskGraph taskGraph = new TaskGraph("s0")
    .transferToDevice(DataTransferMode.EVERY_EXECUTION , matrixA, matrixB)
    .task("t0", objectCompute::mxm, matrixA, matrixB, matrixC, size)
    .transferToHost(DataTransferMode.EVERY_EXECUTION, matrixC);
```

Host Code

Task-Graph is a new Tornado 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

Adding Execution Plans

MANCHESTER 1824
The University of Manchester

How to explore different optimizations? --> ExecutionPlan

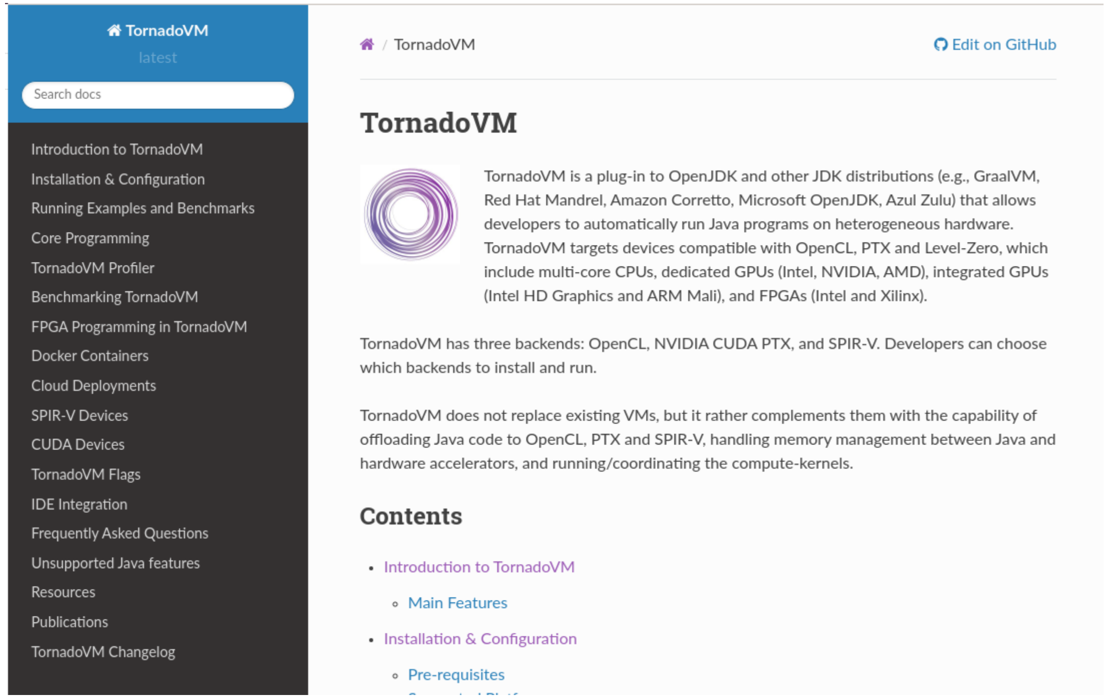
```
ImmutableTaskGraph itg = taskGraph.snapshot();

TornadoExecutionPlan executionPlan = new TornadoExecutionPlan(itg);

executionPlan.withWarmUp()
             .withProfiler(ProfilerMode.CONSOLE)
             .withDynamicReconfiguration(PERFORMANCE, PARALLEL);

TornadoExecutionResult result = executionPlan.execute();

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

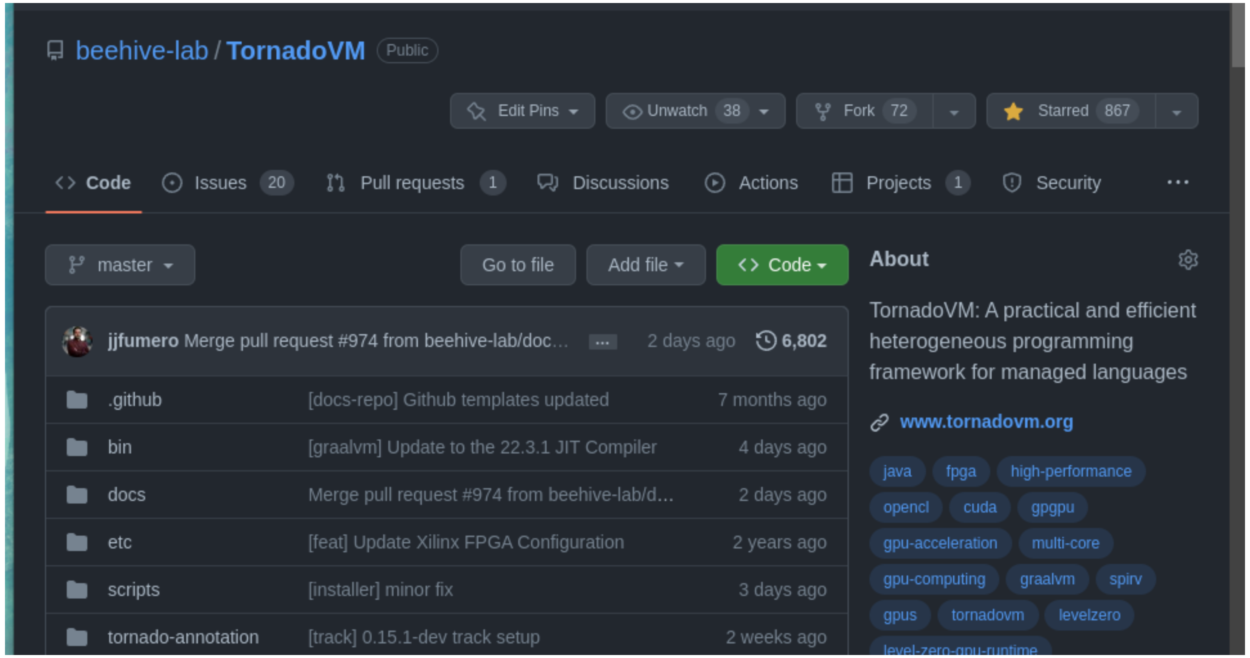
Host Code

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 know more about the APIs





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

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

Points of Abstraction

1. User API:
   - Fully Hardware Agnostic Programming Interface
2. **Runtime GPU/CPU/FPGA Program Orchestration**
   - **TornadoVM Bytecodes**
   - **Device Abstraction**
3. Compiler IR abstraction
   - Common IR for all backends (Java bytecode -> Graal IR -> TornadoVM Common IR)

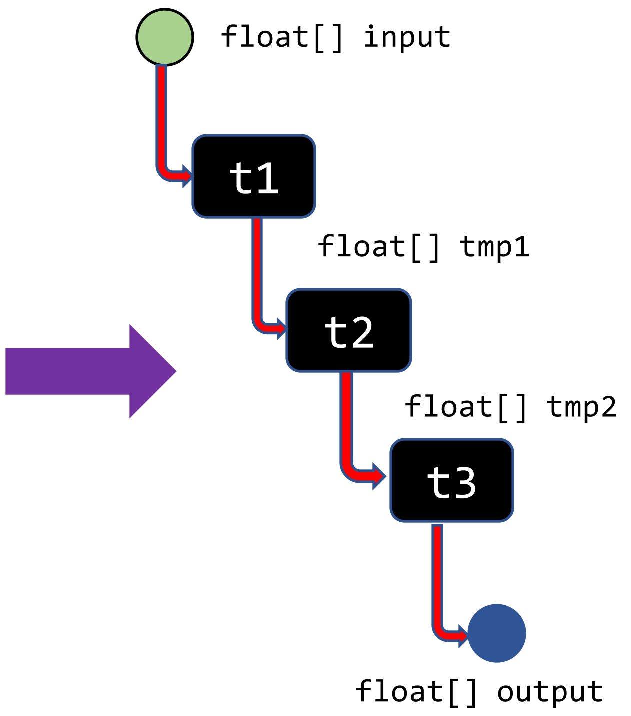
The Main Abstraction: TornadoVM Bytecode + Immediate Actions

MANCHESTER 1824
The University of Manchester

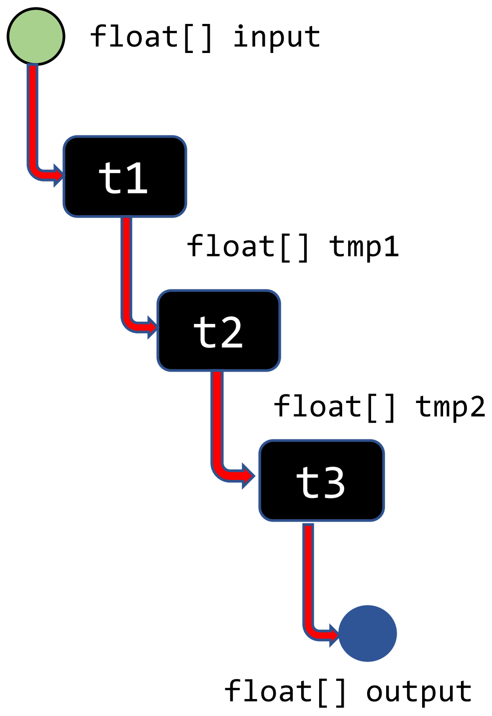
```java
public class Sample {
      public void task1(float[] input, float[] tmp1) {…}
      public void task2(float[] tmp1,  float[] tmp2) {…}
      public void task3(float[] tmp2,  float[] output) {…}

      public void buildTaskGraphAndPlan(float[] input, float[] output) {
         TaskGraph tg = new TaskGraph("sample");
         tg.transferToDevice(DataTransferMode.EVERY_EXECUTION, input, out1, out2)
           .task("t1", this::task1, input, tmp1)
           .task("t2", this::task2, tmp1, tmp2)
           .task("t3", this::task3, tmp2, output)
           .transferToHost(DataTransferMode.EVERY_EXECUTION, output);

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

[RUNTIME] Build a Data-Flow Graph


```java
public class Sample {
	public void task1(float[] input, float[] tmp1) {…}
	public void task2(float[] tmp1,  float[] tmp2) {…}
	public void task3(float[] tmp2,  float[] output) {…}

	public void buildTaskGraphAndPlan(float[] input, float[] output) {
	   TaskGraph tg = new TaskGraph("sample");
	   tg.transferToDevice(DataTransferMode.EVERY_EXECUTION, input)
	     .task("t1", this::task1, input, tmp1)
	     .task("t1", this::task1, tmp1, tmp2)
	     .task("t1", this::task1, tmp2, output)
	     .transferToHost(DataTransferMode.EVERY_EXECUTION, output);

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

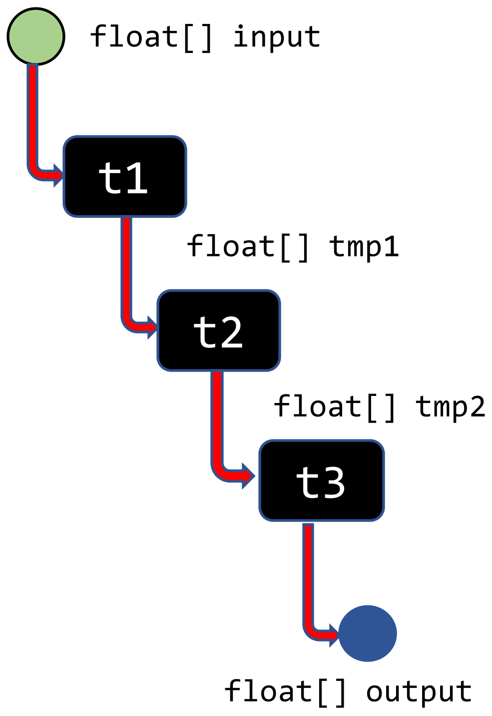


[RUNTIME] Generate BC From DFG



[RUNTIME] Generate BC From DFG

One Java Thread Per VM instance

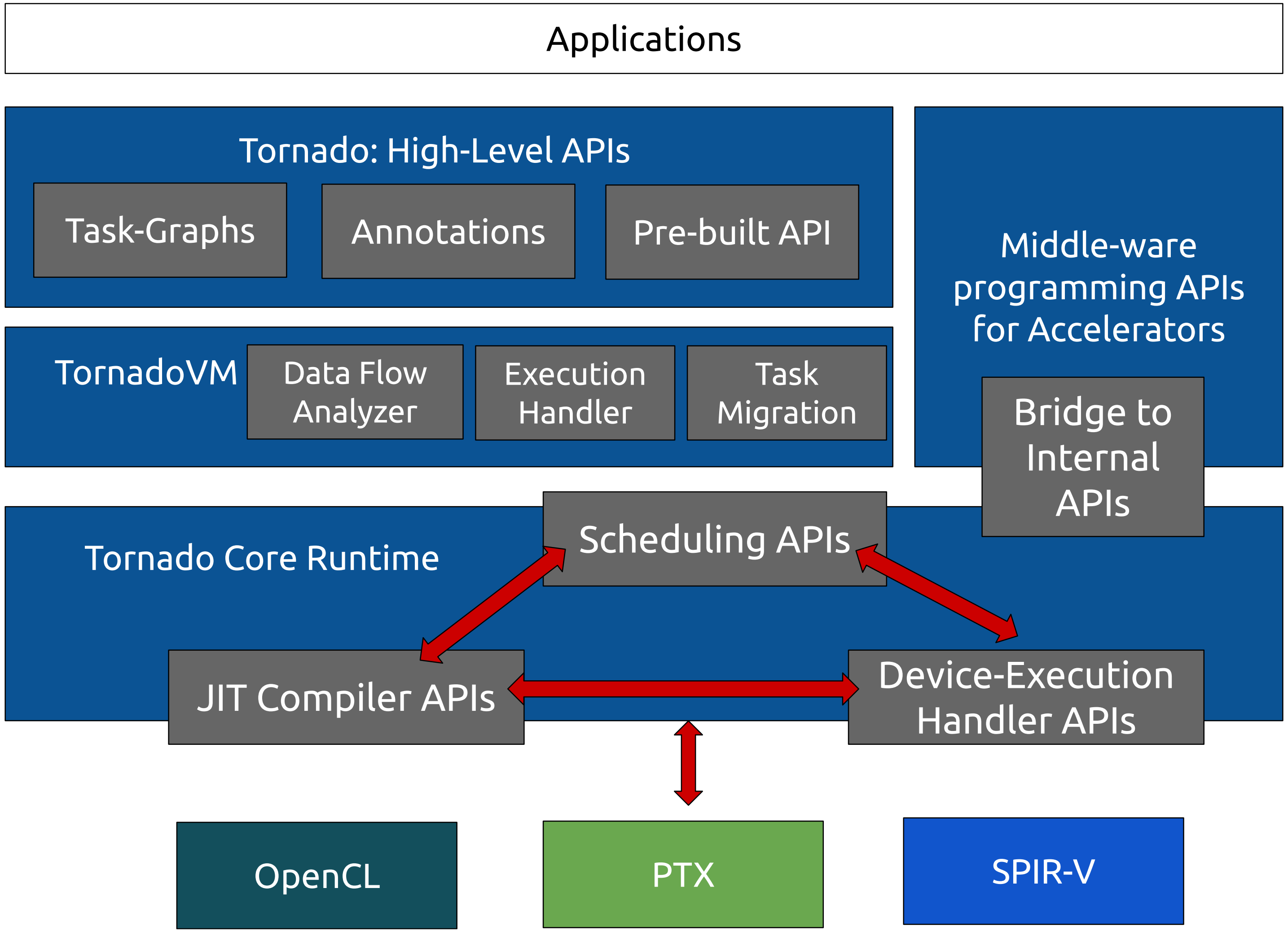



```
BEGIN CONTEXT <ID>
   ALLOC input
   ALLOC tmp1
   ALLOC tmp2
   ALLOC output
   TRANSFER_TO_DEVICE, INPUT, SIZE, OFFSET:0
     LAUNCH T1, INPUT, TMP1
   BARRIER
     LAUNCH T2, TMP1, TMP2
   BARRIER
     LAUNCH T3, TMP2, OUTPUT
   TRANSFER_TO_HOST, OUTPUT, SIZE, OFFSET:0
   BARRIER
   DEALLOC INPUT
   DEALLOC TMP1
   DEALLOC TMP2
   DEALLOC OUTPUT
END
```

TornadoVM can reorder bytecode and repeat patterns (e.g., batch processing), under demand

Dynamic Application Reconfiguration on Heterogeneous Hardware. VEE'19

[RUNTIME] Hardware Agnostic Middle-ware APIs



The TornadoVM BC Interpreter makes calls to the middle-ware API

[RUNTIME] Hardware Agnostic Middle-ware APIs

```java
// 1. Use Java Reflection to obtain the Method reference
Method methodToCompile = CompilerUtil.getMethodForName(klass, methodName);

// 2. Get Tornado Runtime
TornadoCoreRuntime tornadoRuntime = TornadoCoreRuntime.getTornadoRuntime();

// 3. Get the Graal Resolved Java Method
ResolvedJavaMethod resolvedJavaMethod = tornadoRuntime.resolveMethod(methodToCompile);

// 4. Get the backend from TornadoVM
SPIRVBackend spirvBackend = tornadoRuntime.getDriver(SPIRVDriver.class).getDefaultBackend();

// 5. Obtain the SPIR-V device
TornadoDevice device = tornadoRuntime.getDriver(SPIRVDriver.class).getDefaultDevice();

// 6. Create a new task for TornadoVM
ScheduleMetaData scheduleMetaData = new ScheduleMetaData("s0");

// 7. Create a compilable task
CompilableTask compilableTask = new CompilableTask(scheduleMetaData, methodToCompile, data);
TaskMetaData taskMeta = compilableTask.meta();
taskMeta.setDevice(device);
```

The TornadoVM BC Interpreter makes calls to the middle-ware API

[RUNTIME] Implementation some BC

ALLOC

```
// First, we allocate array A
GlobalObjectState stateA = new GlobalObjectState();
DeviceObjectState objectStateA = stateA.getDeviceState(spirvTornadoDevice);

float[] a = new float[size]; …
device.allocateBulk(new Object[] { a },
                    0,
                    new TornadoDeviceObjectState[] { objectStateA });
```

The TornadoVM BC Interpreter makes calls to the middle-ware API

TRANSFER_TO_DEVICE (READ ONLY)

```
device.ensurePresent(a, objectStateA, offset);
```

https://jjfumero.github.io/posts/2022/09/tornadovm-internal-apis/

Points of Abstraction

1. User API:
   - Fully Hardware Agnostic Programming Interface
2. Runtime GPU/CPU/FPGA Program Orchestration
   - TornadoVM Bytecodes
   - Device Abstraction
3. **Compiler IR abstraction**
   - **Common IR for all backends (Java bytecode -> Graal IR -> TornadoVM Common IR)**

Multi-backend JIT Compiler Workflow

Programmer's view

```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

Multi-backend JIT Compiler Workflow

The University of Manchester

Programmer's view

```
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

Java Bytecodes

Static Compilation: No Modifications in Javac

TornadoVM JIT Compiler

Multi-backend JIT Compiler Workflow

MANCHESTER 1824
The University of Manchester

Programmer's view

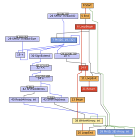
```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

Java Bytecodes

TornadoVM JIT Compiler

Graal IR



TornadoVM Common IR

Multi-backend JIT Compiler Workflow

Programmer's view

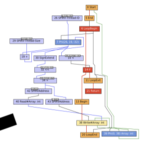
```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

javac

Java Bytecodes

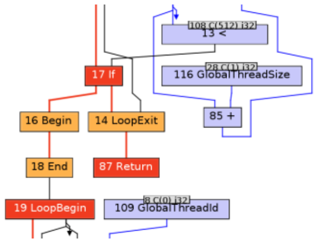
TornadoVM JIT Compiler

Graal IR

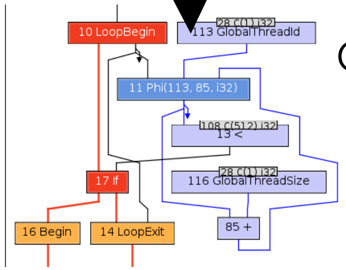


TornadoVM Common IR -> **Sketch**

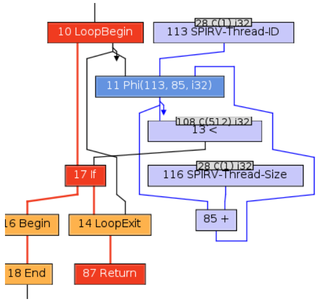
TornadoVM IR for PTX



TornadoVM IR for OpenCL



TornadoVM IR for SPIR-V

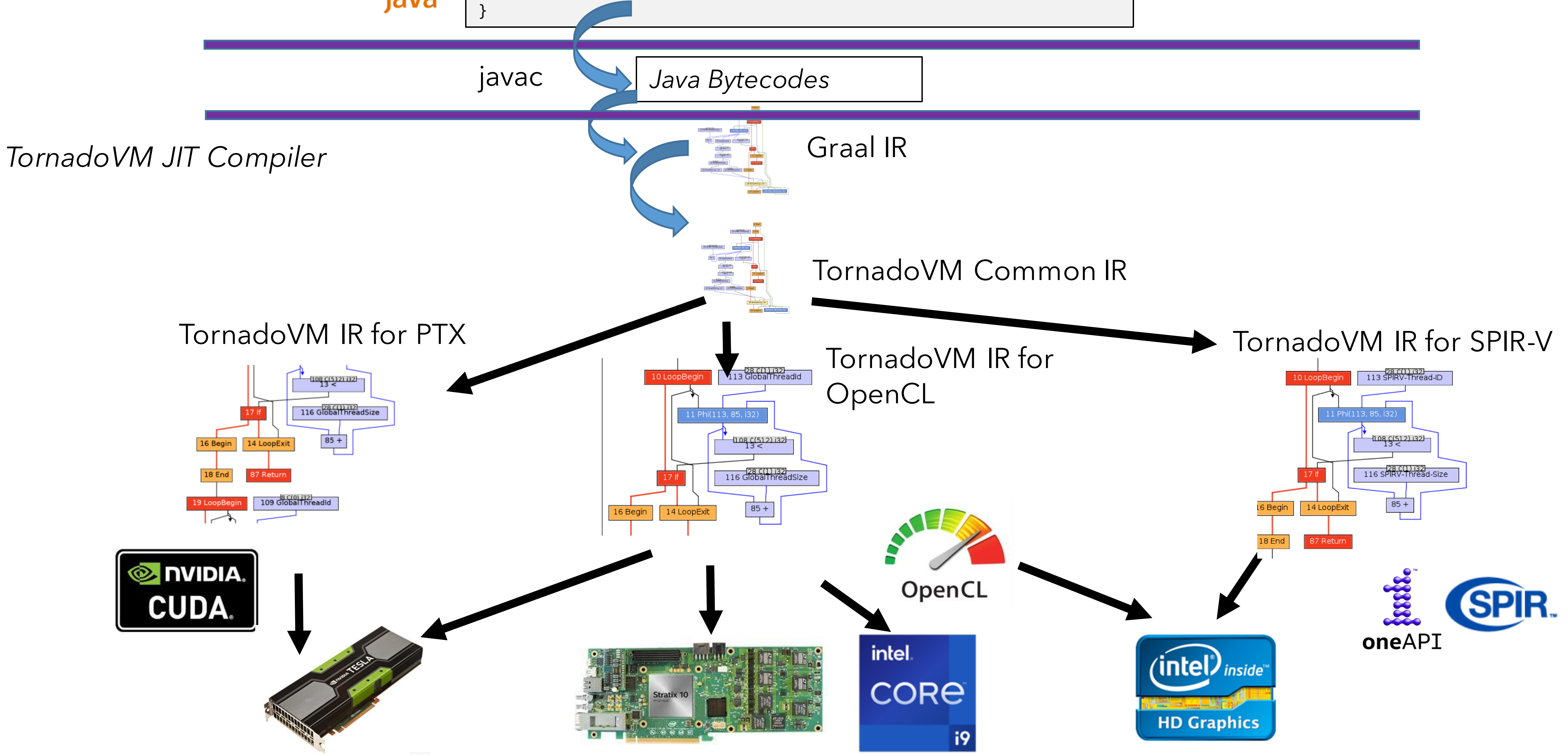


Multi-backend JIT Compiler Workflow

MANCHESTER 1824
The University of Manchester

Programmer's view

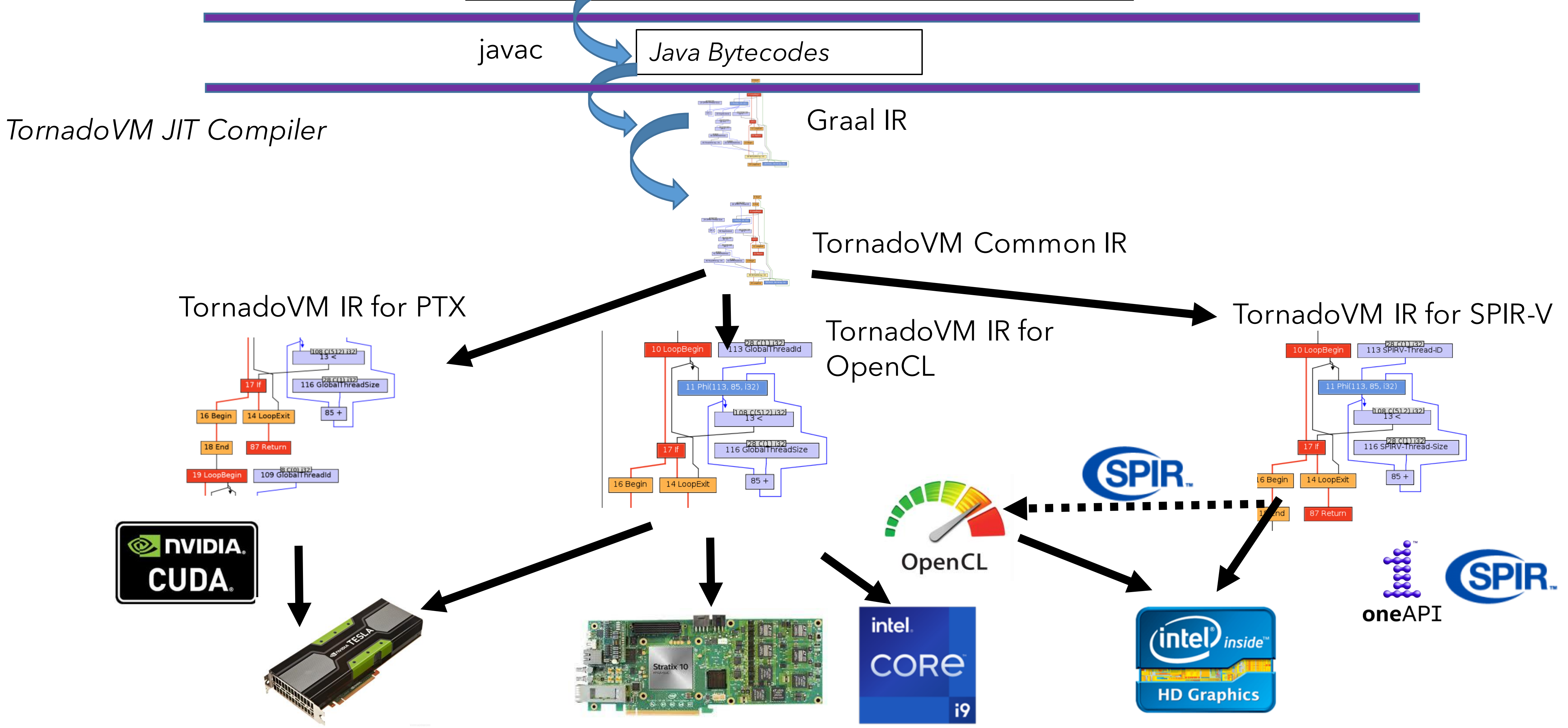
```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```



Multi-backend JIT Compiler Workflow

Programmer's view

```java
public static void saxpy(int[] a, int[] b, int[] c, int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```



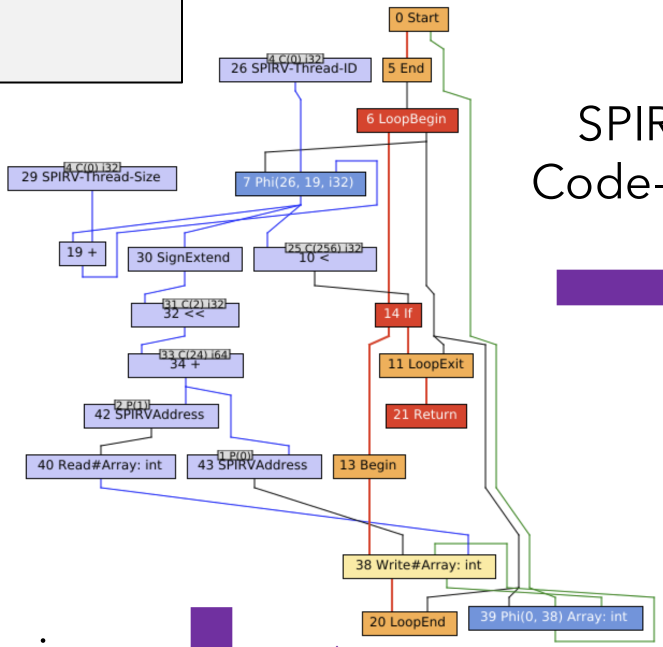
SPIR-V JIT compiler (and runtime) for TornadoVM

MANCHESTER 1824
The University of Manchester

Java

```java
public static void saxpy(int[] a,
                         int[] b,
                         int[] c,
                         int alpha) {
    for (@Parallel int i = 0; i < a.length; i++) {
      a[i] = alpha * b[i] + c[i];
    }
}
```

Build
Graal/Tornado IR



JIT Optimizer
For SPIR-V

SPIR-V
Code-Gen

SPIR

```
. . .
%B2 = OpLabel
%77 = OpLoad %uint %spirv_i_4 Aligned 4
%78 = OpSConvert %ulong %77
OpStore %spirv_l_6 %78 Aligned 8
%79 = OpLoad %ulong %spirv_l_6 Aligned 8
%80 = OpShiftLeftLogical %ulong %79 %uint_2
OpStore %spirv_l_7 %80 Aligned 8
%81 = OpLoad %ulong %spirv_l_7 Aligned 8
%82 = OpIAdd %ulong %81 %ulong_24
OpStore %spirv_l_8 %82 Aligned 8
%83 = OpLoad %ulong %spirv_l_1 Aligned 8
%84 = OpLoad %ulong %spirv_l_8 Aligned 8
%85 = OpIAdd %ulong %83 %84
OpStore %spirv_l_9 %85 Aligned 8
%86 = OpLoad %ulong %spirv_l_9 Aligned 8
%87 = OpConvertUToPtr %_ptr_CrossWorkgroup_uint %86
%88 = OpLoad %uint %87 Aligned 4
OpStore %spirv_i_10 %88 Aligned 4
%89 = OpLoad %ulong %spirv_l_2 Aligned 8
%90 = OpLoad %ulong %spirv_l_8 Aligned 8
. . .
```

oneAPI

LevelZero-JNI dispatch

[RUNTIME] Hardware Agnostic Middle-ware APIs

Implementation of the LAUNCH BYTECODE (compilation part)

```
// 1. Build Common Compiler Phase (Sketcher)
Providers providers = spirvBackend.getProviders();
TornadoSuitesProvider suites = spirvBackend.getTornadoSuites();
Sketch sketch = buildSketchForJavaMethod(resolvedJavaMethod, taskMeta, providers, suites);

// 2. Function f: Sketch -> SPIR-V Compiled Code
SPIRVCompilationResult spirvCompilationResult = SPIRVCompiler.compileSketchForDevice(
     sketch,
     compilableTask,
     spirvBackend.getProviders(),
     spirvBackend,
     new EmptyProfiler());

// 3. Install the SPIR-V code into binary and Insert into the TornadoVM JIT-Code Cache
SPIRVDevice spirvDevice = device.getDeviceContext().getDevice();
SPIRVInstalledCode spirvInstalledCode = spirvDevice.installBinary(spirvCompilationResult);
```

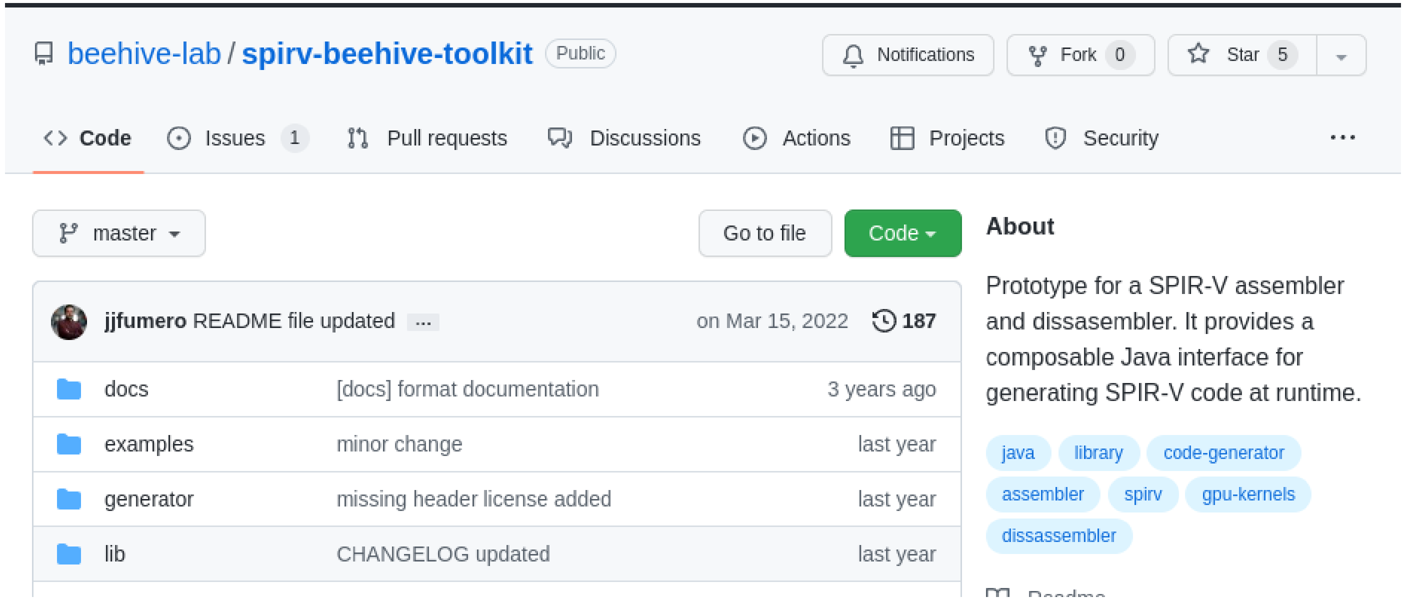
Disclaimer: pseudocode

MANCHESTER 1824
The University of Manchester

SPIR-V Beehive Toolkit for code-gen within TornadoVM

- In-House Java Library for SPIR-V code generation
- Works totally independent from TornadoVM
- It implements **full SPIR-V 1.2**
  - We can sync with SPIR-V 1.5 or any other version quickly
- Plans for open-source it as a stand-alone library

GraalVM → SPIR



https://github.com/beehive-lab/spirv-beehive-toolkit

SPIR-V Beehive Toolkit for code-gen within TornadoVM

MANCHESTER 1824
The University of Manchester

- In-House Java Library for SPIR-V code generation
- Works totally independent from TornadoVM
- It implements **full SPIR-V 1.2**
  - We can sync with SPIR-V 1.5 or any other version quickly
- Plans for open-source it as a stand-alone library

```
// SPIR-V Header
asm.module = new SPIRVModule(
        new SPIRVHeader(
              1,  // Major Version
              2,  // Minor Version
              29, // ID-Generator (new one)
              0,  // Bounds
              0)); // Schema
```

SPIR-V Beehive Toolkit for code-gen within TornadoVM

- In-House Java Library for SPIR-V code generation
- Works totally independent from TornadoVM
- It implements **full SPIR-V 1.2**
  - We can sync with SPIR-V 1.5 or any other version quickly
- Plans for open-source it as a stand-alone library

```
// SPIR-V Header
asm.module = new SPIRVModule(
        new SPIRVHeader(
                1,  // Major Version
                2,  // Minor Version
                32, // ID-Generator (new one)
                0,  // Bounds
                0)); // Schema
```

```
; SPIR-V
; Version: 1.2
; Generator: Khronos; 32
; Bound: 77
; Schema: 0
```

SPIR-V Beehive Toolkit for code-gen within TornadoVM

ADD: a + b

```
SPIRVId add = module.getNextId();
blockScope.add(new SPIRVOpIAdd(
            uint,    // type ID
            add,     // result
            id74,    // a
            id75));  // b
```

```
%add = OpIAdd %uint %74 %75
```

MANCHESTER 1824
The University of Manchester

SPIR-V Beehive Toolkit for code-gen within TornadoVM

MANCHESTER 1824
The University of Manchester

ADD: a + b

```
SPIRVId add = module.getNextId();
blockScope.add(new SPIRVOpIAdd(
            uint,    // type ID
            add,     // result
            id74,    // a
            id75));  // b
```

```
%add = OpIAdd %uint %74 %75
```

Load a[i]

```
SPIRVId idLoad = module.getNextId();
blockScope.add(new SPIRVOpLoad(
      ptrCrossGroupUint,
      idLoad,
      a_addr, // Load A[i]
      new SPIRVOptionalOperand<>(
        SPIRVMemoryAccess.Aligned(
             new SPIRVLiteralInteger(8)))
));
```

```
%idLoad = OpLoad %_ptr_CrossWorkgroup_uint  %addr Aligned 8
```

Standalone library for low-level GPU programming

LevelZero JNI Library for TornadoVM

MANCHESTER 1824
The University of Manchester

- Level Zero Bridge for TornadoVM
  - Since LevelZero is not stable yet, we tried to do a 1-1 mapping between the Java API and C-LevelZero.
  - Easy for us to adapt to new changes
  - In near future, we will leverage this API

```java
// Create the Level Zero Driver
LevelZeroDriver driver = new LevelZeroDriver();
int result =
driver.zeInit(ZeInitFlag.ZE_INIT_FLAG_GPU_ONLY);
LevelZeroUtils.errorLog("zeInit", result);

// Get the number of drivers
int[] numDrivers = new int[1];
result = driver.zeDriverGet(numDrivers, null);
LevelZeroUtils.errorLog("zeDriverGet", result);
```

The Intel Level Zero Spec: https://spec.oneapi.io/level-zero/latest/index.html

LevelZero JNI Library for TornadoVM

The University of Manchester

- Level Zero Bridge for TornadoVM
  - Since LevelZero is not stable, we tried to do a 1-1 mapping between the Java API and C-LevelZero.
  - Easy for us to adapt to new changes
  - In near future, we will leverage this API

```
// Create the Level Zero Driver
LevelZeroDriver driver = new LevelZeroDriver();
int result =
driver.zeInit(ZeInitFlag.ZE_INIT_FLAG_GPU_ONLY);
LevelZeroUtils.errorLog("zeInit", result);

// Get the number of drivers
int[] numDrivers = new int[1];
result = driver.zeDriverGet(numDrivers, null);
LevelZeroUtils.errorLog("zeDriverGet", result);
```

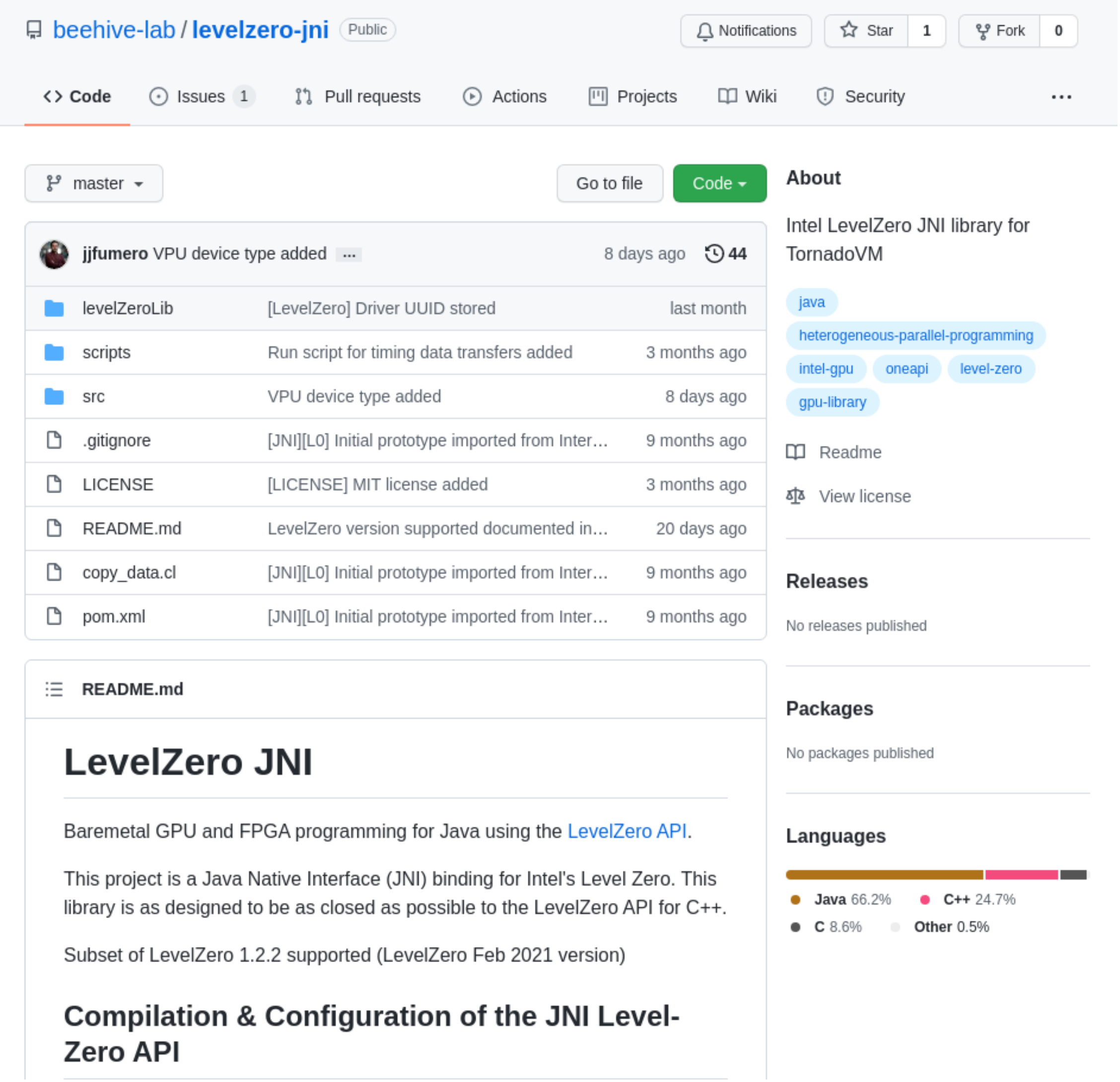
```
// Create buffer
LevelZeroBufferInteger bufferA = new LevelZeroBufferInteger();
// Declare buffer as a shared memory
result = context.zeMemAllocShared(context.getContextHandle(),      // Level Zero Context
                                  deviceMemAllocDesc,              // Device descriptor
                                  hostMemAllocDesc,                // Host Descriptor
                                  bufferSize,                      // Buffer size in Bytes
                                  1,                               // Alignment
                                  device.getDeviceHandlerPtr(),    // Device pointer
                                  bufferA);                        // Buffer to use
LevelZeroUtils.errorLog("zeMemAllocShared", result);
```

MANCHESTER 1824
The University of Manchester

LevelZero JNI Libray for TornadoVM

- This library dispatches SPIR-V kernels
- It does not support full LevelZero, just what we need for TornadoVM, although it could be easy extensible
- It is open source under:
    - **MIT License**

https://github.com/beehive-lab/levelzero-jni/



4. What do we miss from oneAPI?

Brainstorming the Future of oneAPI/LevelZero for Managed Runtime PL

MANCHESTER 1824
The University of Manchester

1. Memory Page Faults/Memory Page migration counters
   - Similar to the NVIDIA NSys Profiler
   - Related issue: https://github.com/oneapi-src/level-zero/issues/100

2. Interaction with the Garbage Collectors (e.g., Java GC)
3. Async Device Exception Handling support
   - E.g., How to handle arithmetic exception in hardware?
4. Features: Device aggregation
   - E.g., Does it make sense to have 2 GPUs acting as 1 big GPU? -> Dynamic kernel dispatch across GPUs using the same system (e.g., Level Zero, SYCL oneAPI, etc)
   - Best device/s mapping (smart device selection mode)

5. Can the Relaxed Limited mode be the default mode?
   - https://github.com/oneapi-src/level-zero/issues/89

6. Improvements in the Kernel Suggest for Group sizes. We see differences in performance between the suggest threads on iGPU vs dGPUs and manual tuning.

7. Use Device Buffers Cached Version by default: https://github.com/intel/compute-runtime/issues/515

GC Issues

The University of Manchester

[ISSUE] https://github.com/gpu/JOCL/issues/7

- https://github.com/gpu/JOCL/commit/d01208c9687dae6015047d4cd55c16f65dbcc6da
- https://github.com/gpu/JOCL/commit/5c6e44f8dd6a84d539ec8cf2b489f707e12f3d07

Thank you!

- Partially supported by the EU Horizon 2020:
    - ELEGANT 957286
    - AERO 101092850
    - INCODE 101093069
    - TANGO 101070052
    - ENCRYPT 101070670
    - E2Data 780245
    - ACTICLOUD 732366
- Partially supported by Intel Grant

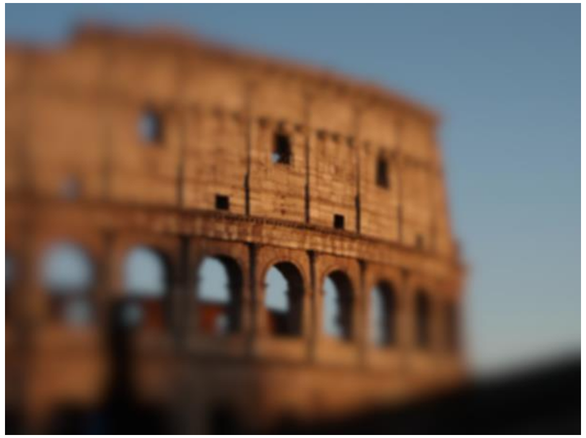
Juan Fumero: juan.fumero@manchester.ac.uk

@snatverk

Discussions

Backup Slides

Example – Blur Filter - Let's run it

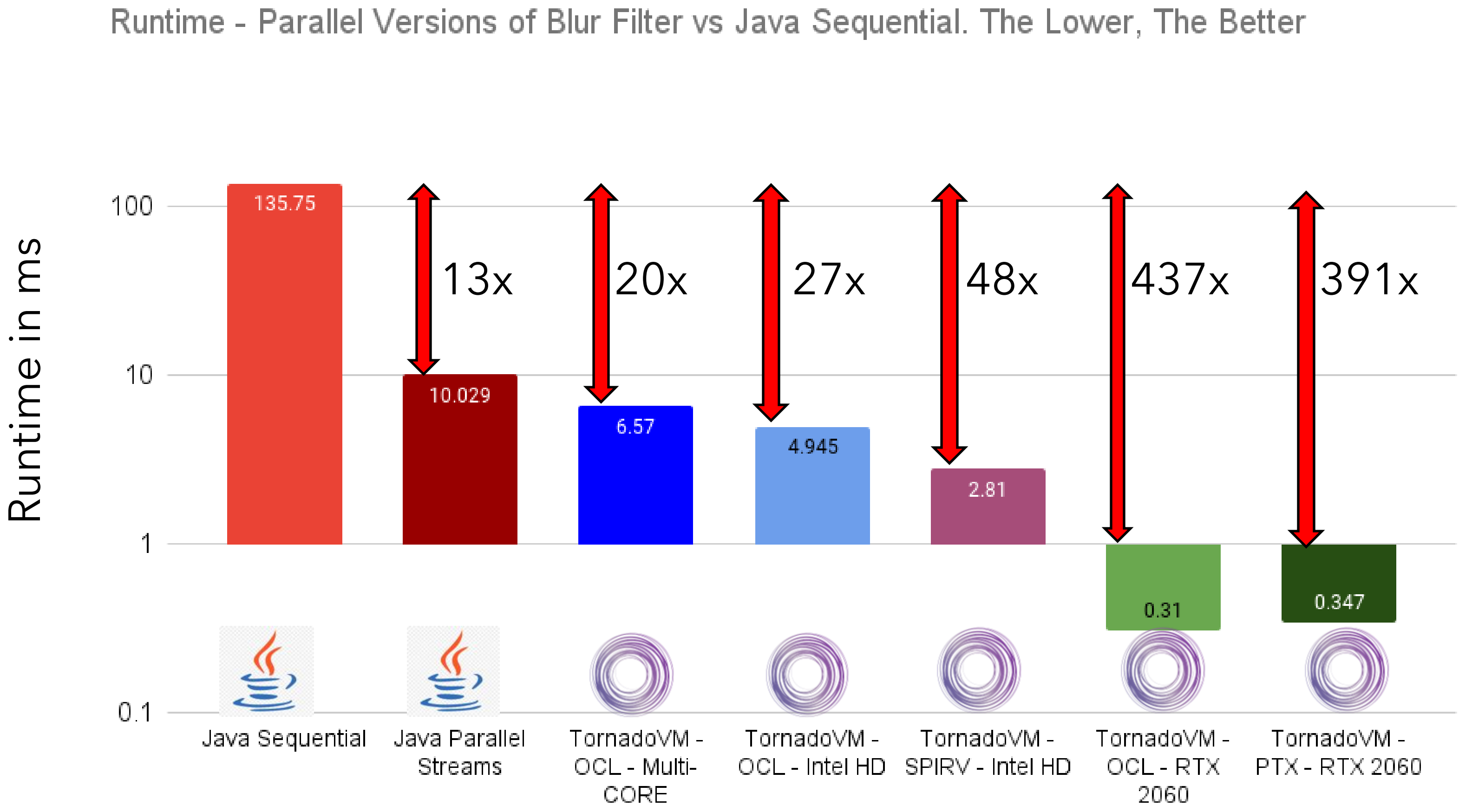


```
$ tornado \
 -cp target/tornadovm-examples-1.0-SNAPSHOT.jar \
 io.github.jjfumero.BlurFilter --tornado
```

The `tornado` command is an alias to `java` and all flags for TornadoVM.

https://github.com/jjfumero/tornadovm-examples

Blur Filter Performance (on my laptop)

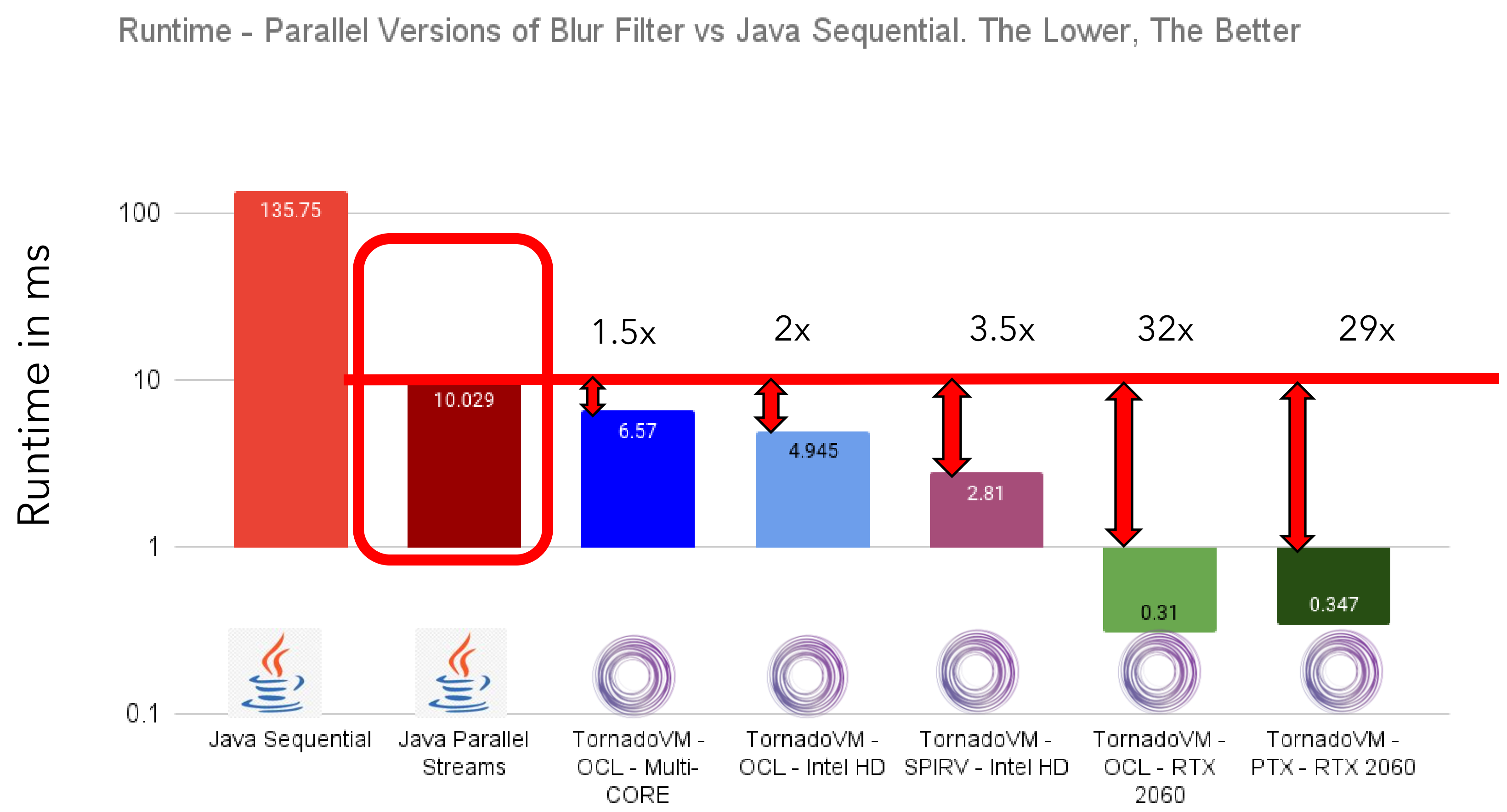


* Running 5K x 4K image
* JDK 17
* TornadoVM v0.14-dev
* Intel Core i9 (16 cores)
* Intel HD 630
* GPU NVIDIA 2060 Mobile

Up to 437x when running with TornadoVM on a GPU

Blur Filter Performance (on my laptop)

* Running 5K x 4K image
* JDK 17
* TornadoVM v0.14-dev
* Intel Core i9 (16 cores)
* Intel HD 630
* GPU NVIDIA 2060 Mobile



Up to 30x compared to Java Multi-Thread Stream (16 cores) when running on a GPU

Where is code executed?

The University of Manchester

```
public class Foo {

   public void methodToAccelere01( … ) {   … }

   public void methodToAccelere02( … ) {   … }

   public void methodToAccelere03( … ) {   … }


   public void runWithTornadoVM() {

      TaskGraph ts = new TaskGraph("foo")
       .transferToDevice(data... )
         .task("m1", this::methodToAccelerate01,…)
         .task("m2", this::methodToAccelerate02,…)
         .task("m3", this::methodToAccelerate03,…)
      .transferToHost(output)
     executionPlan = new TornadoExecutionPlan(ts.snaphot());
     executionPlan.execute();

}}
```

Single Source Property:
GPU/FPGA Kernels and host code in the same source file expressed in the same programming language

Where is code executed?

MANCHESTER 1824
The University of Manchester

```java
public class Foo {

    public void methodToAccelere01( … ) {   … }

    public void methodToAccelere02( … ) {   … }

    public void methodToAccelere03( … ) {   … }

    public void runWithTornadoVM() {

     TaskGraph ts = new TaskGraph("foo")
        .transferToDevice(data... )
          .task("m1", this::methodToAccelerate01,…)
          .task("m2", this::methodToAccelerate02,…)
          .task("m3", this::methodToAccelerate03,…)
       .transferToHost(output)
      executionPlan = new TornadoExecutionPlan(ts.snaphot());
      executionPlan.execute();
}}
```

Java

NVIDIA CUDA

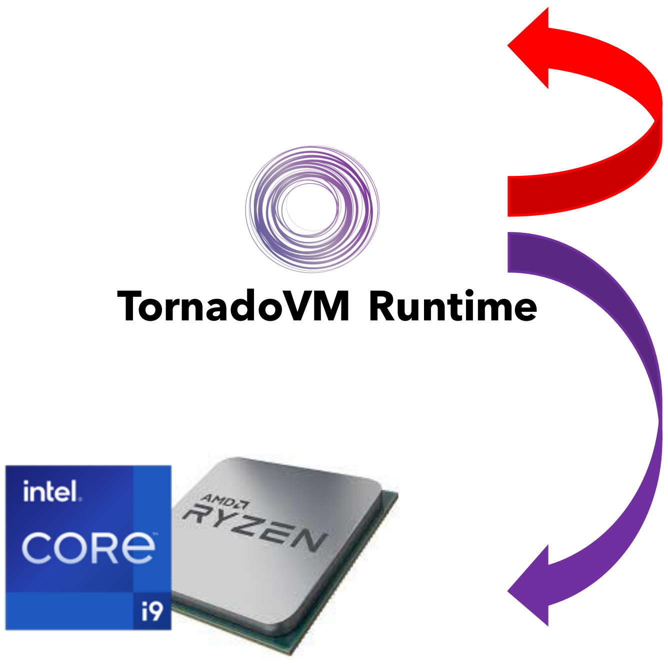
OpenCL

SPIR

Single Source Property:
GPU/FPGA Kernels and host code in the same source file expressed in the same programming language

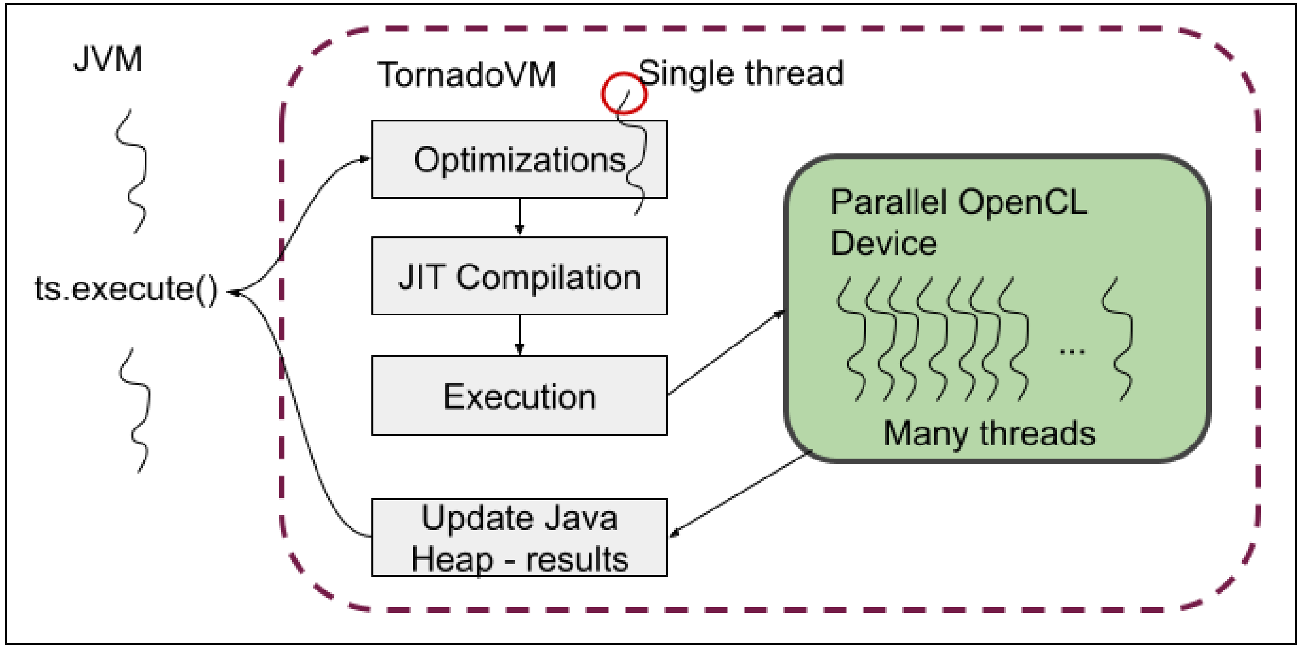
Where is code executed?

```java
public class Foo {
    public void methodToAccelere01( … ) {   … }

    public void methodToAccelere02( … ) {   … }

    public void methodToAccelere03( … ) {   … }

    public void runWithTornadoVM() {
      TaskGraph ts = new TaskGraph("foo")
        .transferToDevice(data... )
          .task("m1", this::methodToAccelerate01,…)
          .task("m2", this::methodToAccelerate02,…)
          .task("m3", this::methodToAccelerate03,…)
       .transferToHost(output)
      executionPlan = new TornadoExecutionPlan(ts.snaphot());
      executionPlan.execute();
}}
```

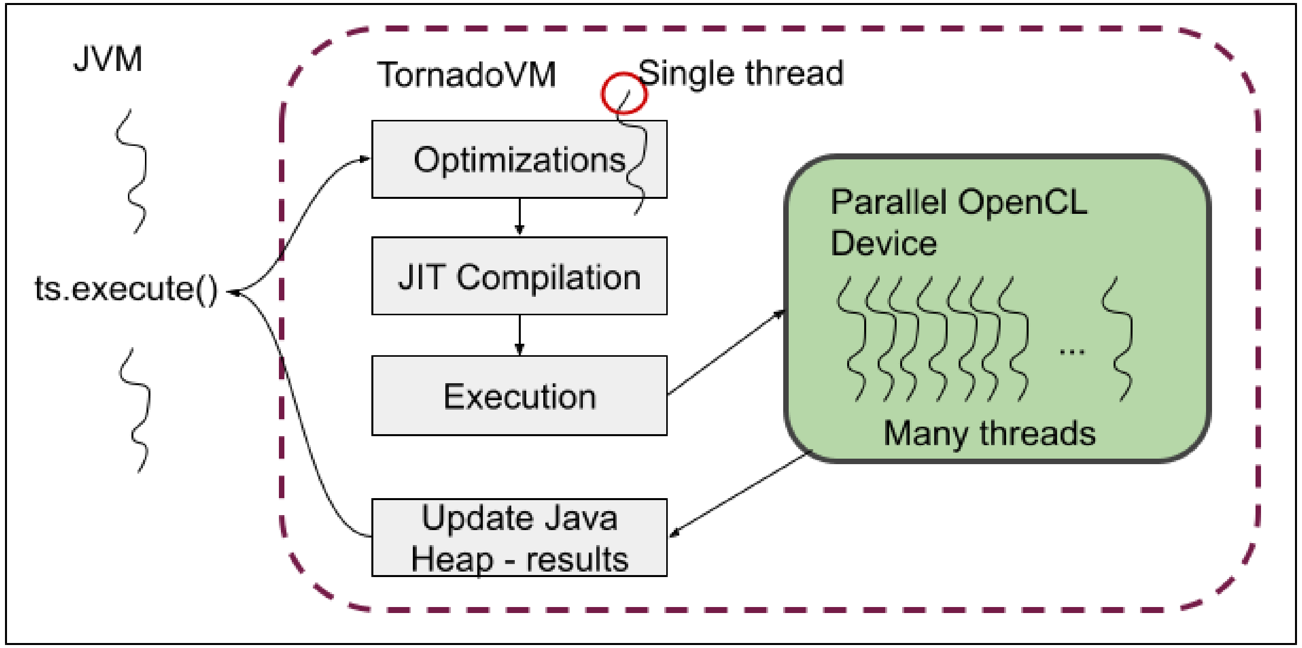


Single Source Property:
GPU/FPGA Kernels and host code in the same source file expressed in the same programming language

How TornadoVM launches Java kernels on Parallel Hardware?



How TornadoVM launches Java kernels on Parallel Hardware?




```java
void blurFilter(. . . ) {

for (@Parallel int r = 0; r < numRows; r++) {
  for (@Parallel int c = 0; c < numCols; c++)
{

    computeFilter(. . . );

  }
}
```

Range of NxM threads
2D (numRow, numColumns)
Each thread computes the body of the parallel loop

Understanding when to use the Parallel Loop API

Pros

- Annotations of (maybe existing) **sequential code**
- Fast development at reasonable performance
- Suitable for **non-experts** on heterogeneous programming
- No hardware knowledge

Cons

- Limited in the number of parallel patterns to support (e.g., scan)
- Lack of low-level control of the hardware
- Hard to port existing parallel code (OpenCL and CUDA)

Understanding when to use the Parallel Loop API

Pros

- Annotations of (maybe existing) **sequential code**
- Fast development at reasonable performance
- Suitable for **non-experts** on heterogeneous programming
- No hardware knowledge

Introduction of a second API -> **Kernel API**

Cons

- Limited in the number of parallel patterns to support (e.g., scan)
- Lack of low-level control of the hardware
- Hard to port existing parallel code (OpenCL and CUDA)

Blur Filter using the Kernel API

MANCHESTER
1824
The University of Manchester

```
void blurFilter(int[] channel, int[] channelBlurred,
                final int numRows, final int numCols,
                float[] filter, final int filterWidth) {
for (@Parallel int r = 0; r < numRows; r++) {
  for (@Parallel int c = 0; c < numCols; c++) {

    computeFilter(. . . );

  }
}
```

Parallel Loop API

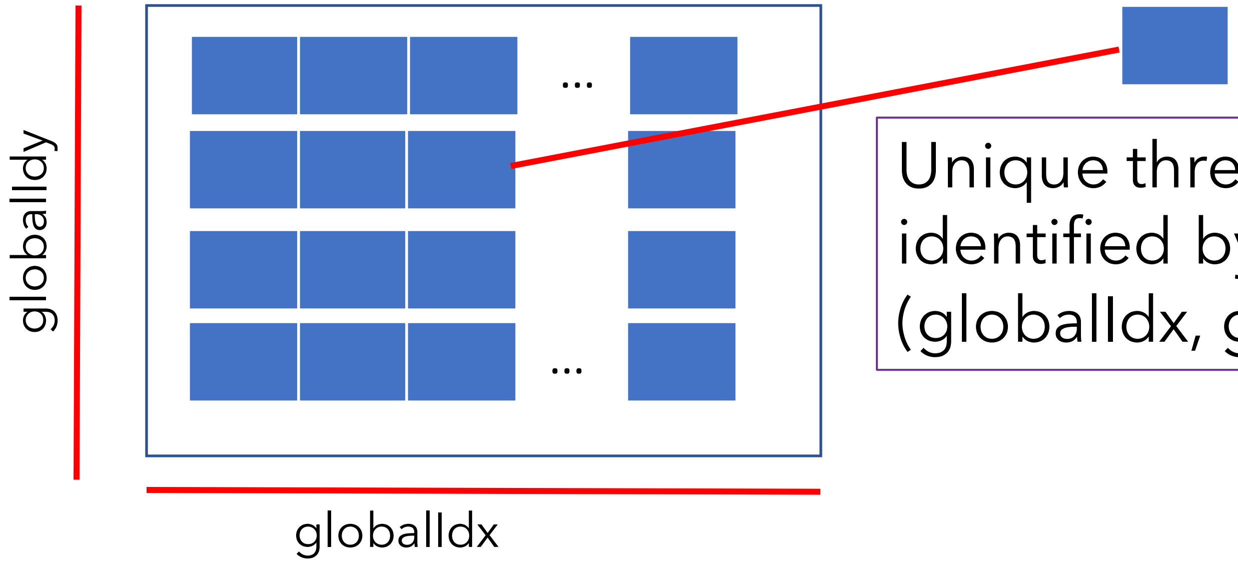
Blur Filter using the Kernel API

```
void blurFilter(int[] channel, int[] channelBlurred,
                final int numRows, final int numCols,
                float[] filter, final int filterWidth) {
for (@Parallel int r = 0; r < numRows; r++) {
  for (@Parallel int c = 0; c < numCols; c++) {

    computeFilter(. . . );

  }
}
```

Parallel Loop API

```
void blurFilter(int[] channel, int[] channelBlurred,
                final int numRows, final int numCols,
                float[] filter, final int filterWidth,
                KernelContext context) {
int r = context.globalIdx;
int c = context.globalIdy;
computeFilter( . . . );
```

Kernel API

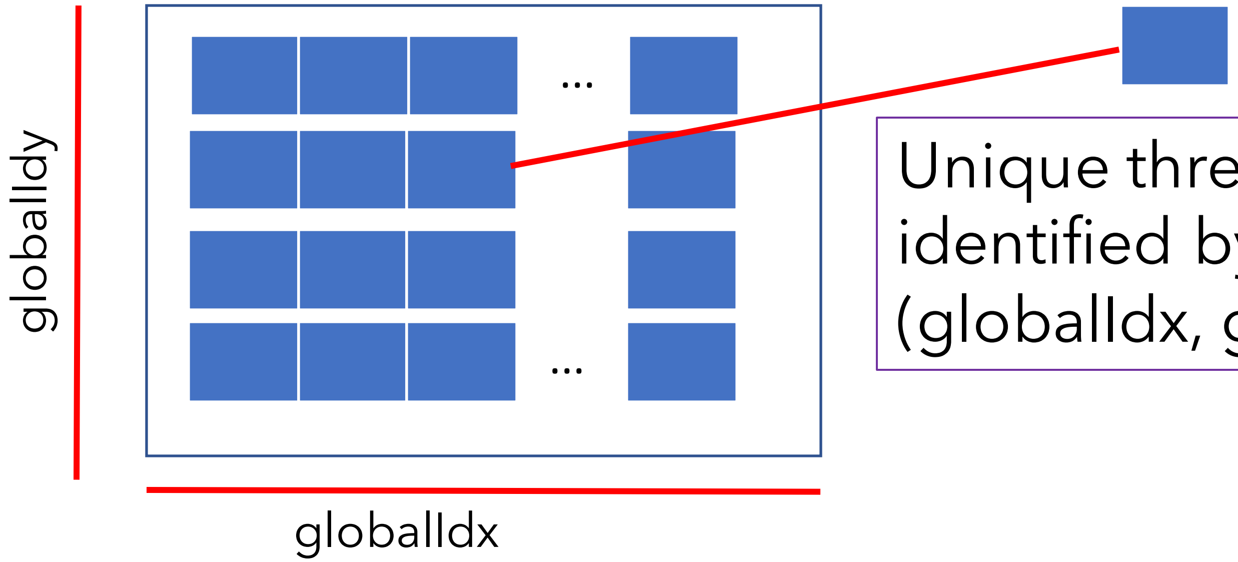


Unique thread ID identified by (globalIdx, globalIdy)

Tuning the Amount of Threads to Run

```
void blurFilter(int[] channel, int[] channelBlurred,
                final int numRows, final int numCols,
                float[] filter, final int filterWidth,
                KernelContext context) {
int r = context.globalIdx;
int c = context.globalIdy;
computeFilter( . . . );
```

Kernel API



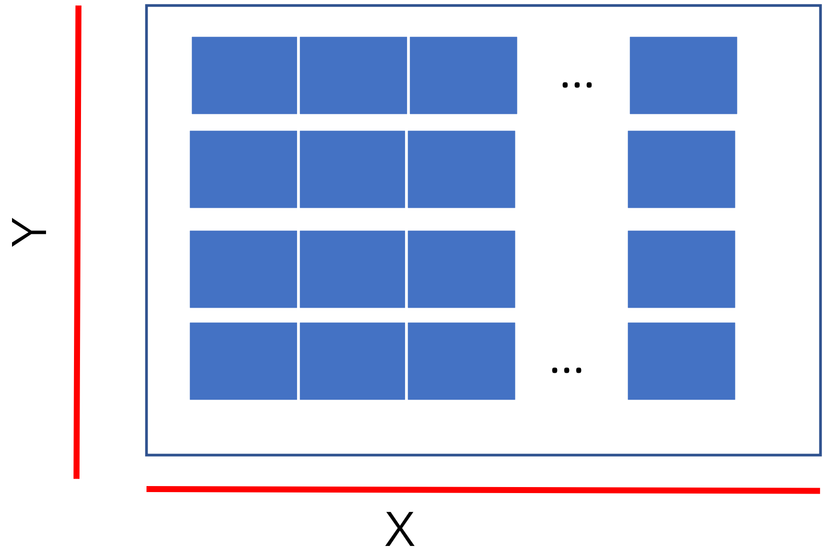
Unique thread ID identified by (globalIdx, globalIdy)

Tuning the Amount of Threads to Run

MANCHESTER 1824
The University of Manchester

```java
void blurFilter(int[] channel, int[] channelBlurred,
                final int numRows, final int numCols,
                float[] filter, final int filterWidth,
                KernelContext context) {
   int r = context.globalIdx;
   int c = context.globalIdy;

   computeFilter( . . . );

}
```

```java
  // Create a 2D Grid

WorkerGrid workerGrid = new WorkerGrid2D(X, Y);
GridScheduler grid = new GridScheduler();

grid.setWorkerGrid("blur.redFilter", workerGrid);

executionPlan.withGridScheduler(grid).execute()
```

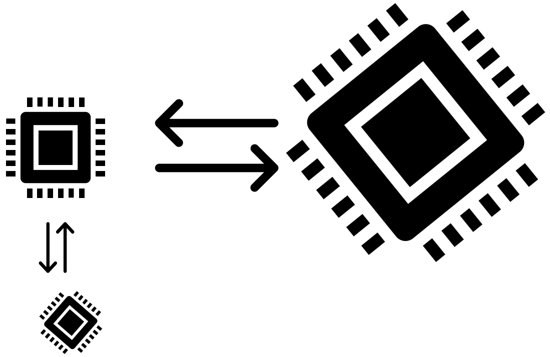


Thread Selection is fully automatic using the Parallel Loop API

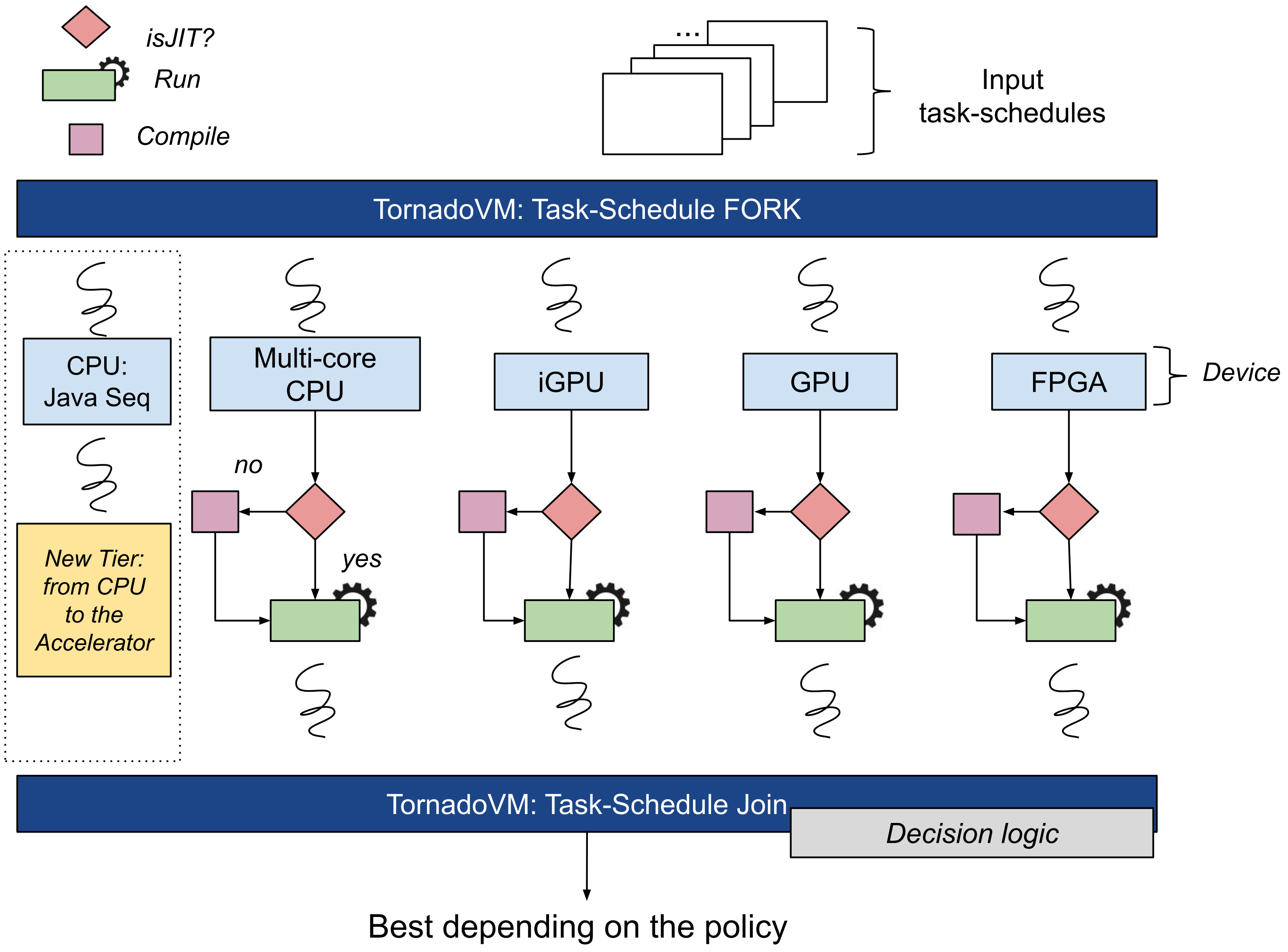
Using the Kernel API is a requirement

Key Features
&
what is coming
with TornadoVM

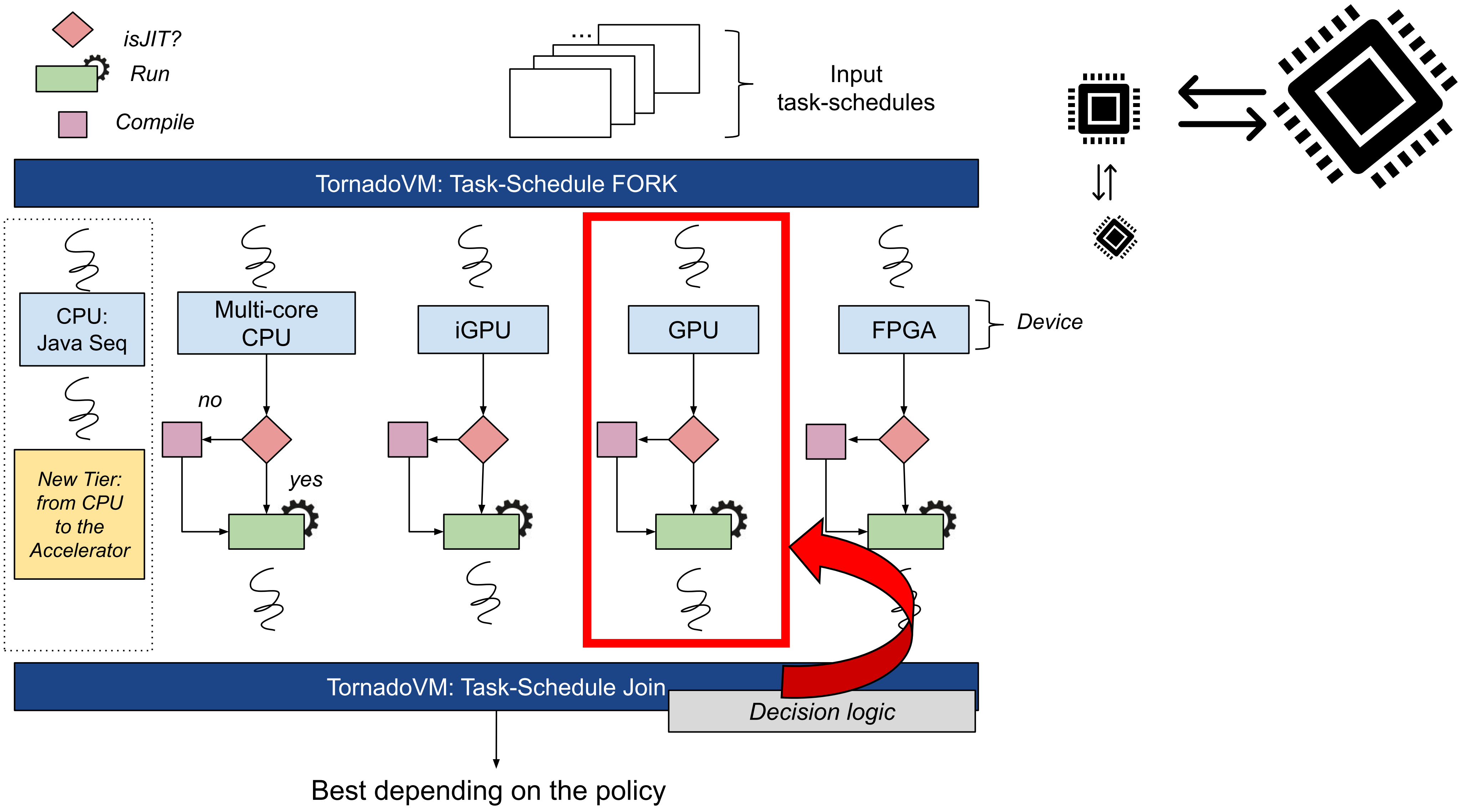
1) Live Task Migration – It's Upstream



Live Task Migration



Live Task Migration



Key Idea of TornadoVM

TornadoVM is not a substitution of the usual Java execution and Java compilers (e.g., C1/C2, Graal), but rather a complement to achieve higher performance for specific types of applications

Our vision is Java to automatically migrate from CPU to GPU when performance can be increased.

2) Batch [upstream] and Parallel Batch Processing [experimental]

MANCHESTER 1824
The University of Manchester

Input Java user-code

```java
class Compute {
  public static void add(double[] a, double[] b,
  double[] c) {
   for (@Parallel int i = 0; i < c.length; i++)
      c[i] = a[i] + b[i];
   }
  }
}
```

```java
// 16GB data
double[] a = new double[2000000000];
double[] b = new double[2000000000];
double[] c = new double[2000000000];
TaskSchedule ts = new TaskSchedule("s0");

ts.batch("300MB")
  .task(Compute::add, a, b, c)
  .streamOut(c)
  .execute();
```

Tornado VM

```
vm: BEGIN
vm: COPY_IN bytes=300000000, offset=0
vm: COPY_IN bytes=300000000, offset=0
vm: ALLOCATE bytes=300000000
vm: LAUNCH s0.t0 threads=37500000, offset=0
vm: STREAM_OUT bytes=300000000, offset=0
vm: COPY_IN bytes=300000000, offset=300000000
vm: COPY_IN bytes=300000000, offset=300000000
vm: ALLOCATE bytes=300000000
vm: LAUNCH task s0.t0 threads=37500000, offset=300000000
vm: STREAM_OUT bytes=300000000, offset=300000000
vm: …
vm: …
vm: STREAM_OUT_BLOCKING bytes=100000000, offset=1500000000
vm: END
```

Easy to orchestrate heterogeneous execution

Parallel Batch Processing

MANCHESTER 1824
The University of Manchester

- Batch Processing is already public and upstream!
- We are working towards facilitating parallel batch processing

Enabling Pipeline Parallelism in Heterogeneous Managed Runtime Environments via Batch Processing

Florin Blanaru
florin.blanaru@manchester.ac.uk
The University of Manchester
Manchester, United Kingdom

Athanasios Stratikopoulos
athanasios.stratikopoulos@manchester.ac.uk
The University of Manchester
Manchester, United Kingdom

Juan Fumero
juan.fumero@manchester.ac.uk
The University of Manchester
Manchester, United Kingdom

Christos Kotselidis
christos.kotselidis@manchester.ac.uk
The University of Manchester
Manchester, United Kingdom

Abstract

During the last decade, managed runtime systems have been constantly evolving to become capable of exploiting underlying hardware accelerators, such as GPUs and FPGAs. Regardless of the programming language and their corresponding runtime systems, the majority of the work has been focusing on the compiler front trying to tackle the challenging task of how to enable just-in-time compilation and execution of arbitrary code segments on various accelerators. Besides this challenging task, another important aspect that defines both functional correctness and performance of managed runtime systems is that of automatic memory management. Although automatic memory management improves productivity by abstracting away memory allocation and maintenance, it hinders the capability of using specific memory regions, such as pinned memory, in order to perform data transfer times between the CPU and hardware accelerators.

In this paper, we introduce and evaluate a series of memory optimizations specifically tailored for heterogeneous managed runtime systems. In particular, we propose: (i) transparent and automatic "parallel batch processing" for overlapping data transfers and computation between the host and hardware accelerators in order to enable pipeline parallelism, and (ii) "off-heap pinned memory" in combination with parallel state-of-the-art open-source TornadoVM and their combination can lead up to 2.5*x* end-to-end performance speedup against sequential batch processing.

CCS Concepts: • **Software and its engineering → Runtime environments**; • **Computing methodologies → Parallel programming languages**; • **Computer systems organization → Single instruction, multiple data**; • **General and reference → Performance**.

Keywords: Data Transfers, GPUs, Heterogeneous Architectures, Memory Management, Optimizations, Virtual Machines

ACM Reference Format:
Florin Blanaru, Athanasios Stratikopoulos, Juan Fumero, and Christos Kotselidis. 2022. Enabling Pipeline Parallelism in Heterogeneous Managed Runtime Environments via Batch Processing. In *Proceedings of the 18th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '22), March 1, 2022, Virtual, Switzerland*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3516807.3516821

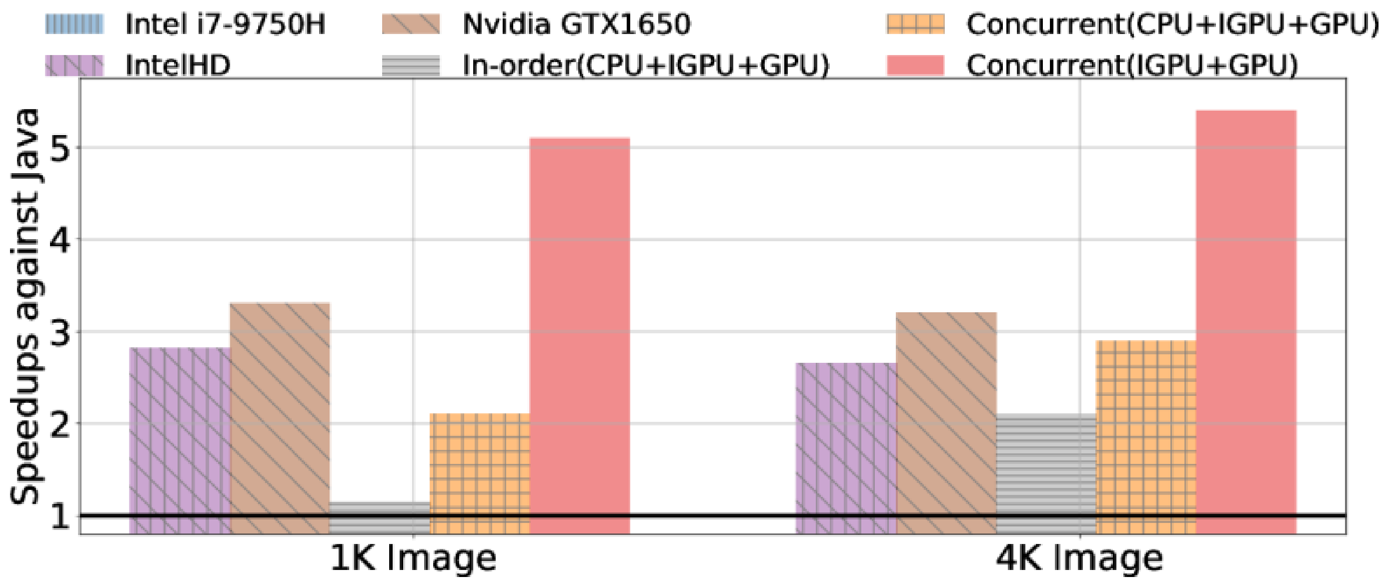
1 Introduction

Following the development of heterogeneous programming

More Info [Blanaru et al., VEE'22]

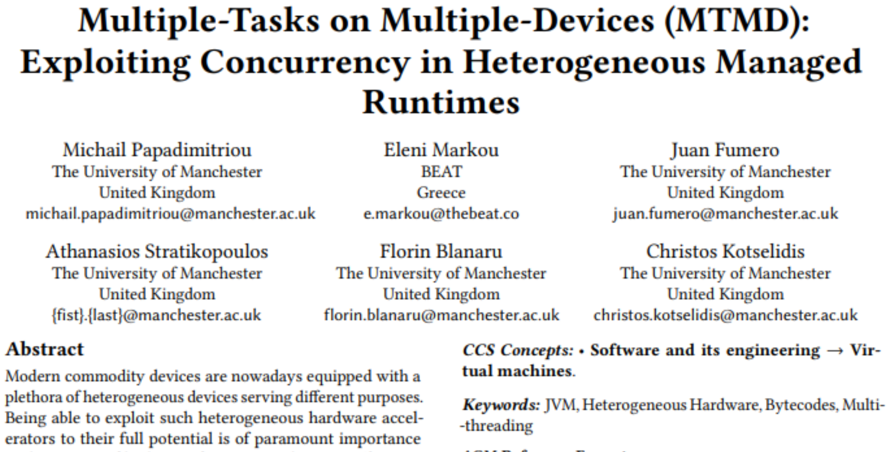
3) Multi-Device Execution [experimental]

MANCHESTER 1824
The University of Manchester



Multi-GPU Blur Filter is > 5x faster compared to a single GPU

Transparent Multi-device Selection



Multiple-Tasks on Multiple-Devices (MTMD): Exploiting Concurrency in Heterogeneous Managed Runtimes

Michail Papadimitriou, The University of Manchester, United Kingdom, michail.papadimitriou@manchester.ac.uk

Eleni Markou, BEAT, Greece, e.markou@thebeat.co

Juan Fumero, The University of Manchester, United Kingdom, juan.fumero@manchester.ac.uk

Athanasios Stratikopoulos, The University of Manchester, United Kingdom, {fist}.{last}@manchester.ac.uk

Florin Blanaru, The University of Manchester, United Kingdom, florin.blanaru@manchester.ac.uk

Christos Kotselidis, The University of Manchester, United Kingdom, christos.kotselidis@manchester.ac.uk

Abstract

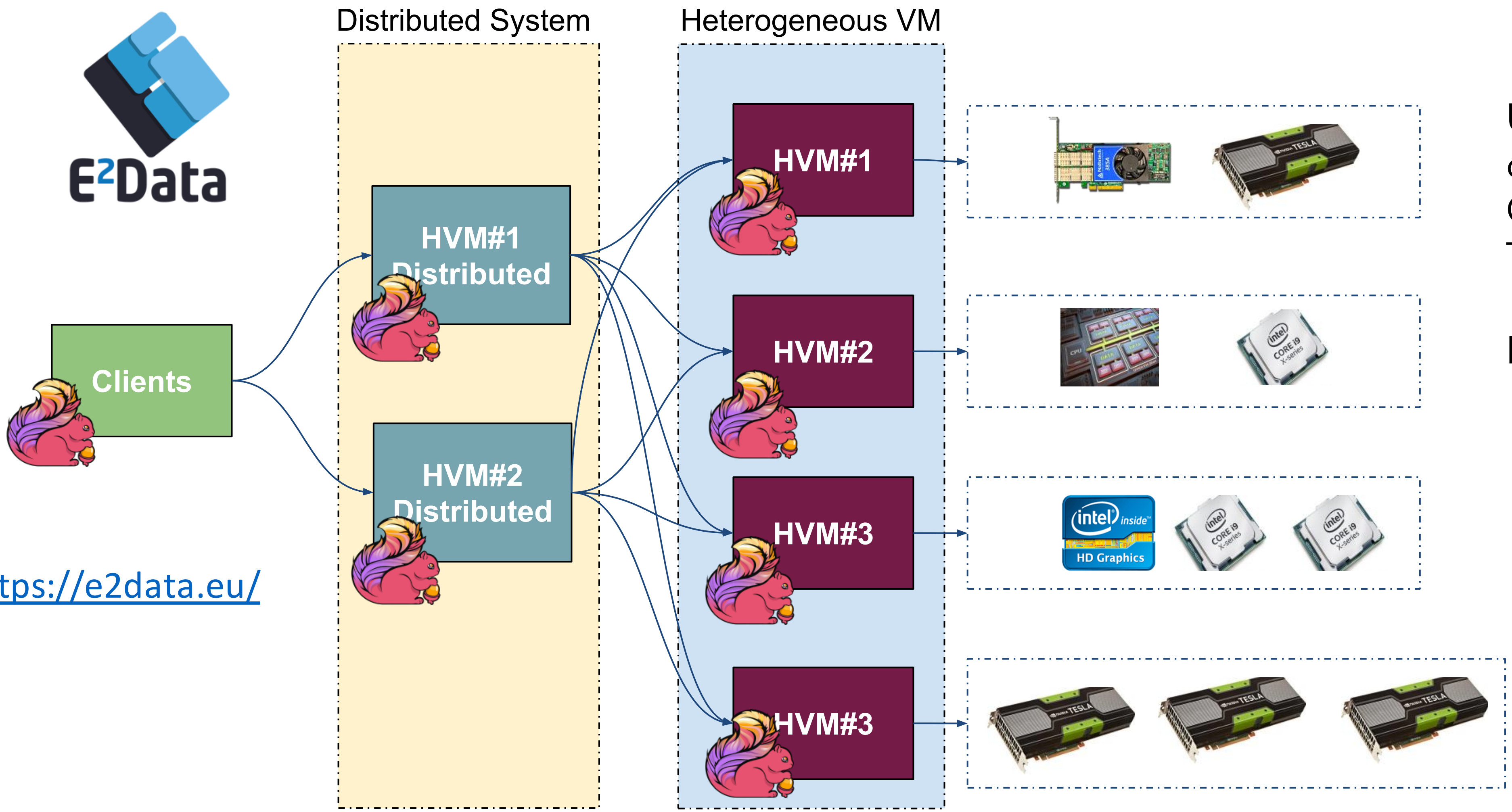
Modern commodity devices are nowadays equipped with a plethora of heterogeneous devices serving different purposes. Being able to exploit such heterogeneous hardware accelerators to their full potential is of paramount importance

CCS Concepts: • **Software and its engineering** → **Virtual machines**.

Keywords: JVM, Heterogeneous Hardware, Bytecodes, Multi-threading

More Details:
[Papadimitriou et al, VEE'21]

Integration with Big Data Platforms (e.g., Flink) [Experimental]

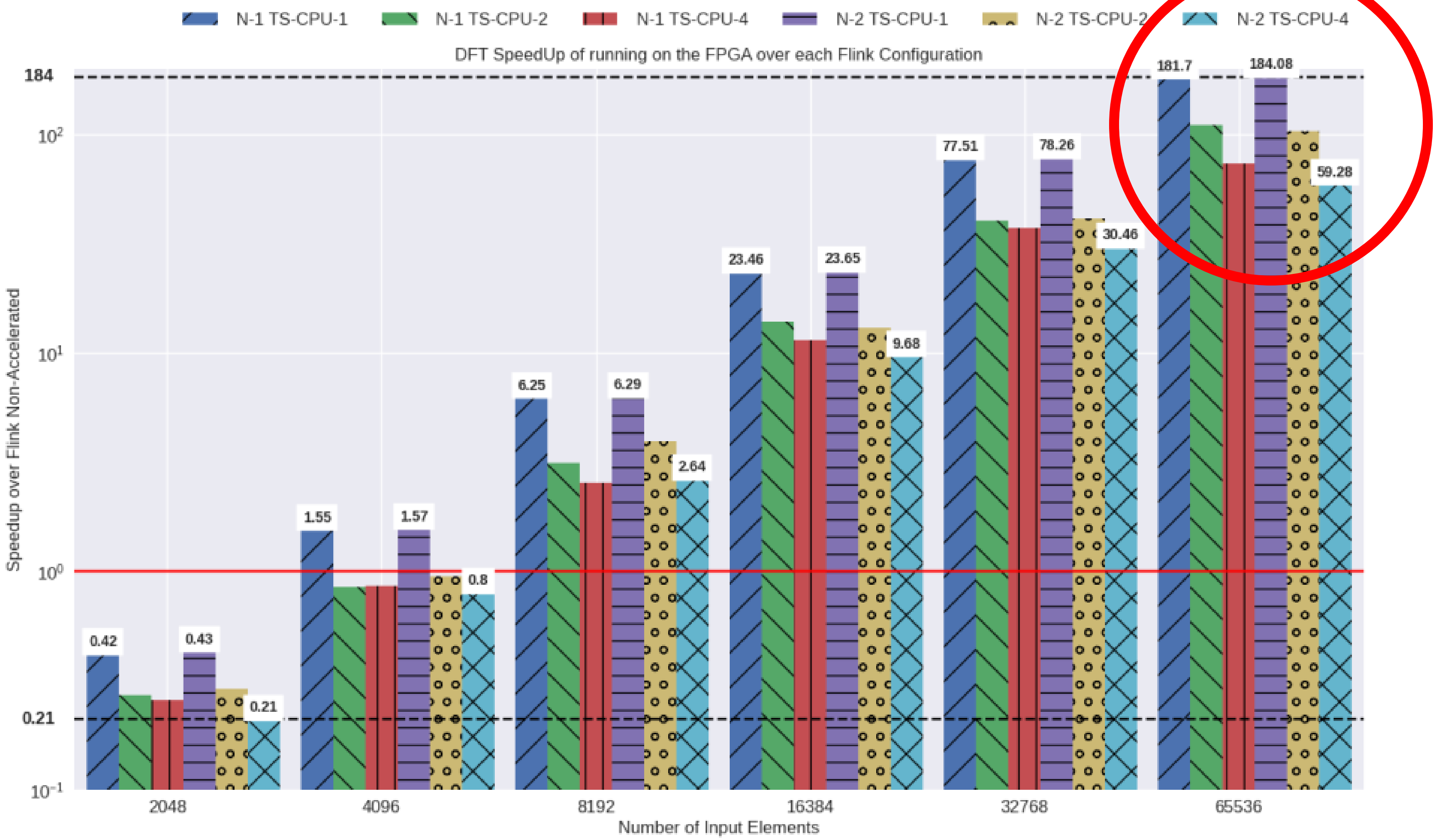


https://e2data.eu/

Unmodified Flink code accelerated on GPUs and FPGAs with TornadoVM

Maria Xekalaki's PhD

Preliminary Results, Running Flink on FPGAs using TornadoVM



DFT Application on FPGA

> 180x compared to Flink 2 nodes with 8 cores each running on an Intel FPGA

More Information Coming Soon!