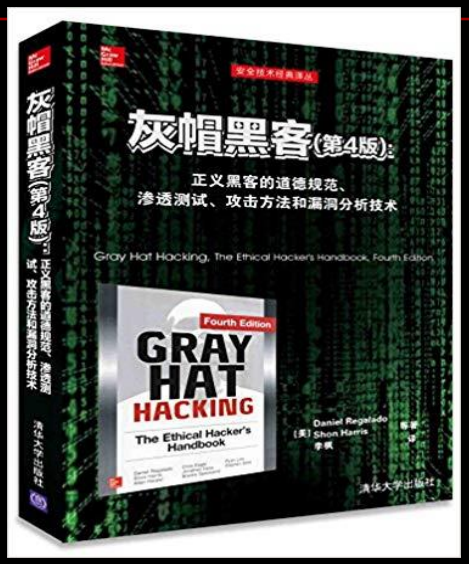
OSDT China 2021

GraalVM for Heterogeneous Parallel Computing

Feng Li (李枫)
hkli2013@126.com
Dec 19，2021

Who Am I

- The main translator of the book «Gray Hat Hacking The Ethical Hacker's Handbook, Fourth Edition» (ISBN：9787302428671) & «Linux Hardening in Hostile Networks, First Edition» (ISBN: 9787115544384)



- Pure software developmement for ~15 years
- Actively participate in various activities of the open source community
- https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/Conf
  https://github.com/XianBeiTuoBaFeng2015/MySlides/tree/master/LTS
- Recently, focus on infrastructure of Cloud/Edge Computing, AI, Virtualization, Program Runtimes, Network, 5G, RISC-V, EDA…

Agenda

I. GraalVM

- Introduction
- Recent changes
- Automated Vectorization
- Project Beehive
- Testbed

II. TornadoVM

- Architecture & Design
- Acceleration of JVM Apps
- Speed up data processing
- Mambo and ProtonVM
- Practice

III. GraalVM as a generic platform for HPC

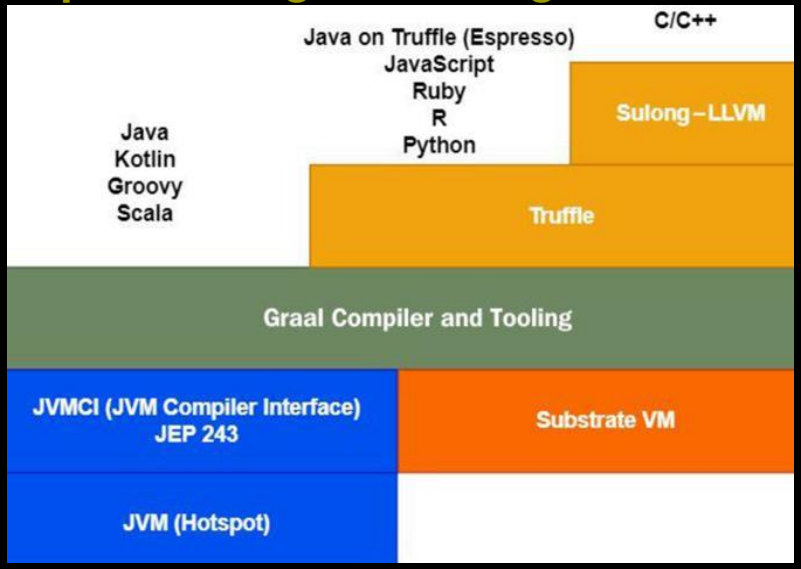
- Smart Runtime
- Support for OpenCL 3.x and SPIR-V backend
- Common Runtime for FOSS EDA

IV Wrap-up

I. GraalVM

1) Introduction

- https://en.wikipedia.org/wiki/GraalVM
- https://www.graalvm.org/

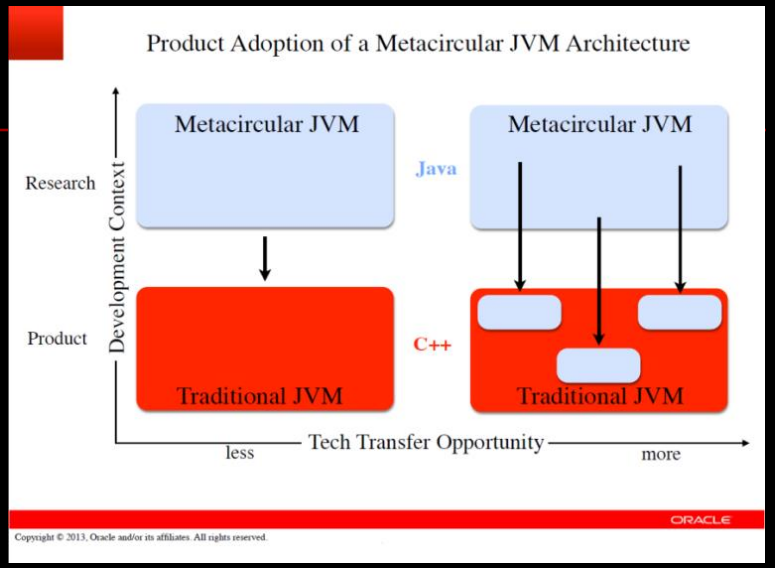


Source: https://static.packt-cdn.com/downloads/9781800564909_ColorImages.pdf

- A Universal High-Performance, Cloud Native, Polyglot VM
- A meta-runtime for Language-Level Virtualization
- Base on OpenJDK 8, 11, 16, and 17 with JVMCI support.
- https://github.com/graalvm
- https://github.com/oracle/graal

Meta-Circular

- MaxineVM → GraalVM



Source: https://chrisseaton.com/truffleruby/jokerconf17/

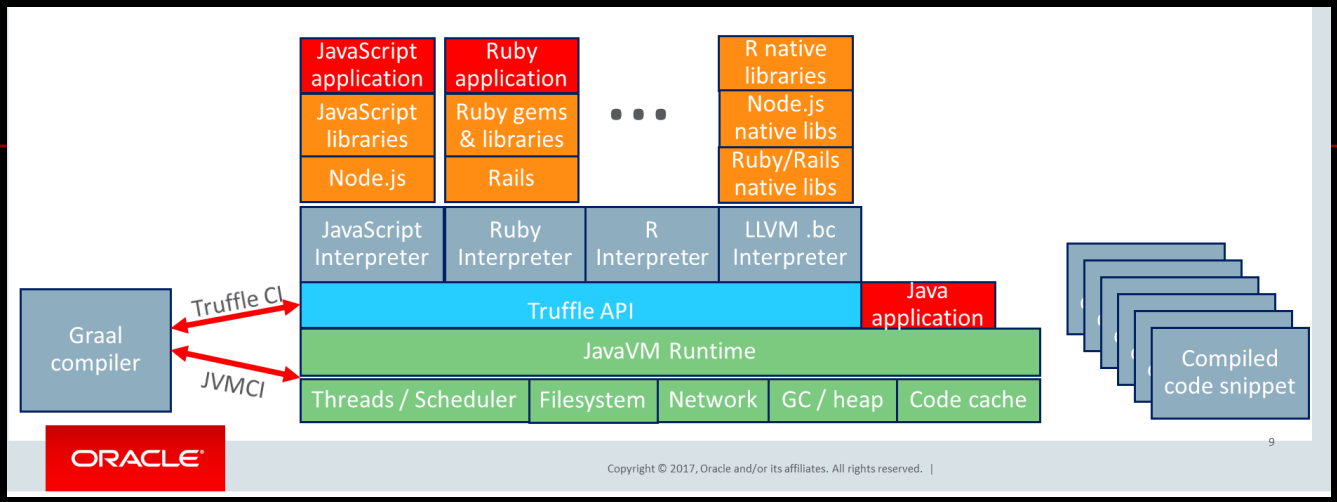
- https://en.wikipedia.org/wiki/Maxine_Virtual_Machine
- https://github.com/beehive-lab/Maxine-VM
- https://maxine-vm.readthedocs.io/en/stable/
- …

Academic Research

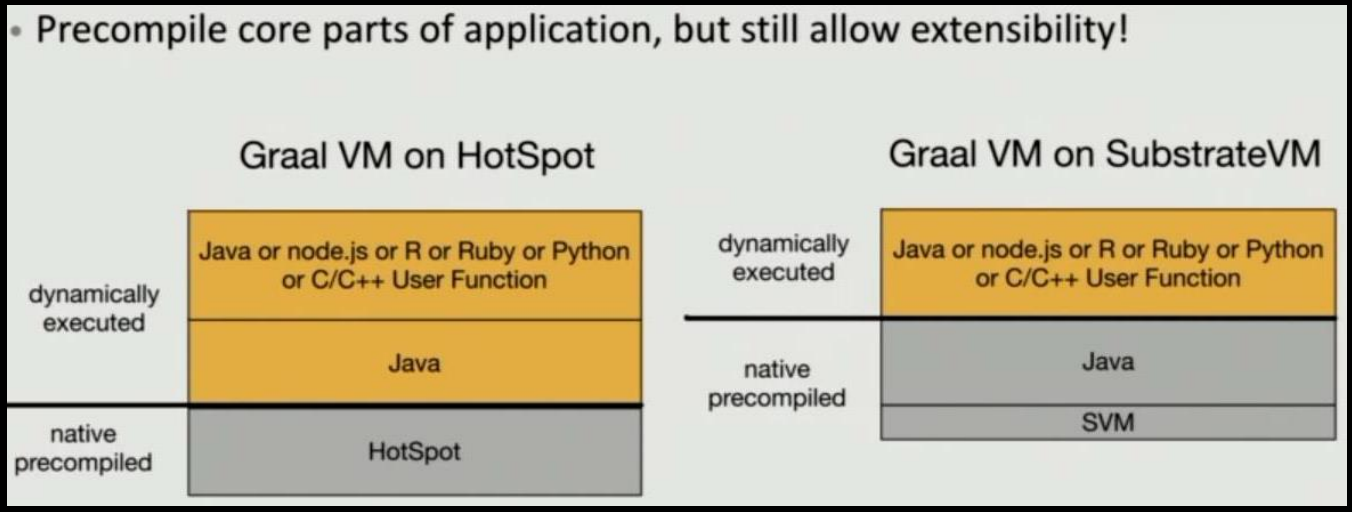
- **JKU**
  http://ssw.jku.at/
  https://ssw.jku.at/Research/Projects/JVM/Graal.html
  https://ssw.jku.at/Research/Projects/JVM/Truffle.html
  …
- **Oracle Labs**
  https://labs.oracle.com
  Oracle Labs Zurich(Thomas Wuerthinger…)
  …
- **APT**
  The Advanced Processor Technologies Research Group.
  http://apt.cs.manchester.ac.uk/
  …
- **…**

Architecture

- A hybrid of static & dynamic runtimes

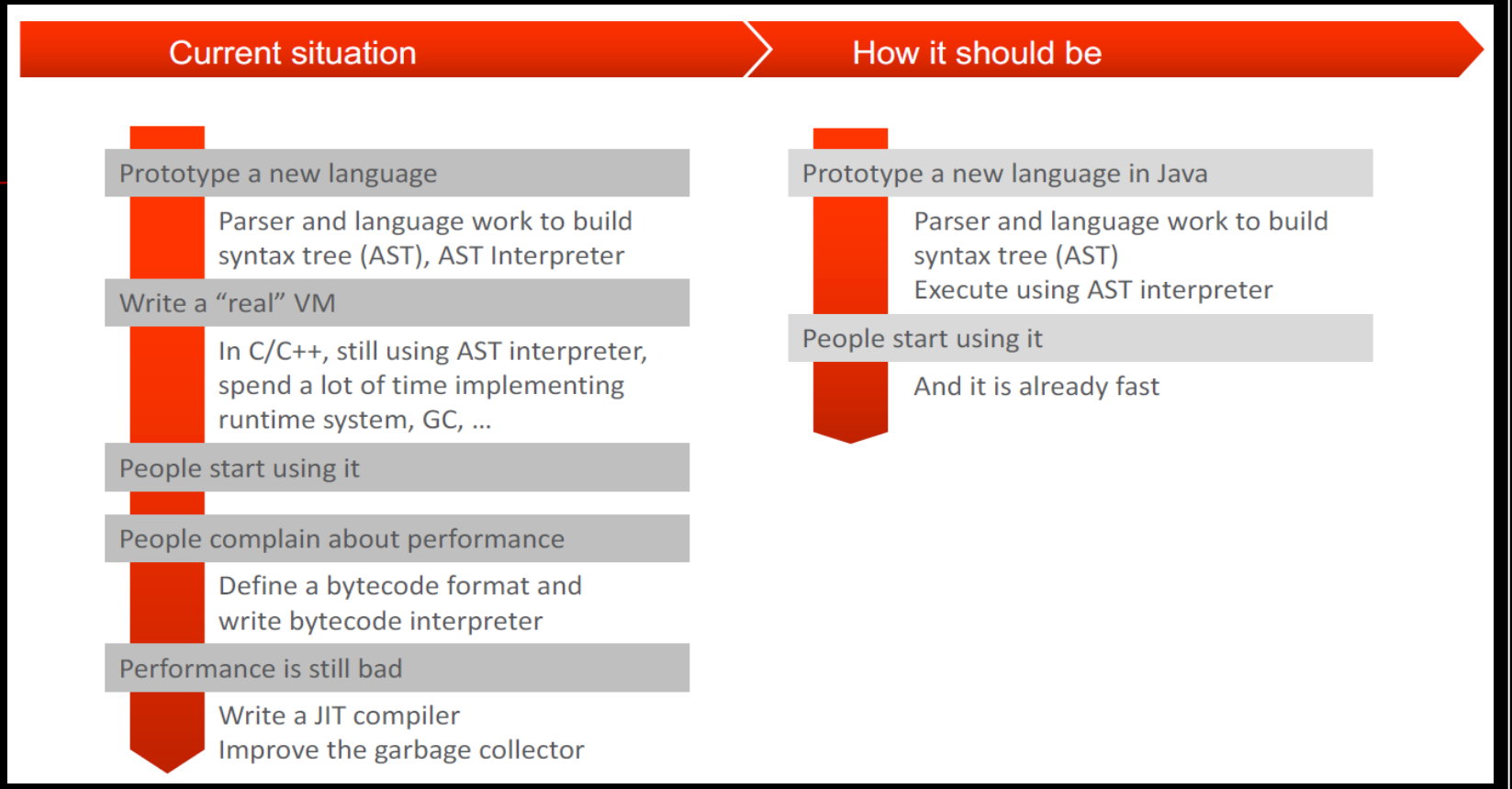


Source: https://ics.psu.edu/wp-content/uploads/2017/02/GraalVM-PSU.pptx



Source: "Adopting Java for the Serverless world", Vadym Kazulkin, JUG London 2020.

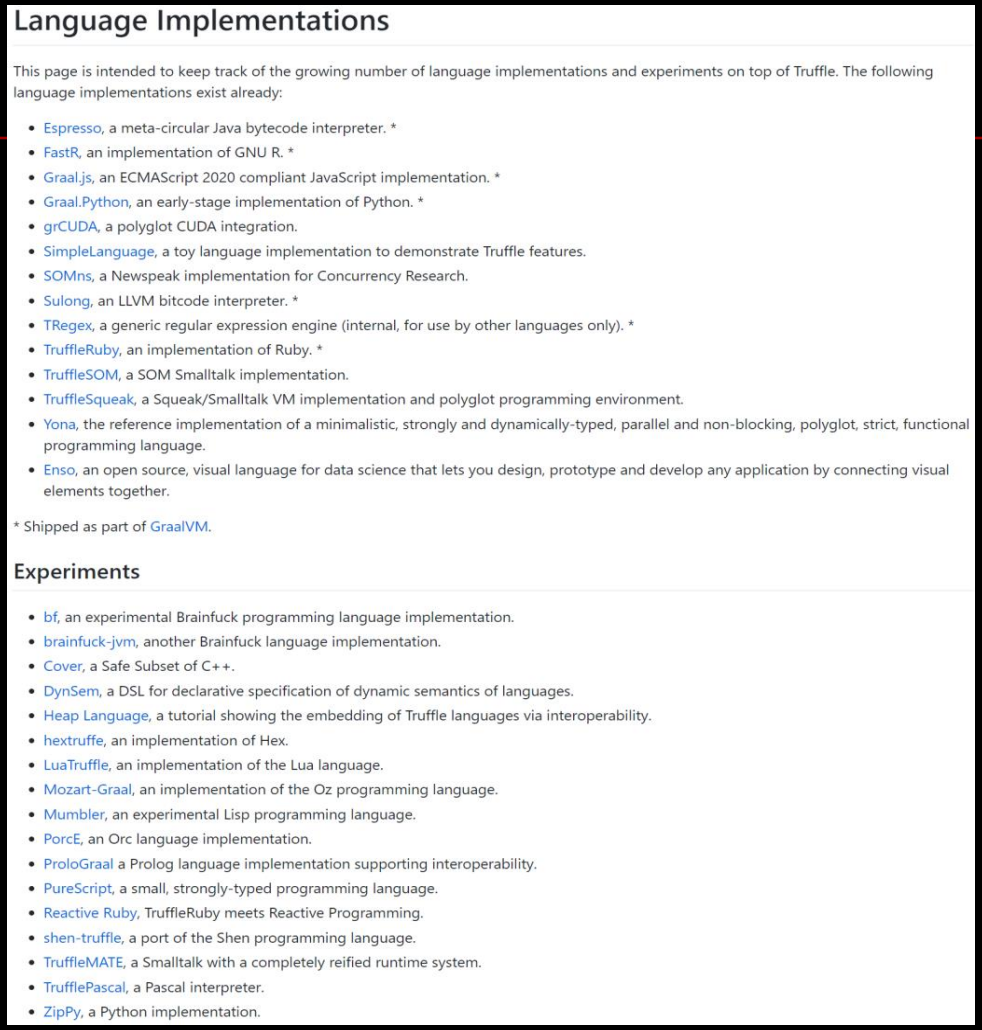
Implement a new language runtime



Source: "Turning the JVM into a Polyglot VM with Graal", Chris Seaton, Oracle Labs

Languages

- **https://github.com/oracle/graal/blob/master/truffle/docs/Languages.md**



Language Implementations

This page is intended to keep track of the growing number of language implementations and experiments on top of Truffle. The following language implementations exist already:

- Espresso, a meta-circular Java bytecode interpreter. *
- FastR, an implementation of GNU R. *
- Graal.js, an ECMAScript 2020 compliant JavaScript implementation. *
- Graal.Python, an early-stage implementation of Python. *
- grCUDA, a polyglot CUDA integration.
- SimpleLanguage, a toy language implementation to demonstrate Truffle features.
- SOMns, a Newspeak implementation for Concurrency Research.
- Sulong, an LLVM bitcode interpreter. *
- TRegex, a generic regular expression engine (internal, for use by other languages only). *
- TruffleRuby, an implementation of Ruby. *
- TruffleSOM, a SOM Smalltalk implementation.
- TruffleSqueak, a Squeak/Smalltalk VM implementation and polyglot programming environment.
- Yona, the reference implementation of a minimalistic, strongly and dynamically-typed, parallel and non-blocking, polyglot, strict, functional programming language.
- Enso, an open source, visual language for data science that lets you design, prototype and develop any application by connecting visual elements together.

* Shipped as part of GraalVM.

Experiments

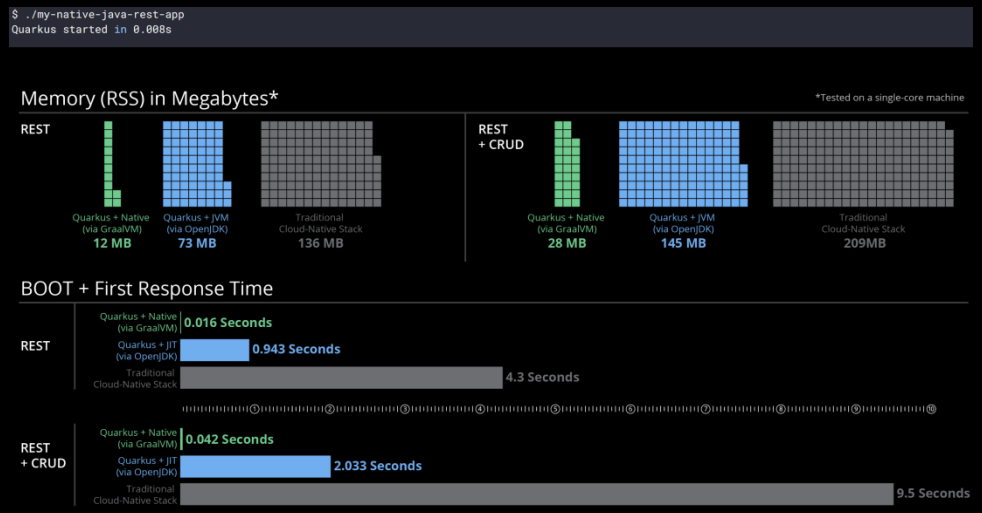
- bf, an experimental Brainfuck programming language implementation.
- brainfuck-jvm, another Brainfuck language implementation.
- Cover, a Safe Subset of C++.
- DynSem, a DSL for declarative specification of dynamic semantics of languages.
- Heap Language, a tutorial showing the embedding of Truffle languages via interoperability.
- hextruffe, an implementation of Hex.
- LuaTruffle, an implementation of the Lua language.
- Mozart-Graal, an implementation of the Oz programming language.
- Mumbler, an experimental Lisp programming language.
- PorcE, an Orc language implementation.
- ProloGraal a Prolog language implementation supporting interoperability.
- PureScript, a small, strongly-typed programming language.
- Reactive Ruby, TruffleRuby meets Reactive Programming.
- shen-truffle, a port of the Shen programming language.
- TruffleMATE, a Smalltalk with a completely reified runtime system.
- TrufflePascal, a Pascal interpreter.
- ZipPy, a Python implementation.

- **…**

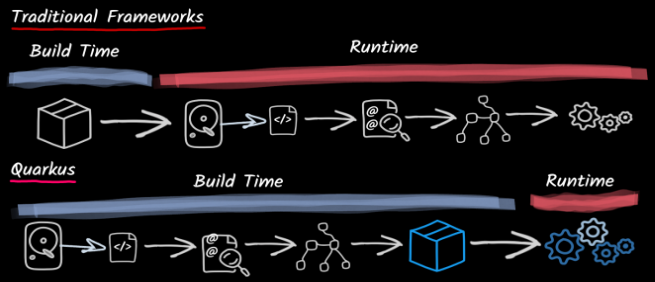
Cloud Native

- https://quarkus.io/

  A Kubernetes Native Java stack tailored for OpenJDK HotSpot and GraalVM, crafted from the best of breed Java libraries and standards.
- Amazingly fast boot time, incredibly low RSS memory!



- https://github.com/quarkusio/quarkus
- https://quarkus.io/vision/container-first

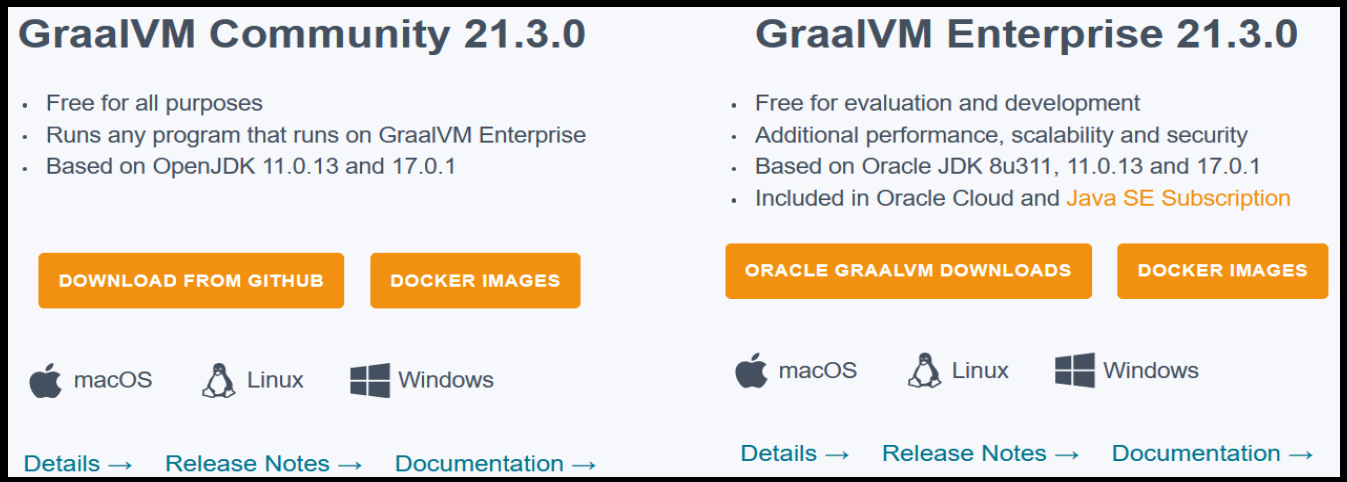


- Spring Framework & Boot, Micronaut, and more.
- Project Helidon
  https://helidon.io/
  A cloud-native, open-source set of Java libraries for writing microservices that run on a fast web core powered by Netty.
- Serverless
  https://www.graalvmonlambda.com/
  https://www.datadoghq.com/state-of-serverless/
  https://medium.com/graalvm/graalvm-at-facebook-af09338ac519
  …

2) Recent changes

- https://www.graalvm.org/release-notes/version-roadmap/

2.1 GraalVM 21.3

- 

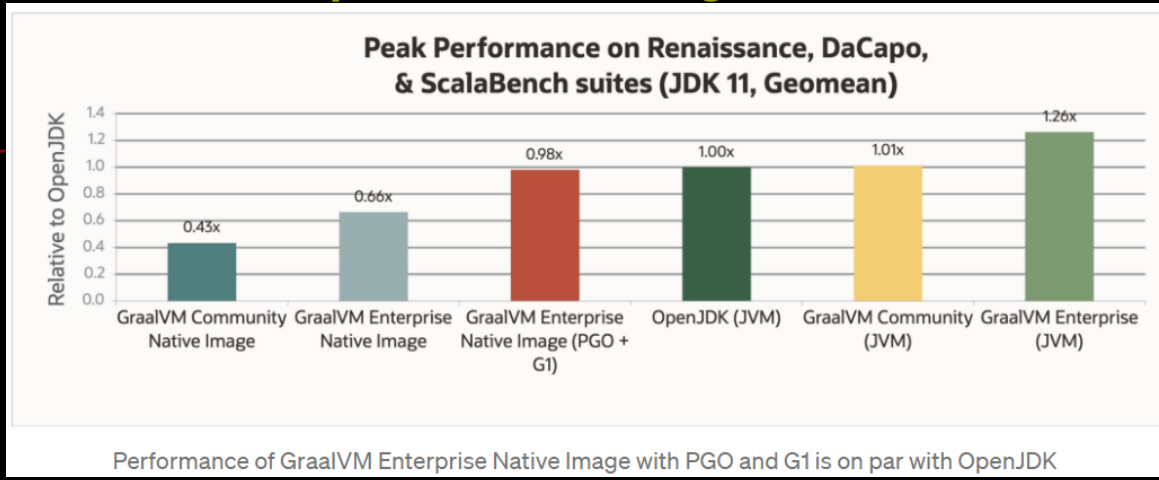
- https://www.graalvm.org/release-notes/21_3/
- Native Image

  introduces conditional reflection configuration for Native Image.

GraalVM Version	Image Build Time	Image Size
GraalVM 21.1	170 sec.	138 MByte
GraalVM 21.2	163 sec.	133 MByte
GraalVM 21.3	133 sec.	119 MByte

...

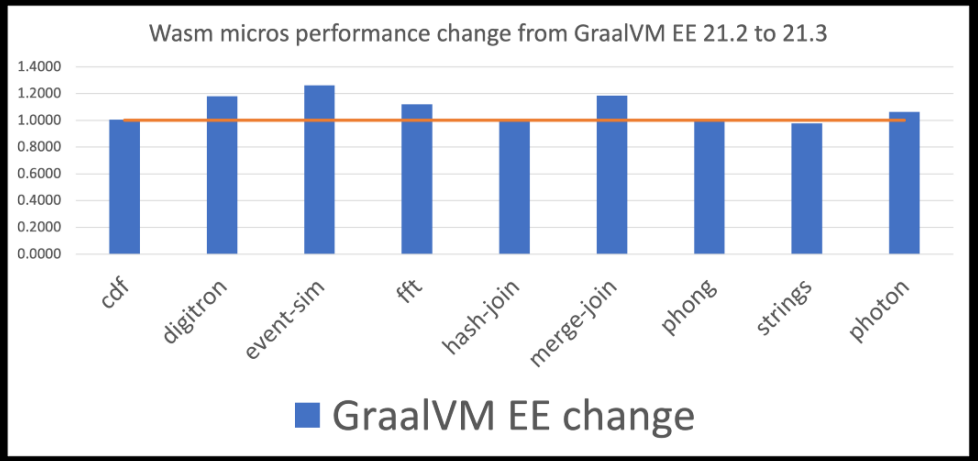
21.3 brings significant improvements to peak performance of GraalVM Enterprise Native Image.



Performance of GraalVM Enterprise Native Image with PGO and G1 is on par with OpenJDK

...

- **Wasm**



...

- **LLVM Runtime**

LLVM toolchain for compiling C/C++ was updated to version 12.0.1. We also improved modularization of the codebase around "managed mode" and "native mode" code. This reduces static build dependencies between these two modes, at the same time making the managed mode codebase more robust by removing all unmanaged memory accesses.

- **Support Java 17**

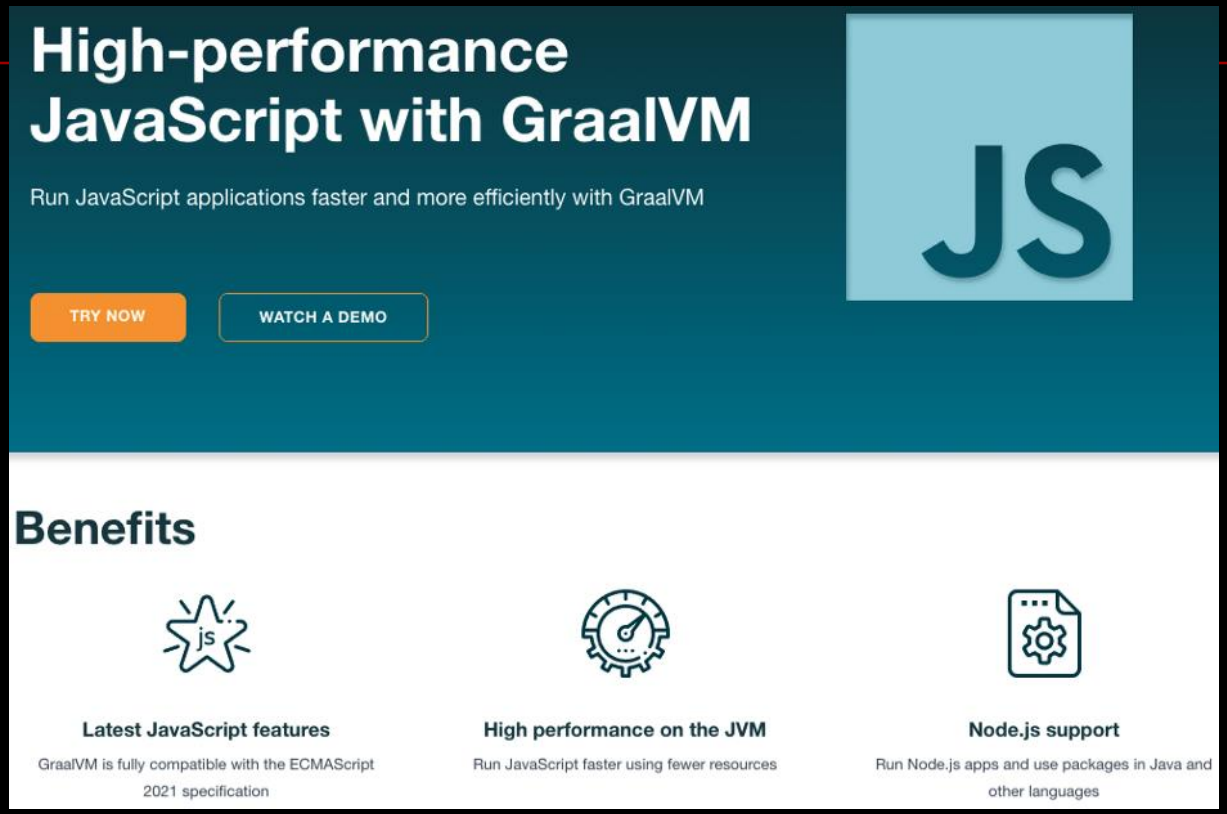
https://openjdk.java.net/projects/jdk/17/

306: Restore Always-Strict Floating-Point Semantics
356: Enhanced Pseudo-Random Number Generators
382: New macOS Rendering Pipeline
391: macOS/AArch64 Port
398: Deprecate the Applet API for Removal
403: Strongly Encapsulate JDK Internals
406: Pattern Matching for switch (Preview)
407: Remove RMI Activation
409: Sealed Classes
410: Remove the Experimental AOT and JIT Compiler
411: Deprecate the Security Manager for Removal
412: Foreign Function & Memory API (Incubator)
414: Vector API (Second Incubator)
415: Context-Specific Deserialization Filters

Faster LTS and free JDK with Java 17…

2.2 Wasm & JS

- https://medium.com/graalvm/graalvm-supports-ecmascript-2021-and-beyond-e12cdd288561



- https://medium.com/graalvm/mle-executing-javascript-in-oracle-database-c545feb1a010
- …

2.3 Wasm

- GraalWASM

https://github.com/oracle/graal/tree/master/wasm

GraalWasm is a WebAssembly engine implemented in the GraalVM. It can interpret and compile WebAssembly programs in the binary format, or be embedded into other programs.

We are working hard towards making GraalWasm more stable and more efficient, as well as to implement various WebAssembly extensions. Feedback, bug reports, and open-source contributions are welcome!

https://www.graalvm.org/reference-manual/wasm/
https://github.com/oracle/graal/tree/master/wasm
https://medium.com/graalvm/announcing-graalwasm-a-webassembly-engine-in-graalvm-25cd0400a7f2
…

2.4 Multi-Tier Compilation

- https://medium.com/graalvm/multi-tier-compilation-in-graalvm-5fbc65f92402

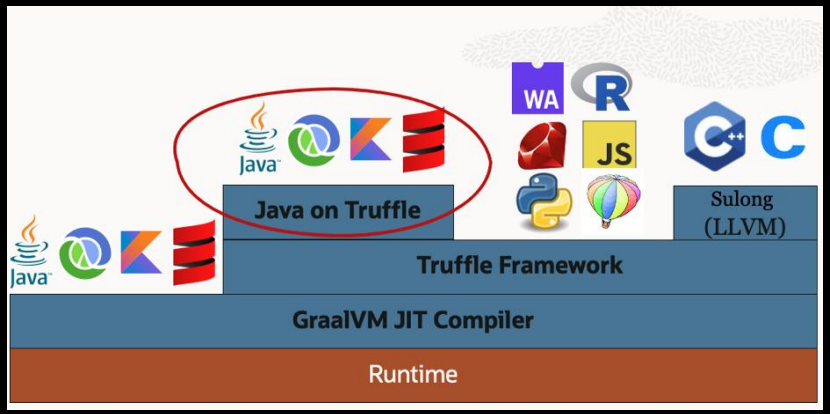
GraalVM 21.1 introduces a new feature called *multi-tier compilation* for languages implemented on Truffle. The multi-tier mode improves the warmup behavior, and is especially beneficial for programs that consist of large codebases, improving the startup times by 30%-50%. The core idea in the multi-tier mode is to separate compilation of the call targets into two tiers — the faster *first-tier*, and the slower *second-tier*. First-tier compilations perform fewer optimizations, but are quickly available and thus help the program to start-up faster. Second-tier compilations are slower because they do more optimizations, but they ensure that the program eventually reaches peak performance. As of the 21.1 release, the multi-tier compilation in GraalVM is turned on by default.

- …

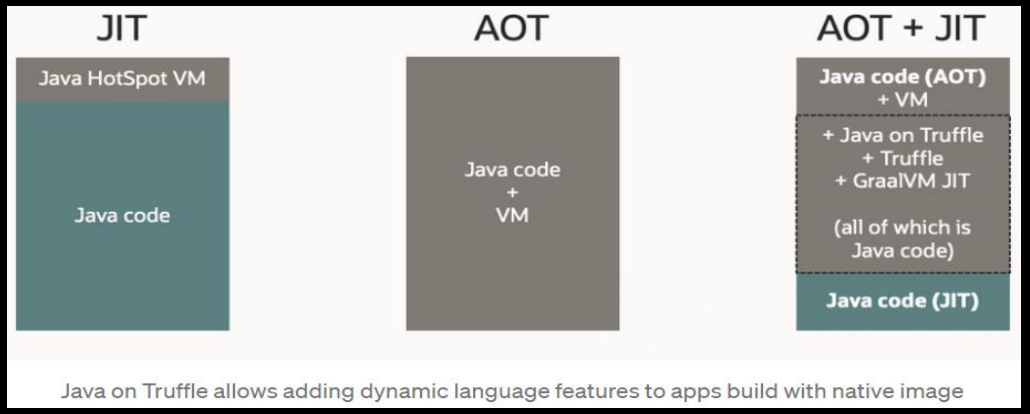
2.5 Java on Truffle

- https://medium.com/graalvm/java-on-truffle-going-fully-metacircular-215531e3f840

GraalVM 21.0 introduces a new installable component, named espresso, which provides a JVM implementation written in Java.



Mixing AOT and JIT



...

2.6 Development

- **VS Code**

  https://medium.com/graalvm/graalvm-21-0-vs-code-extensions-released-ab196354faeb

  https://medium.com/graalvm/debugging-polyglot-applications-on-graalvm-in-vscode-a62f97b48878

  https://medium.com/graalvm/performance-and-memory-analysis-with-graalvm-and-visualvm-in-vs-code-432bf7763e7a

  https://medium.com/graalvm/native-image-debugging-in-vs-code-2d5dda1989c1

- **Miscs**

  https://medium.com/graalvm/gradle-and-maven-plugins-for-native-image-with-initial-junit-testing-support-dde00a8caf0b

3) Automated Vectorization

- https://medium.com/graalvm/enhanced-automated-vectorization-in-graalvm-76cd99925b6d

In this article we look at how GraalVM Enterprise approached loop and linear vectorization optimizations introduced in 21.2 for new hardware architectures. We discuss considerations related to the GraalVM Native Image feature and a practical demo that you can try yourself. The particular optimizations discussed here are a part of GraalVM Enterprise, which is available with the Oracle Java SE Subscription.

Vectorization Optimizations

How does a compiler find the opportunity to utilize SIMD instructions? In GraalVM Enterprise Edition we have two optimizations which target two different potential sources of SIMD opportunities:

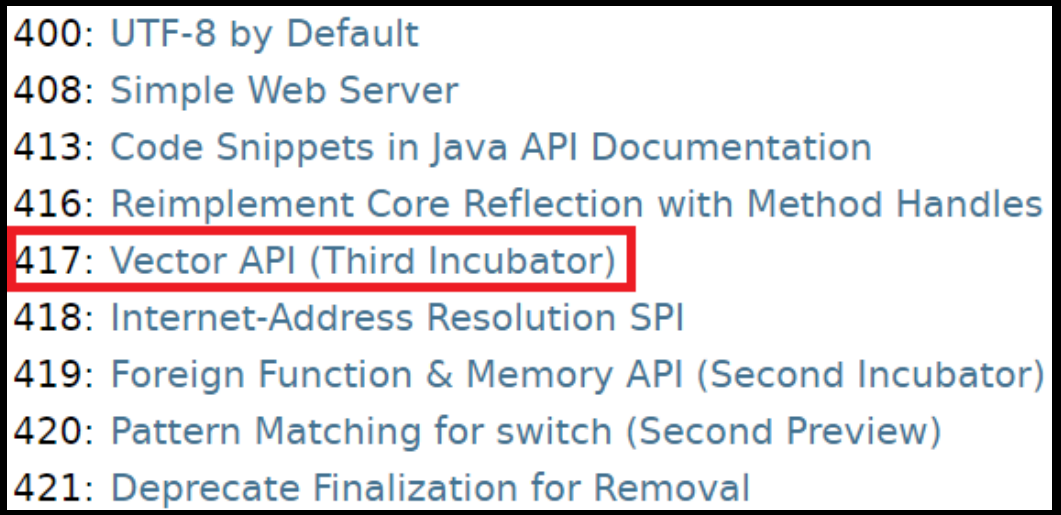
- Loop vectorizer — analyzes loops to try and reduce the entire loop into a SIMD form
- Linear vectorizer — analyzes sequential code for SIMD opportunities

The GraalVM LoopVectorization phase is enabled by default in all recent releases of GraalVM Enterprise Edition while the SIMDVectorization phase is new in GraalVM Enterprise Edition 21.2 and can be enabled using `-Dgraal.VectorizeSIMD=true`.

...

Auto Vectorization in Java

- http://daniel-strecker.com/blog/2020-01-14_auto_vectorization_in_java/
- https://inside.java/2021/01/27/extending-c2-autovectorization-capabilities/
- https://blogs.oracle.com/javamagazine/post/java-vector-api-simd
- https://openjdk.java.net/projects/jdk/18/



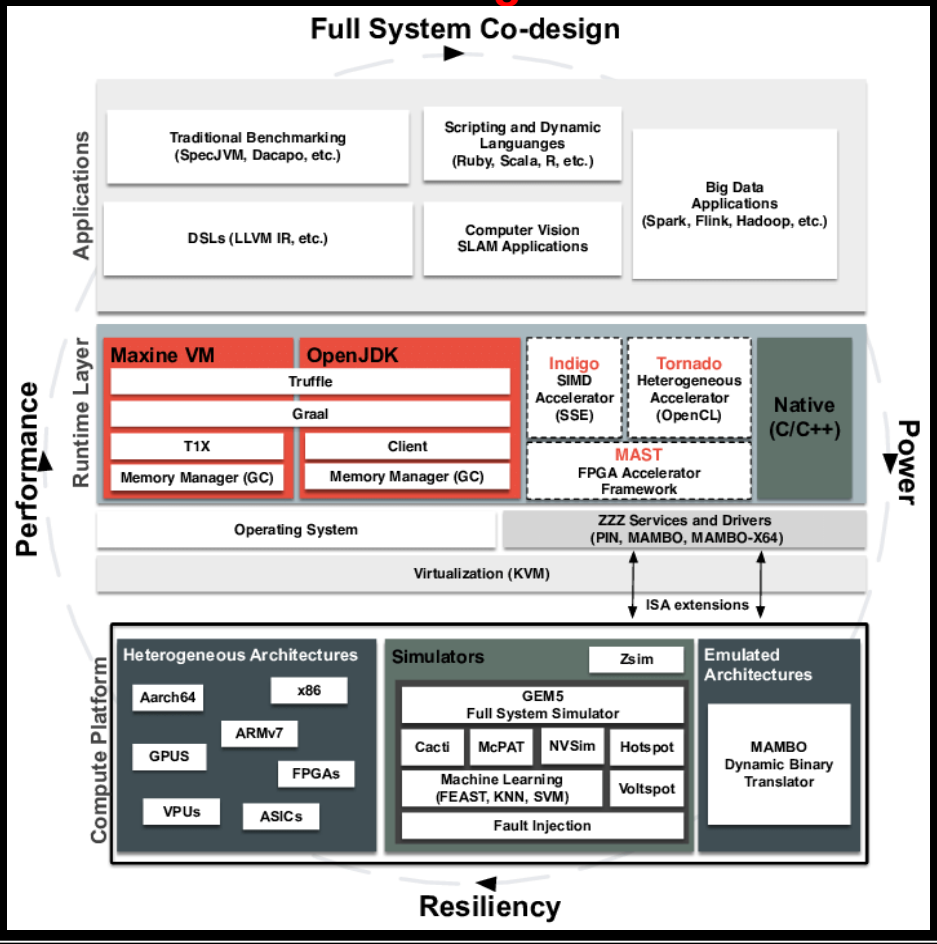
- ...

4) Project Beehive

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

  Beehive lab is part of the Advanced Processor Technologies Group at the University of Manchester specializing in hw/sw codesign.

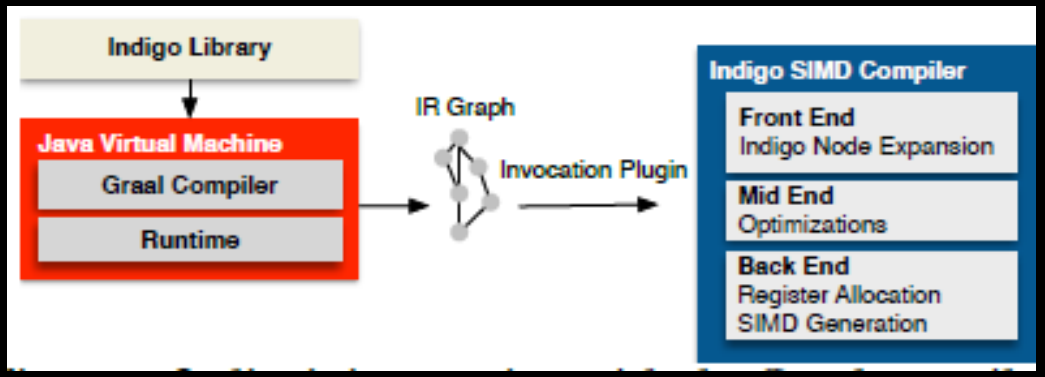
- Architecture and Design



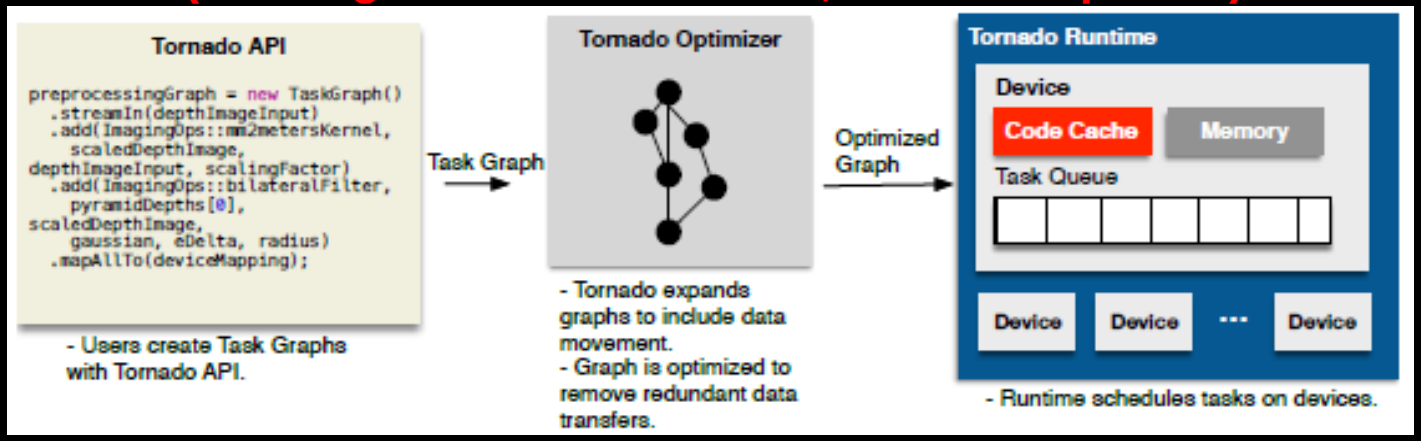
History

- https://research.ac.upc.edu/multiprog/multiprog2016/papers/multiprog-15.pdf
  Project Beehive: A Hardware/Software Co-designed Stack for Runtime and Architectural Research.

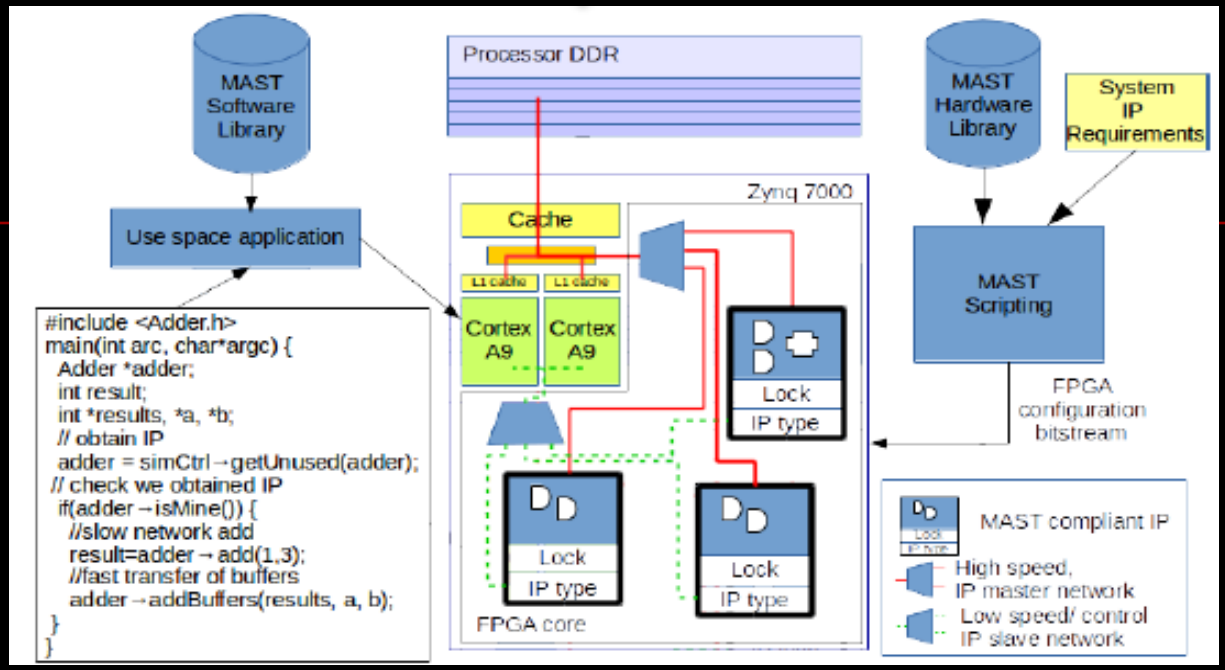
- Indigo(SIMD Accelerator)



- Tornado(Heterogeneous Accelerator, base on OpenCL)

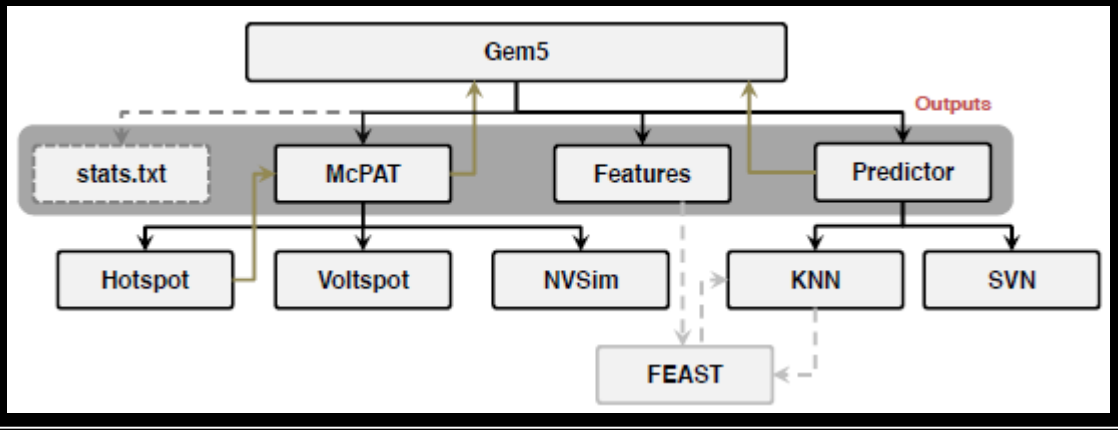


- **MAST(FPGA Accelerator Framework)**

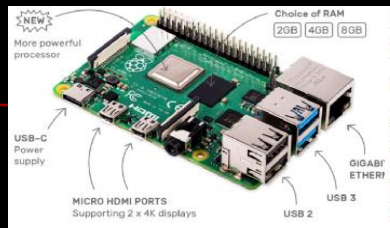


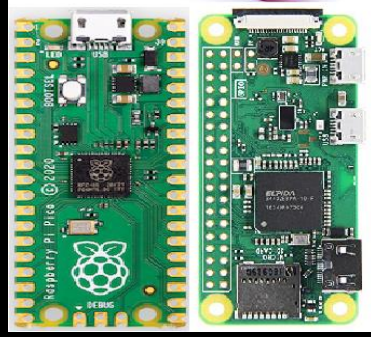
Source: https://dl.acm.org/doi/10.1145/3140607.3050764

- **Gem5 for Simulation**

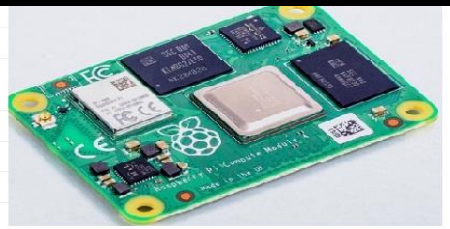


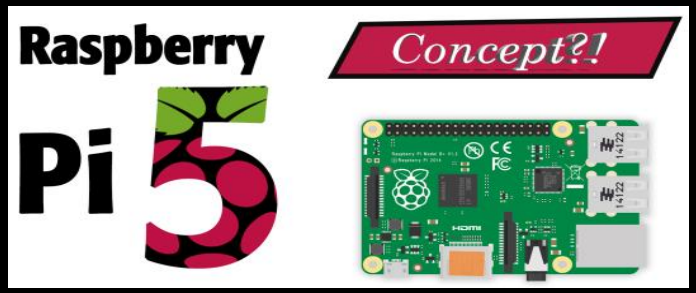
5) Testbed
5.1 Raspberry Pi





Features/Specs	Raspberry Pi 4B	Raspberry Pi 3 B+
Release date	24th June 2019	14th March 2018
SoC	Broadcom BCM2711 quad-core Cortex-A72 @ 1.5 GHz	Broadcom BCM2837B0 quad-core Cortex-A53 @ 1.4 GHz
GPU	VideoCore VI with OpenGL ES 1.1, 2.0, 3.0	VideoCore IV with OpenGL ES 1.1, 2.0
Video Decode	H.265 4Kp60, H.264 1080p60	H.264 & MPEG-4 1080p30
Video Encode	H.264 1080p30	
Memory	1GB, 2GB, or 4GB LPDDR4	1GB LPDDR2
Storage	microSD card	
Video & Audio Output	2x micro HDMI ports up to 4Kp60 3.5mm AV port (composite + audio) MIPI DSI connector	1x HDMI 1.4 port up to 1080p60 3.5mm AV port (composite + audio) MIPI DSI connector
Camera	MIPI CSI connector	
Ethernet	Native Gigabit Ethernet	Gigabit Ethernet over USB (300 Mbps max.)
WiFi	Dual band 802.11 b/g/n/ac	
Bluetooth	Bluetooth 5.0 + BLE	Bluetooth 4.2 + BLE
USB	2x USB 3.0 + 2x USB 2.0	4x USB 2.0
Expansion	40-pin GPIO header	
Power Supply	5V via USB type-C up to 3A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT	5V via micro USB up to 2.5A 5V via GPIO header up to 3A Power over Ethernet via PoE HAT
Dimensions	85×56 mm	
Default OS	Raspbian (after June 24, 2019)	Raspbian (after March 2018)
Price	$35 (1GB RAM), $45 (2GB RAM), $55 (4GB RAM)	$35 (1GB RAM)





Vulkan

- https://www.raspberrypi.com/news/vulkan-update-version-1-1-conformance-for-raspberry-pi-4/
- https://qengineering.eu/install-vulkan-on-raspberry-pi.html
- https://github.com/karnkaul/rpi4-install-vulkan
- https://archive.fosdem.org/2021/schedule/event/rpi4_vulkan/
- …

OpenCL

- Upcoming…

IPO

- https://www.tomshardware.com/news/raspberry-pi-plans-spring-2022-listing

5.1.1 System

Fedora

- A Linux distribution developed by the community-supported Fedora Project which is sponsored primarily by Red Hat
- https://en.wikipedia.org/wiki/Fedora_(operating_system)
- https://getfedora.org/
- https://alt.fedoraproject.org/alt/
- https://spins.fedoraproject.org/
- https://fedoraproject.org/wiki/Architectures/ARM
- https://fedoramagazine.org/
- https://silverblue.fedoraproject.org/
- Developer friendly!

Fedora 36

- Enhance the (rpm-)ostree stack to natively support OCI/Docker containers as a transport and delivery mechanism for operating system content.
  https://fedoraproject.org/wiki/Changes/OstreeNativeContainer
- ...

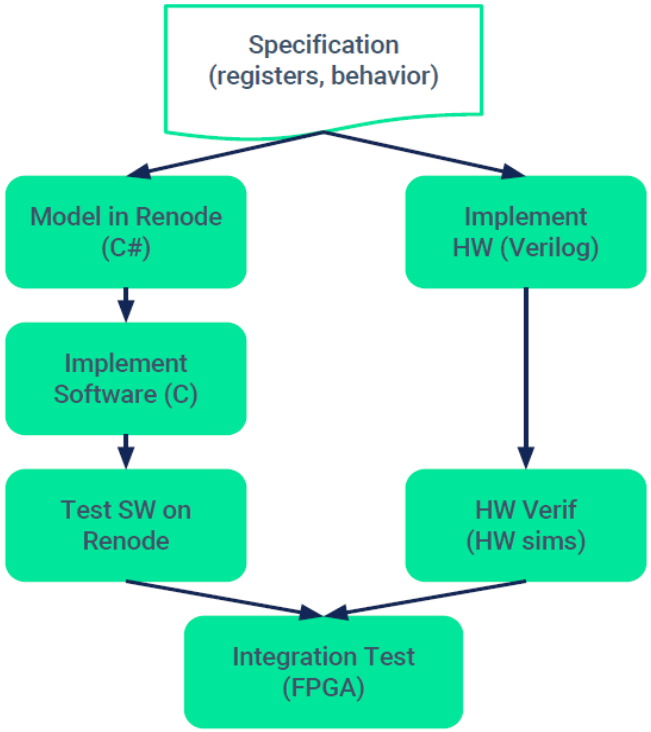
5.2 Renode

- **Parallel HW/SW Development with Renode**

- HW spec developed between HW and SW teams
- SW team implements spec in Renode and writes firmware against spec, testing on Renode
- In parallel, HW is implementing and testing HW design
- Integration test via FPGA and/or HW simulator

EXAMPLE

HDMI device. We had SW working against spec, under Renode, in advance of HW.



Source: "Hardware/Software Co-Design with the Open Source Renode Framework and RISC-V", Michael Gielda, Getting Started With RISC-V NA Tour 2019.

- **For details, you may refer to my previous talk "Renode: a Swiss Army Knife for RISC-V development" at .Net Conf China 2021(Online).**

5.3 RK3588/RK3568 based SBC

RK3568

- https://www.rock-chips.com/a/en/products/RK35_Series/2021/0113/1276.html

CPU	• Quad-Core ARM Cortex-A55，up to 2.0GHz
GPU	• ARM G52 2EE
	• Support OpenGL ES 1.1/2.0/3.2，OpenCL 2.0，Vulkan 1.1
	• High performance dedicated 2D processor
NPU	• Support 0.8T
Multi-Media	• Support 4K 60fps H.265/H.264/VP9 decoder
	• Support 1080P 60fps H.265/H.264 encoder
	• Support 8M ISP with HDR
Display	• Support multi-display
	• Support eDp/HDMI2.0/MIPI/LVDS/24bit RGB/T-CON
Interface	• Support USB2.0/USB3.0/PCIE3.0/PCIE2.1/SATA3.0/QSGMII

- **Not the best choice…**

RK3588

- **Spec**

https://www.cnx-software.com/2021/12/16/rockchip-rk3588-datasheet-sbc-coming-soon/

- CPU – 4x Cortex-A76 @ **up to 2.4/2.6 GHz** and 4x Cortex-A55 cores **@ 1.8 GHz** in dynamIQ configuration
- GPU
  - **Arm Mali-G610 MP4** "Odin" GPU with support for OpenGLES 1.1, 2.0, and 3.2, OpenCL up to 2.2 and Vulkan1.2
  - 2D graphics engine up to 8192×8192 source, 4096×4096 destination
- AI Accelerator – 6 TOPS NPU 3.0 (Neural Processing Unit)
- VPU
  - Video decoding
    - 8Kp60 H.265, VP9, AVS2, 8Kp30 H.264 AVC/MVC
    - **4Kp60 AV1**
    - 1080p60 MPEG-2/-1, VC-1, VP8
  - Real-time 8Kp30 encoding with H.265/H.264; multi-channel encoding supported at lower resolutions
- Memory I/F – LPDDR4/LPDDR4x/LPDDR5 up to 32GB
- Storage – eMMC 5.1, SD/MMC, SATA 3.0 (multiplexed with PCIe 2.0), FSPI (Flexible SPI)
- Video Output
  - Dual HDMI 2.1 / eDP 1.3 up to 8Kp60
  - Dual DisplayPort 1.4a up to 8Kp30 (multiplexed with USB 3.0)
  - Dual MIPI DSI output up to 4Kp60
  - Bt.1120 video output up to 1080p60
  - Optional dual LVDS up to 1080p60 via RK628 chip.
  - Up to four independent displays (**up to 1x 8Kp60, 2x 4Kp60, 1x 1080p60**)
- Video Input/Camera
  - 48MP (2x 24MP) ISP with HDR and 3D NR support; multi-camera input
  - 2x MIPI DC (4-lane DPHY v2.0 or 3-lane CPHY V1.1)
  - 4x 2-lane MIPI CSI
  - DVP camera interface
  - HDMI Rx 2.0 interface up to 4Kp60 with HDCP 2.3 support

…

- **Dev Board**

图：搭载RK3588的开发板首度向开发者公开

- **Waiting…**

II. TornadoVM

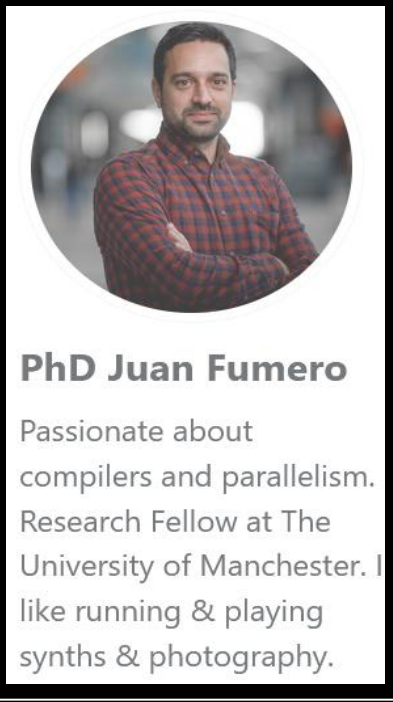
1) Architecture & Design

1.1 Overview

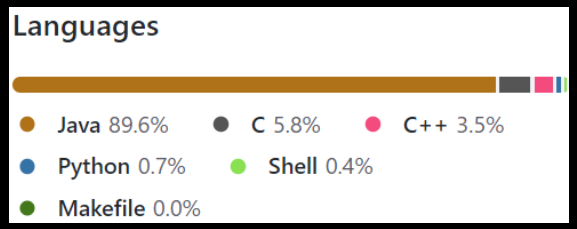
- https://www.tornadovm.org/
- https://github.com/beehive-lab/TornadoVM

  A practical and efficient heterogeneous programming framework for managed languages.

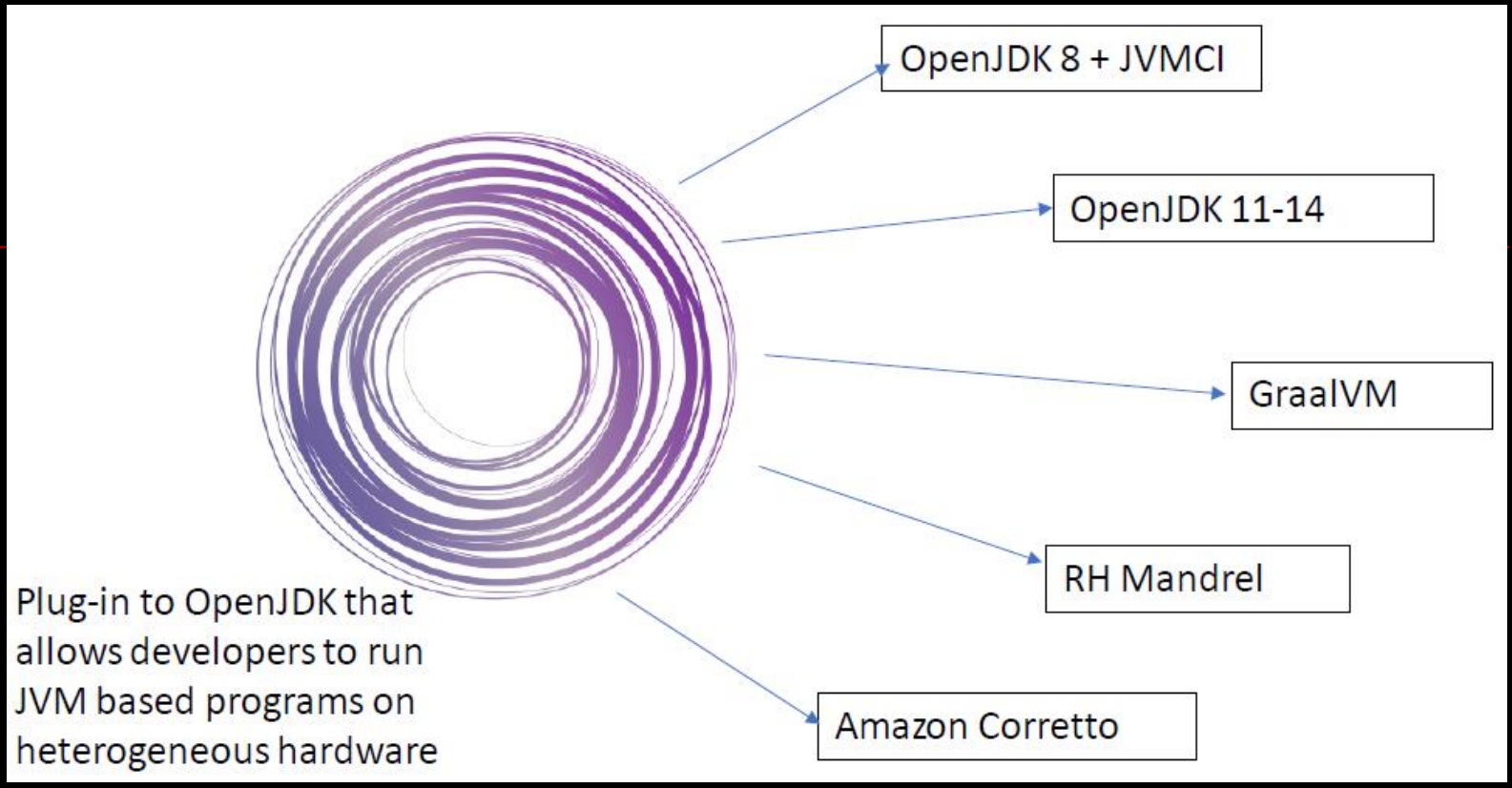
TornadoVM is a plug-in to OpenJDK and GraalVM that allows programmers to automatically run Java programs on heterogeneous hardware. TornadoVM currently targets OpenCL-compatible devices and it runs on multi-core CPUs, dedicated GPUs (NVIDIA, AMD), integrated GPUs (Intel HD Graphics and ARM Mali), and FPGAs (Intel and Xilinx).



- Src



- ...



Source: https://jjfumero.github.io/talks/

Stats

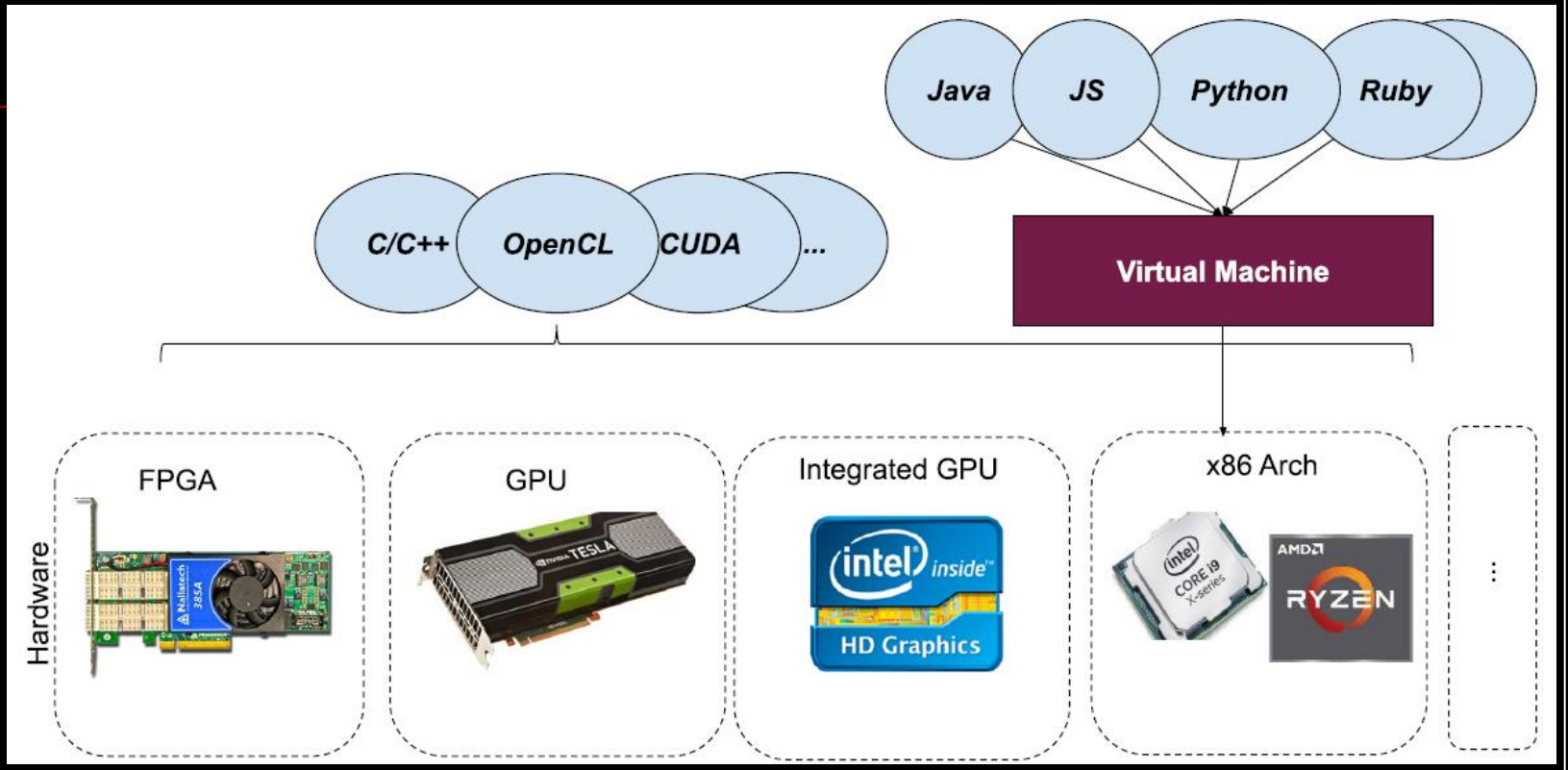
```
[mydev@fedora TornadoVM-master]$ git branch
* master
[mydev@fedora TornadoVM-master]$ tokei
===============================================================================
 Language            Files        Lines         Code     Comments       Blanks
===============================================================================
 BASH                    3          390          298           56           36
 C Header               47        10509         6256         2706         1547
 CMake                   3          120           92           10           18
 C++                    28         5280         2889         1681          710
 Java                 1265       184257       116455        40536        27266
 JavaScript              1           71           41           22            8
 JSON                    5          159          151            0            8
 Python                  7         1190          824          177          189
 Shell                  13          641          421          112          108
 Plain Text              2          234            0          187           47
 XML                    20         3502         3395           51           56
-------------------------------------------------------------------------------
 Markdown               32         3241            0         2187         1054
 |- BASH                17          694          629           32           33
 |- C                    4          111           95           13            3
 |- Java                 7          480          360           36           84
 |- JSON                 1           25           25            0            0
 |- LLVM                 1          427          422            5            0
 |- XML                  4           87           79            1            7
 (Total)                            5065         1610         2274         1181
===============================================================================
 Total                1426       211418       132432        47812        31174
===============================================================================
[mydev@fedora TornadoVM-master]$
```

hierarchical structure

```
[mydev@fedora TornadoVM-master]$ tree -L 2 .
.
├── assembly
│   ├── assembly.xml
│   ├── pom.xml
│   └── src
├── benchmarks
│   ├── pom.xml
│   └── src
├── bin
│   ├── compile.sh
│   ├── pullGraalJars.sh
│   └── updatePATHS.sh
├── CODE_OF_CONDUCT.md
├── CONTRIBUTING.md
├── docs -> assembly/src/docs/
├── drivers
│   ├── drivers-common
│   ├── opencl
│   ├── opencl-jni
│   ├── pom.xml
│   ├── ptx
│   ├── ptx-jni
│   ├── spirv
│   └── spirv-levelzero-jni
├── etc
│   ├── git-commit-template.txt
│   ├── intel-fpga.conf
│   ├── tornado.conf
│   ├── tornado.properties
│   ├── tornadoVM_Logo.jpg
│   ├── virtual-device-template.json
│   └── xilinx-fpga.conf
├── examples
│   ├── pom.xml
│   └── src
├── INSTALL.md
├── Jenkinsfile
├── LICENSE_APACHE2
├── LICENSE_GPLv2
├── LICENSE_GPLv2CE
├── LICENSE_MIT
├── Makefile
├── matrices
│   ├── pom.xml
│   └── src
├── pom.xml
├── README.md
├── runtime
│   ├── pom.xml
│   └── src
├── scripts
│   ├── eclipseSetup.py
│   ├── enable_timers.sh
│   ├── runSPIRV-Example.sh
│   ├── spirvTests.sh
│   ├── spirv-validator.sh
│   └── templates
├── tornado-annotation
│   ├── pom.xml
│   └── src
├── tornado-api
│   ├── pom.xml
│   └── src
└── unittests
    ├── pom.xml
    └── src
```

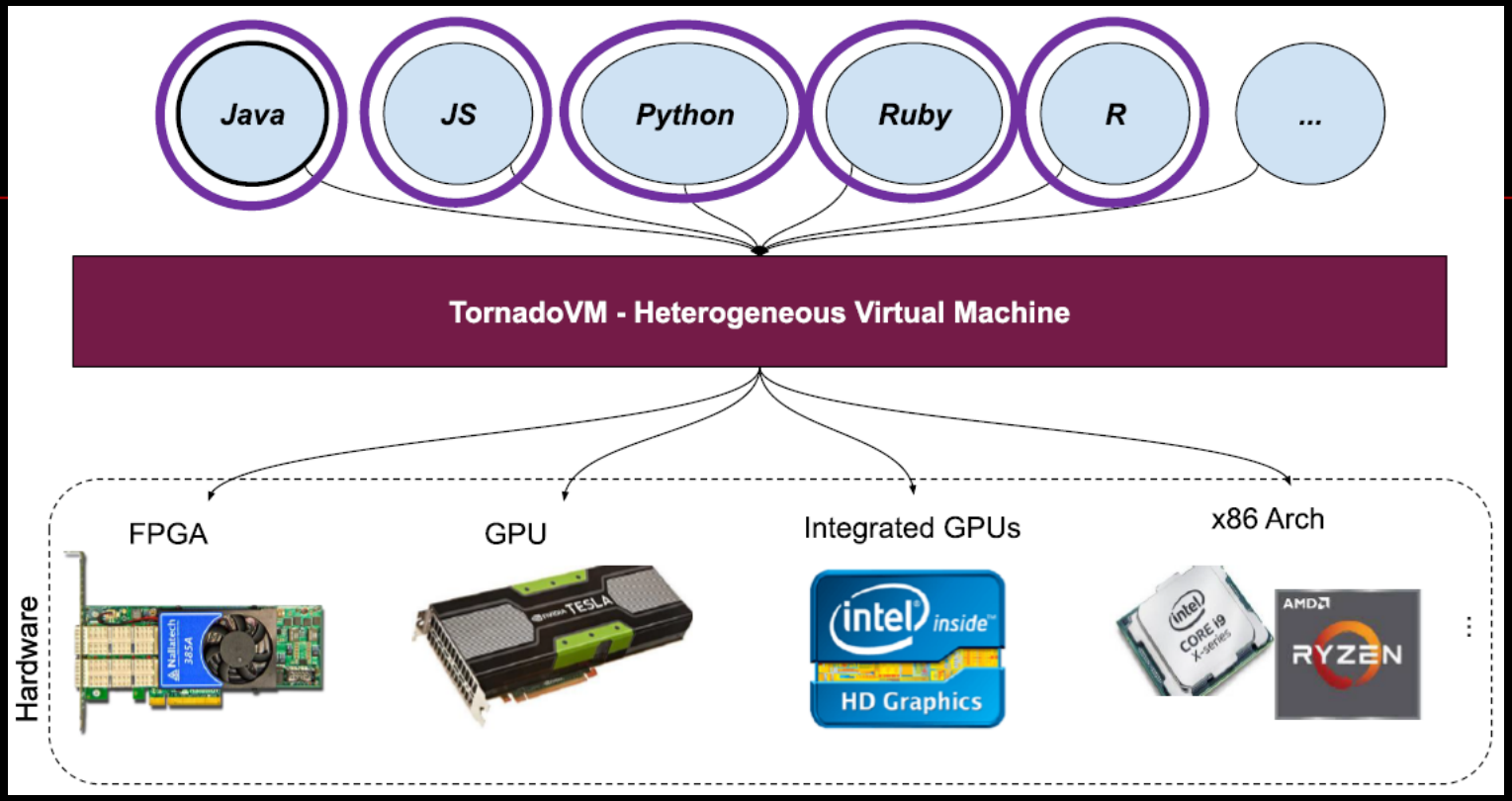
1.2 Motivation

- Current Computer Systems & Prog. Lang



Source: https://jjfumero.github.io/talks/

Ideal System for Managed Languages



Source: https://jjfumero.github.io/talks/

1.3 Heterogeneous Parallel Computing

- https://en.wikipedia.org/wiki/Heterogeneous_computing
- …

1.3.1 OpenCL

1.3.1.1 OpenCL 3.0

- https://www.khronos.org/opencl/

OpenCL 3.0

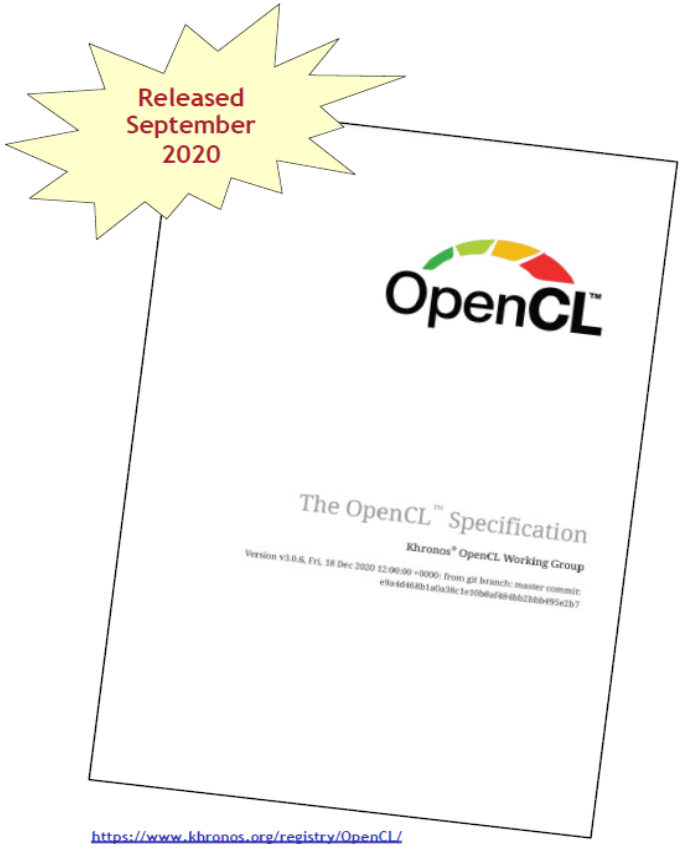
Increased Ecosystem Flexibility
All functionality beyond OpenCL 1.2 queryable
Macros for optional OpenCL C language features
Widely adopted extensions to be integrated into core

OpenCL C++ for OpenCL
Open-source C++ for OpenCL front end compiler combines OpenCL C and C++17 replacing OpenCL C++ language spec

Unified Specification
All versions of OpenCL in one specification for easier maintenance, evolution and accessibility
Source on Khronos GitHub for community feedback, functionality requests and bug fixes

New Functionality
Subgroups with SPIR-V 1.3 in core (optional)
Asynchronous DMA extension for embedded processors

Easy OpenCL 3.0 migration for applications
OpenCL 1.2 applications - no change
OpenCL 2.X applications - no code changes if all used functionality present
Queries recommended for future portability

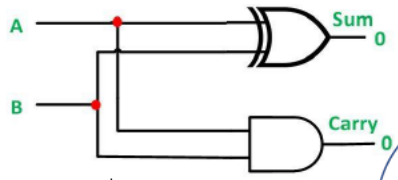


Source: "State of the Union OpenCL Working Group", Neil Trevett, IWOCL 2021.

- https://github.com/KhronosGroup/OpenCL-SDK
- …

Replace Verilog/VHDL with OpenCL

```
library ieee;
use ieee.std_logic_1164.all;


entity half_adder is                -- Entity
 port (a, b: in std_logic;
   sum, carry: out std_logic);
end half_adder;


architecture structure of half_adder is   -- Architecture
 component xor_gate                -- xor component
  port (i1, i2: in std_logic;
    o1: out std_logic);
 end component;


 component and_gate                -- and component
  port (i1, i2: in std_logic;
    o1: out std_logic);
 end component;


begin
  u1: xor_gate port map (i1 => a, i2 => b, o1 => sum);
  u2: and_gate port map (i1 => a, i2 => b, o1 => carry);
end structure;
```




```
kernel void sum
(global float a,
 global float b,
 global float *result)
{
  result[0] = a + b;
}
```

Industry is pushing for OpenCL on FPGAs!

Source: https://jjfumero.github.io/talks/

1.4 API

- E.g.

```
class Compute {
  public static void mxm(Matrix2DFloat A, Matrix2DFloatB,
                         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);
      }
    }
  }
}
```

Annotation

```
TaskSchedule ts = new TaskSchedule("s0");
ts.task("t0", Compute::mxm, matrixA, matrixB, matrixC, size)
   .streamOut(matrixC)
   .execute();
```

To run:

```
$ tornado Compute
```

tornado command is just an alias to Java and all the parameters to enable TornadoVM

Source: https://jjfumero.github.io/talks/

- **Similar to Numba**
  http://numba.pydata.org/
  An open source JIT compiler that translates a subset of Python and NumPy code into fast machine code.

Accelerate Python Functions

Numba translates Python functions to optimized machine code at runtime using the industry-standard LLVM compiler library. Numba-compiled numerical algorithms in Python can approach the speeds of C or FORTRAN.

You don't need to replace the Python interpreter, run a separate compilation step, or even have a C/C++ compiler installed. Just apply one of the Numba decorators to your Python function, and Numba does the rest.

```
from numba import jit
import random

@jit(nopython=True)
def monte_carlo_pi(nsamples):
    acc = 0
    for i in range(nsamples):
        x = random.random()
        y = random.random()
        if (x ** 2 + y ** 2) < 1.0:
            acc += 1
    return 4.0 * acc / nsamples
```

Built for Scientific Computing

Numba is designed to be used with NumPy arrays and functions. Numba generates specialized code for different array data types and layouts to optimize performance. Special decorators can create universal functions that broadcast over NumPy arrays just like NumPy functions do.

Numba also works great with Jupyter notebooks for interactive computing, and with distributed execution frameworks, like Dask and Spark.

```
@numba.jit(nopython=True, parallel=True)
def logistic_regression(Y, X, w, iterations):
    for i in range(iterations):
        w -= np.dot(((1.0 /
                (1.0 + np.exp(-Y * np.dot(X, w)))
                - 1.0) * Y), X)
    return w
```

Parallelize Your Algorithms

Numba offers a range of options for parallelizing your code for CPUs and GPUs, often with only minor code changes.

Simplified Threading

```
@jit(nopython=True, parallel=True)
def simulator(out):
    # iterate loop in parallel
    for i in prange(out.shape[0]):
        out[i] = run_sim()
```

Numba can automatically execute NumPy array expressions on multiple CPU cores and makes it easy to write parallel loops.

Learn More » Try Now »

SIMD Vectorization

```
LBB0_8:
    vmovups (%rax,%rdx,4), %ymm0
    vmovups (%rcx,%rdx,4), %ymm1
    vsubps  %ymm1, %ymm0, %ymm2
    vaddps  %ymm2, %ymm2, %ymm2
```

Numba can automatically translate some loops into vector instructions for 2-4x speed improvements. Numba adapts to your CPU capabilities, whether your CPU supports SSE, AVX, or AVX-512.

Learn More » Try Now »

GPU Acceleration

NVIDIA CUDA

With support for both NVIDIA's CUDA and AMD's ROCm drivers, Numba lets you write parallel GPU algorithms entirely from Python.

Numba CUDA » Numba ROCm »

…

https://github.com/numba/numba

https://numba.readthedocs.io/

https://github.com/numba/llvmlite

…

Expanding

Expressiveness:
Single API for expressing loop parallelism (**@Parallel**) and kernel parallelism (**implicit thread indexing**)

Powerful API:
Single API to express **data parallelism**, **task parallelism** and **pipeline parallelism**

```
var ts = new TaskSchedule("s0")
 .task("t0", kernel1, in, tmp1)
 .task("t1", kernel2, tmp1, temp2)
 …
 .task("tn", kernelN, tmpn, out)
 .streamOut(out);
```

High-level & Low-level enough:
If required, users can access to local memory, barriers, and thread dispatch

Source: https://jjfumero.github.io/talks/

Expressing Kernel Parallelism within TornadoVM

TornadoVM's default API

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

It exploits loop parallelism

TornadoVM's New API

```
public void vectorAdd(float[] a,
                      float[] b,
                      float[] c,
                      TornadoVMContext context) {
  int i = context.threadIdx;
  c[i] = a[i] + b[i];
}
```

It exploits **kernel parallelism**.
The kernel expresses the work to be done per thread. The scale-out happens when creating multiple instances of this kernel.

Source: https://jjfumero.github.io/talks/

1.5 Task

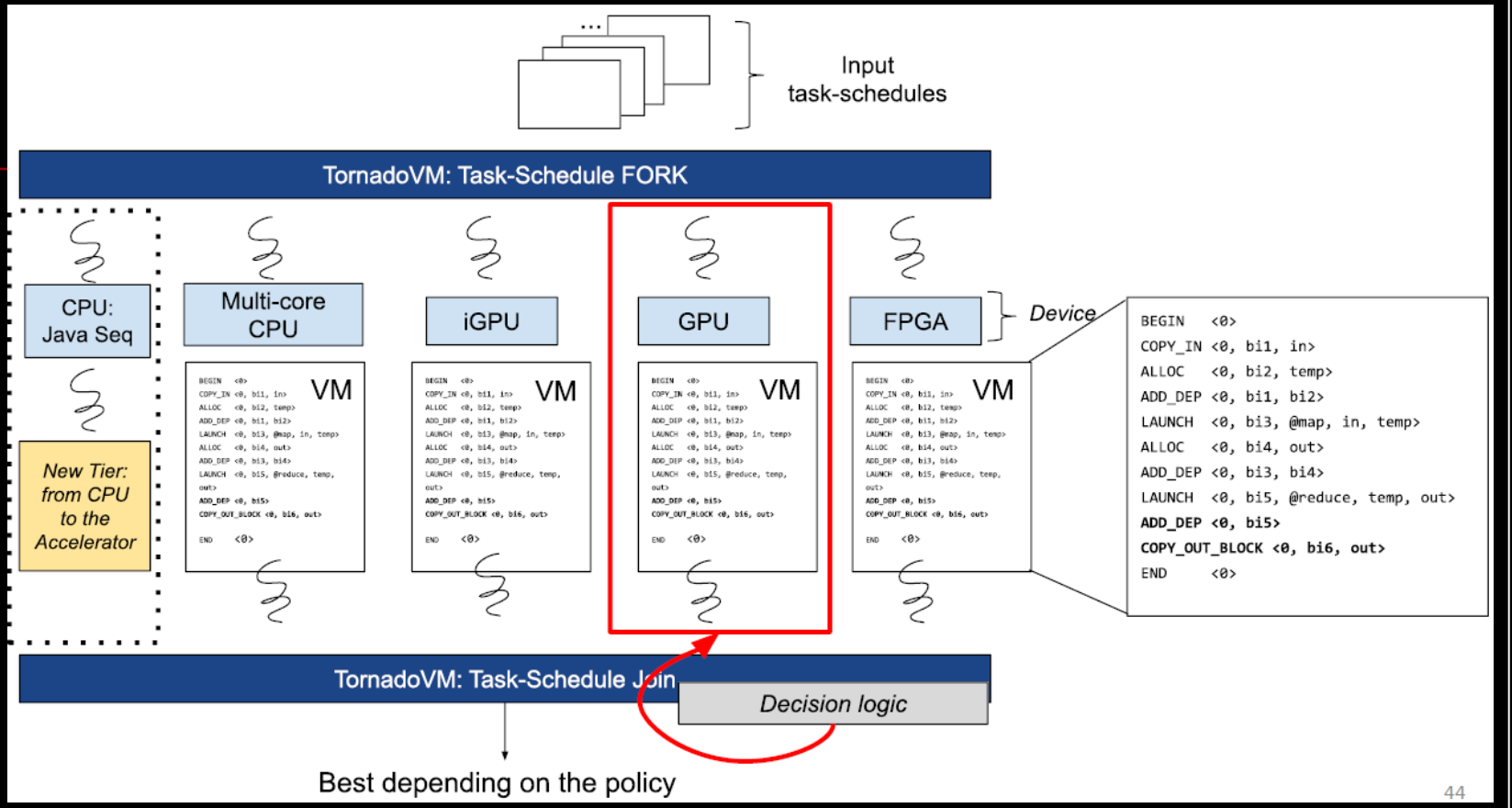
Overview

- $SRC_TORNADOVM/runtime/src/main/java/uk/ac/manchester/tornado/runtime/graph/nodes/TaskNode.java
  $SRC_TORNADOVM/tornado-api/src/main/java/uk/ac/manchester/tornado/api/TaskSchedule.java
  …

Task Scheduling

- For some new ideas, you may refer to my follow-up like “The 3rd round discussion on eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing".

Live Task Migration



Source: https://jjfumero.github.io/talks/

1.6 TornadoVM Bytecode

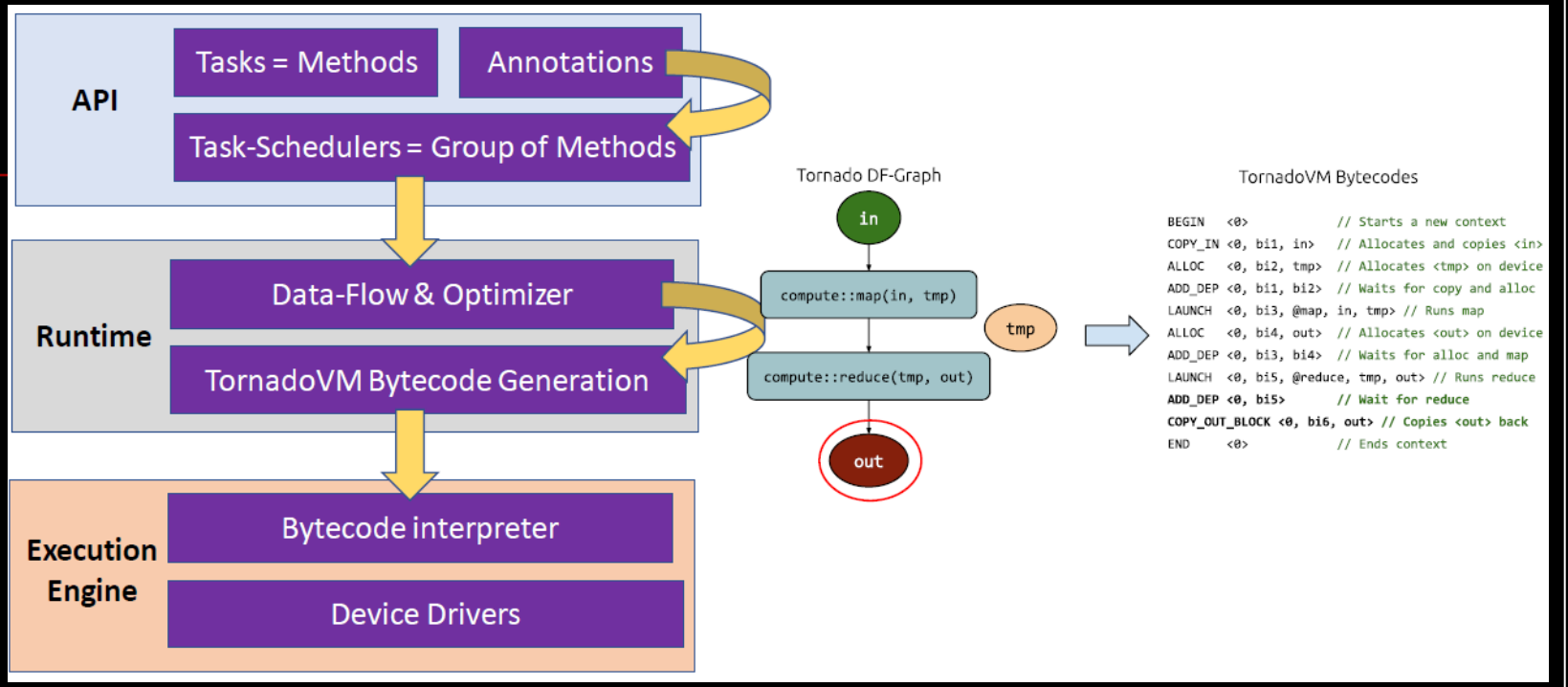
Overview

- **$SRC_TORNADOVM/runtime/src/main/java/uk/ac/manchester/tornado/runtime/graph/TornadoGraphAssembler.java**

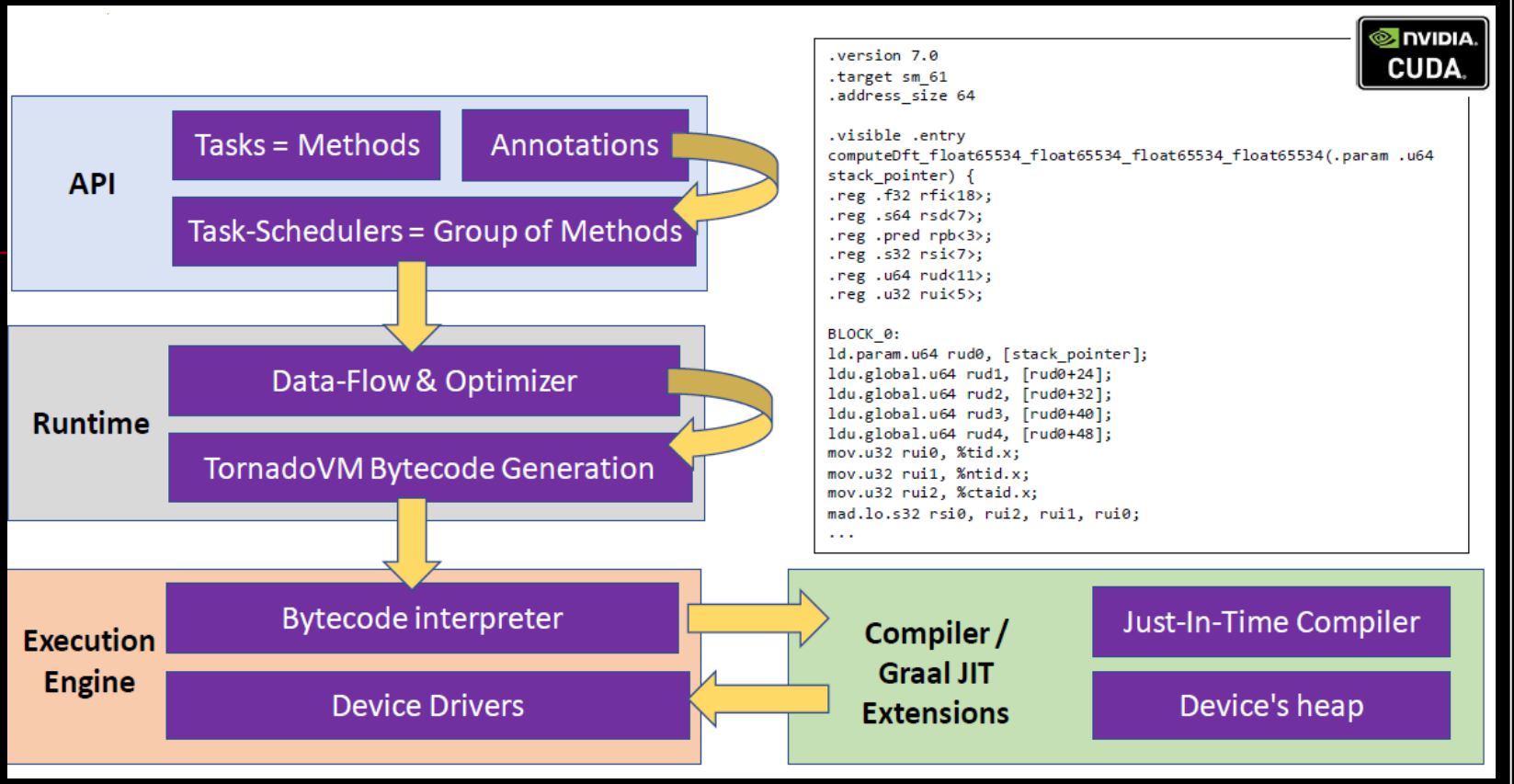
```
public class TornadoGraphAssembler {

    public enum TornadoVMBytecodes {
        // @formatter:off
        ALLOCATE((byte) 10),              // ALLOCATE(obj,dest)
        COPY_IN((byte) 11),               // COPY(obj, src, dest)
        STREAM_IN((byte) 12),             // STREAM_IN(obj, src, dest)
        STREAM_OUT((byte) 13),            // STREAM_OUT(obj, src, dest)
        STREAM_OUT_BLOCKING((byte) 14), // STREAM_OUT(obj, src, dest)
        LAUNCH((byte) 15),                // LAUNCH(dep list index)
        BARRIER((byte) 16),               // BARRIER <events>
        SETUP((byte) 17),
        BEGIN((byte) 18),                 // BEGIN(num contexts, num stacks, num dep lists)
        ADD_DEP((byte) 19),               // ADD_DEP(list index)
        CONTEXT((byte) 20),               // CONTEXT(ctx)
        END((byte) 21),                   // END(ctx)
        CONSTANT_ARGUMENT((byte) 22),
        REFERENCE_ARGUMENT((byte) 23);
        // @formatter:on

        private byte value;

        TornadoVMBytecodes(byte value) {
            this.value = value;
        }
```

- ...

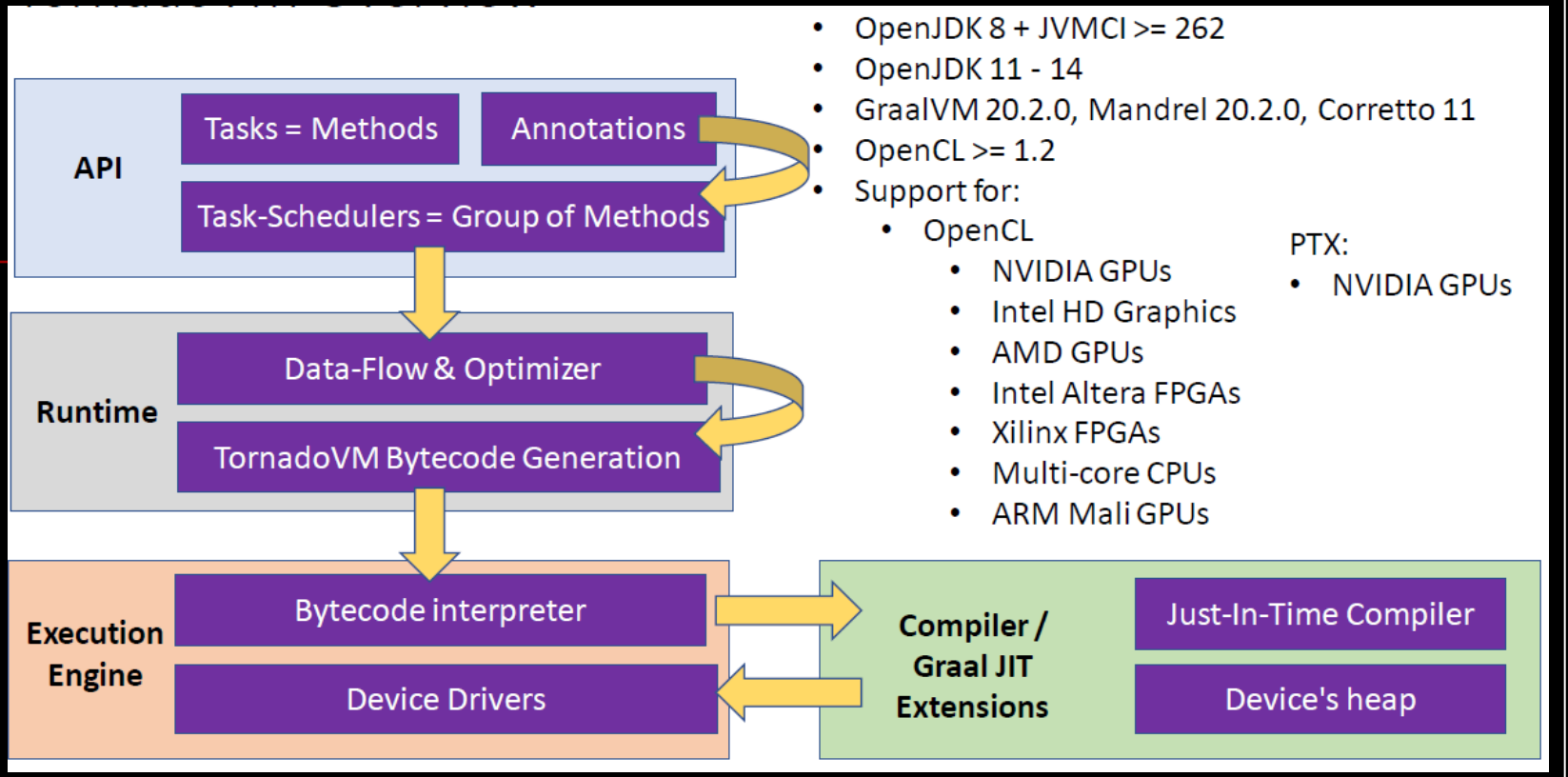
Example



Source: https://jjfumero.github.io/talks/



Source: https://jjfumero.github.io/talks/

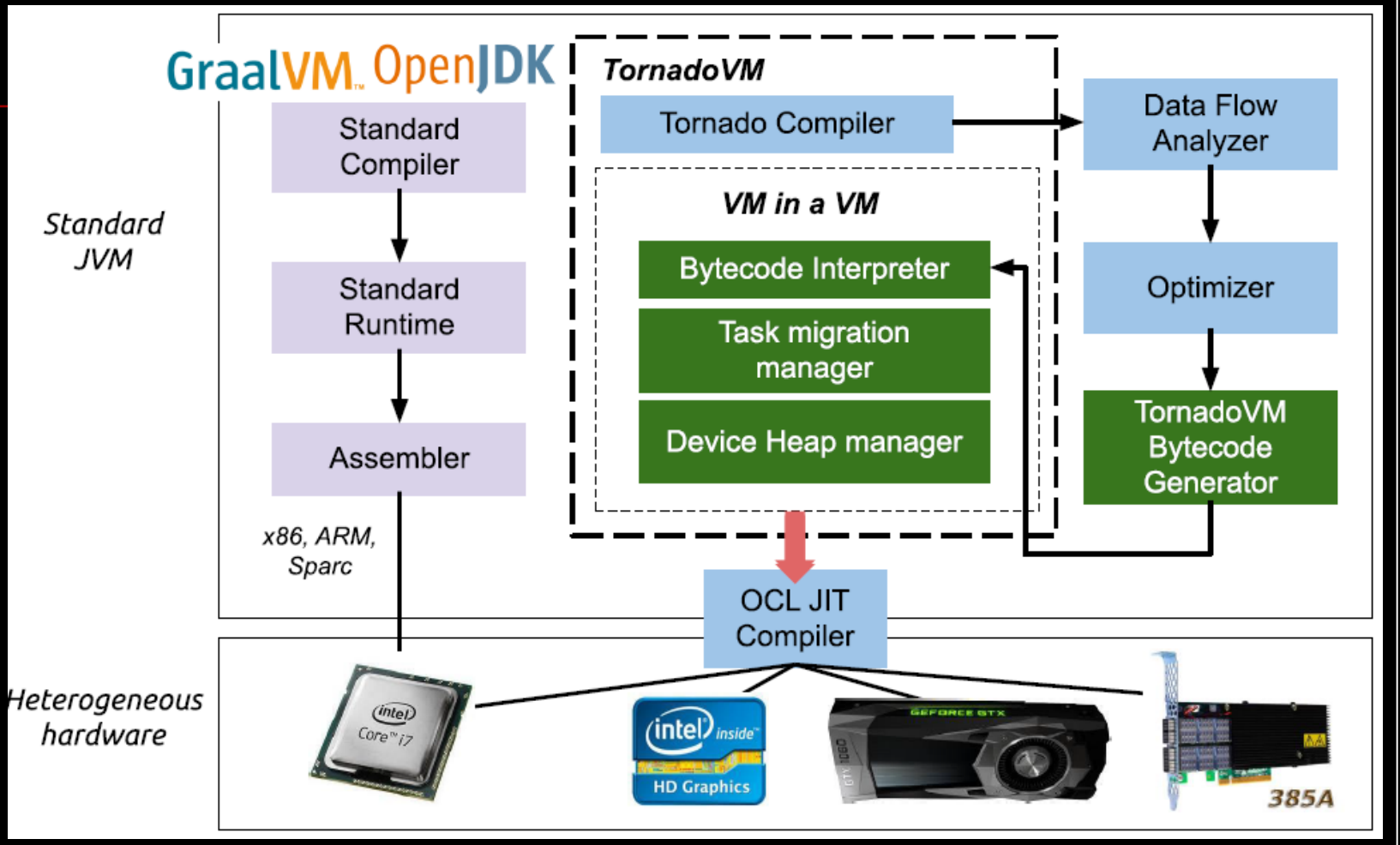


Source: https://jjfumero.github.io/talks/

...

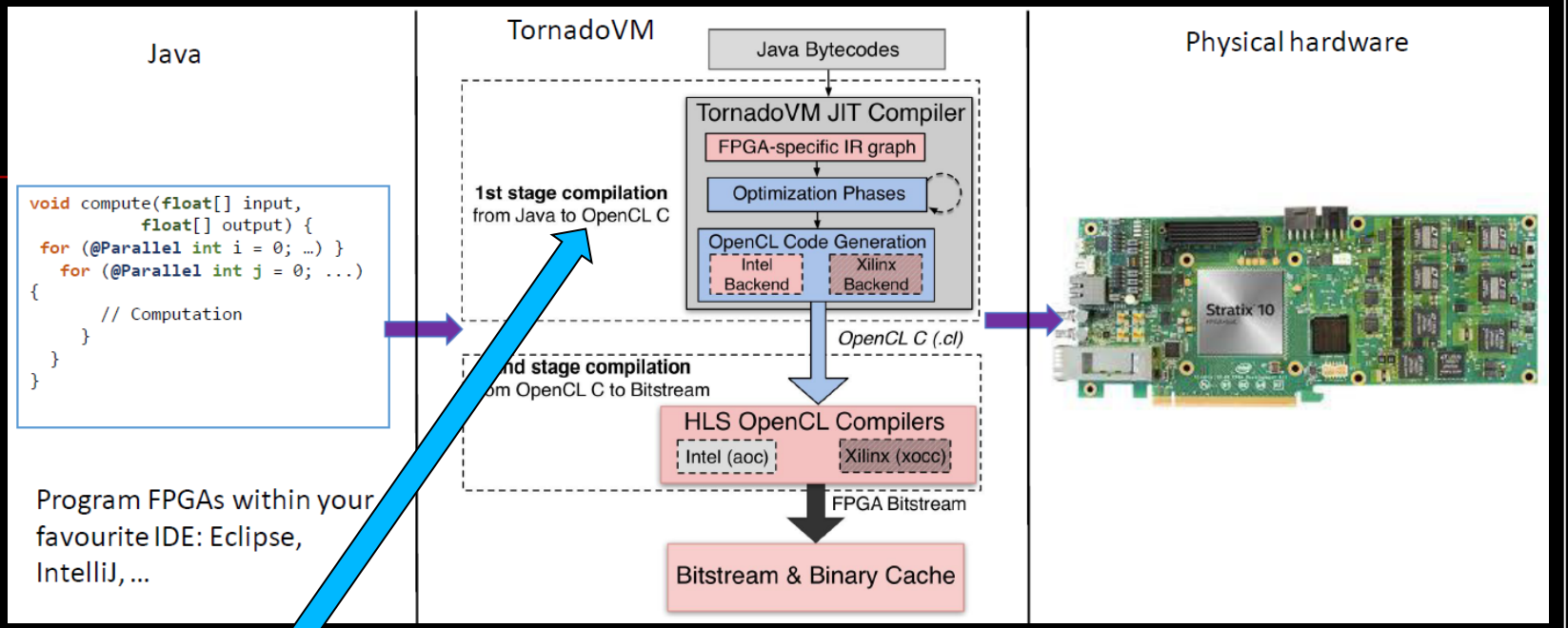
1.7 Workflow

- VM in VM



Source: https://jjfumero.github.io/talks/

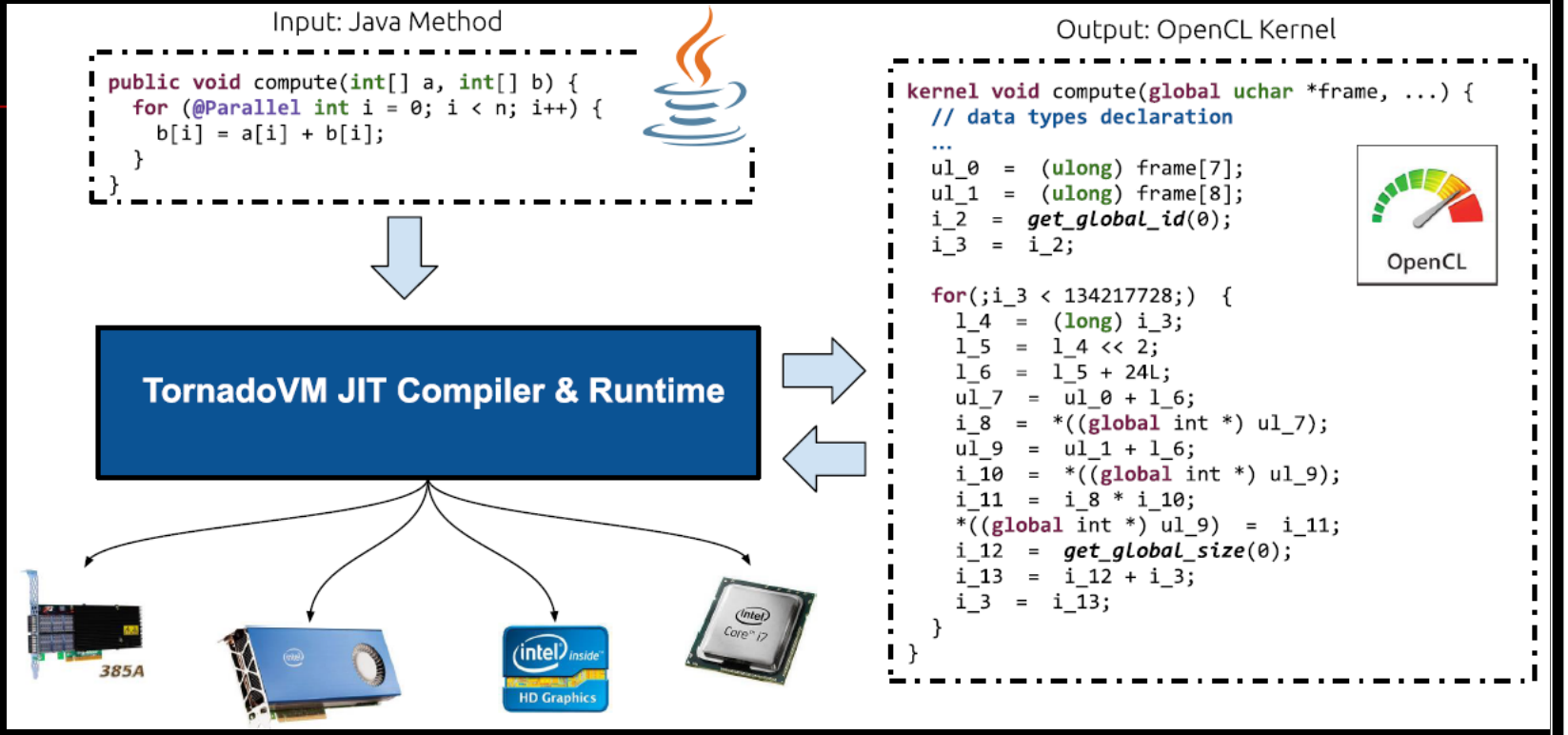
FPGA Workflow

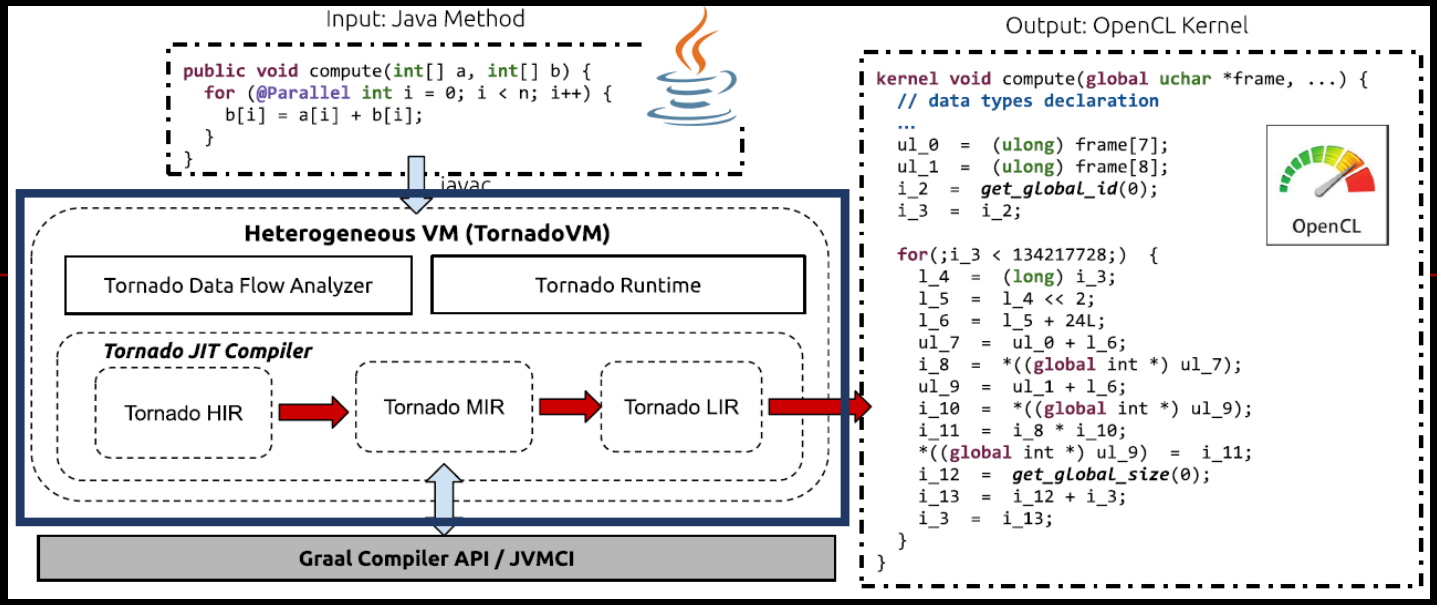


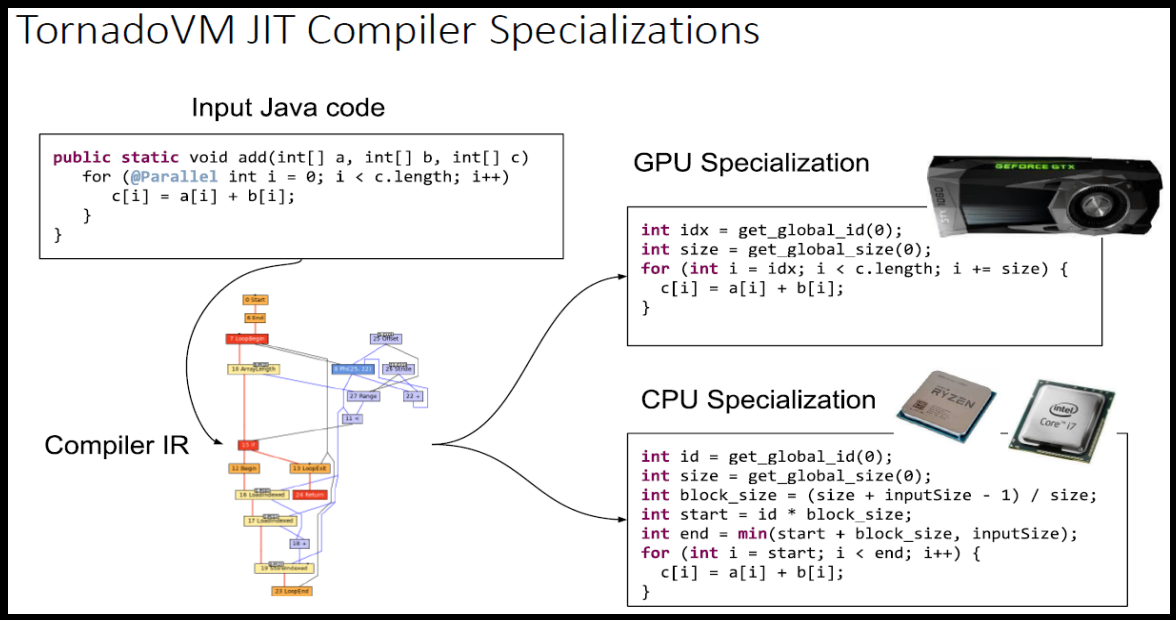
DSLs as new OpenCL kernel language in the future…

1.8 Compiler & Runtime Overview

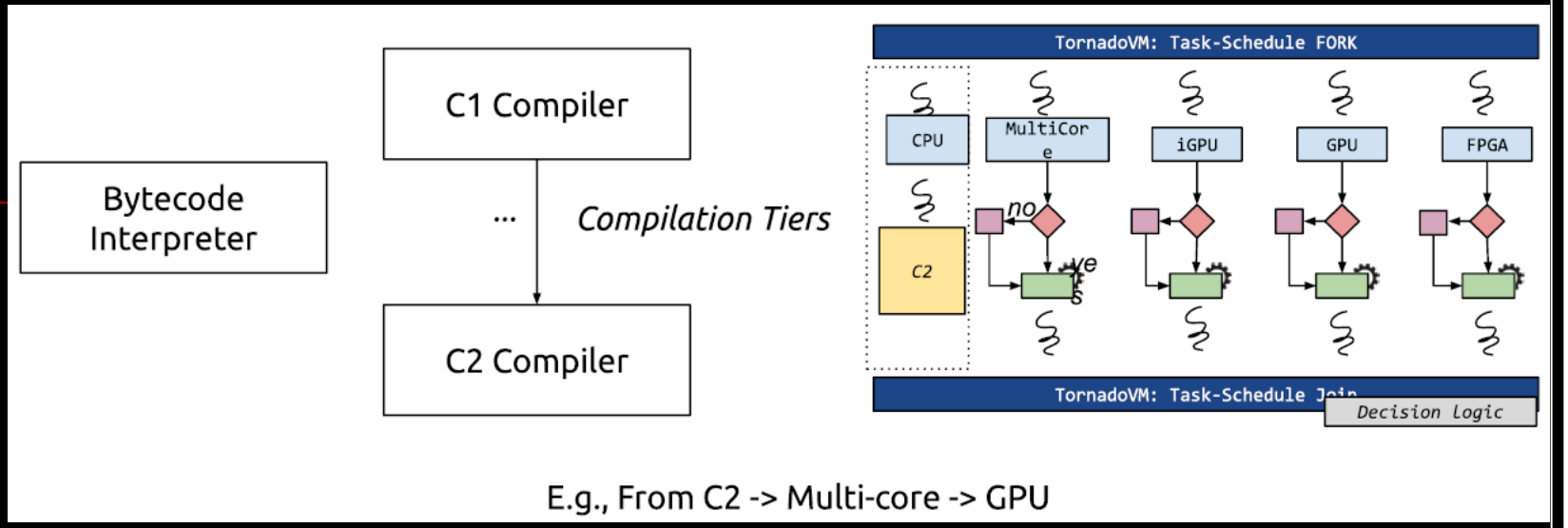
- https://jjfumero.github.io/talks




Input: Java Method
public void compute(int[] a, int[] b) {
for (@Parallel int i = 0; i < n; i++) {
b[i] = a[i] + b[i];
}
}
javac
Heterogeneous VM (TornadoVM)
Tornado Data Flow Analyzer
Tornado Runtime
Tornado JIT Compiler
Tornado HIR
Tornado MIR
Tornado LIR
Graal Compiler API / JVMCI
Output: OpenCL Kernel
kernel void compute(global uchar *frame, ...) {
// data types declaration
...
ul_0 = (ulong) frame[7];
ul_1 = (ulong) frame[8];
i_2 = get_global_id(0);
i_3 = i_2;
for(;i_3 < 134217728;) {
l_4 = (long) i_3;
l_5 = l_4 << 2;
l_6 = l_5 + 24L;
ul_7 = ul_0 + l_6;
i_8 = *((global int *) ul_7);
ul_9 = ul_1 + l_6;
i_10 = *((global int *) ul_9);
i_11 = i_8 * i_10;
*((global int *) ul_9) = i_11;
i_12 = get_global_size(0);
i_13 = i_12 + i_3;
i_3 = i_13;
}
}
OpenCL


TornadoVM JIT Compiler Specializations
Input Java code
public static void add(int[] a, int[] b, int[] c)
for (@Parallel int i = 0; i < c.length; i++)
c[i] = a[i] + b[i];
}
}
Compiler IR
GPU Specialization
int idx = get_global_id(0);
int size = get_global_size(0);
for (int i = idx; i < c.length; i += size) {
c[i] = a[i] + b[i];
}
CPU Specialization
int id = get_global_id(0);
int size = get_global_size(0);
int block_size = (size + inputSize - 1) / size;
int start = id * block_size;
int end = min(start + block_size, inputSize);
for (int i = start; i < end; i++) {
c[i] = a[i] + b[i];
}

New compilation tier for Heterogeneous Systems



Source: https://jjfumero.github.io/talks/

- ...

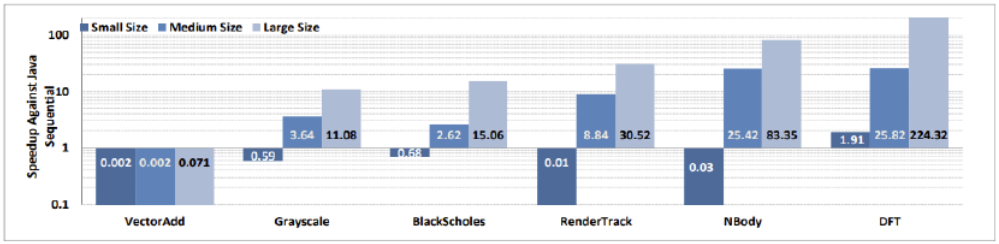
1.9 Official Demos

- https://github.com/jjfumero/nyjavasig-tornadovm-2021

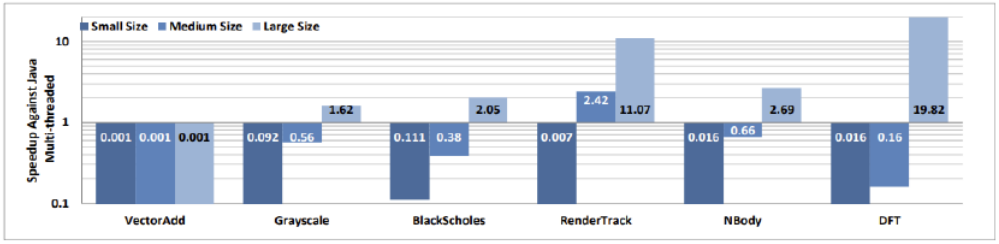
- Demo 1: DFT (same as before)
- Demo 2: BlackScholes (Kernel based on Fintech algorithms)

Source: https://jjfumero.github.io/talks/

Results for FPGAs



A) Speedup against Sequential Java Hotspot. 224x compared to Java Hotspot

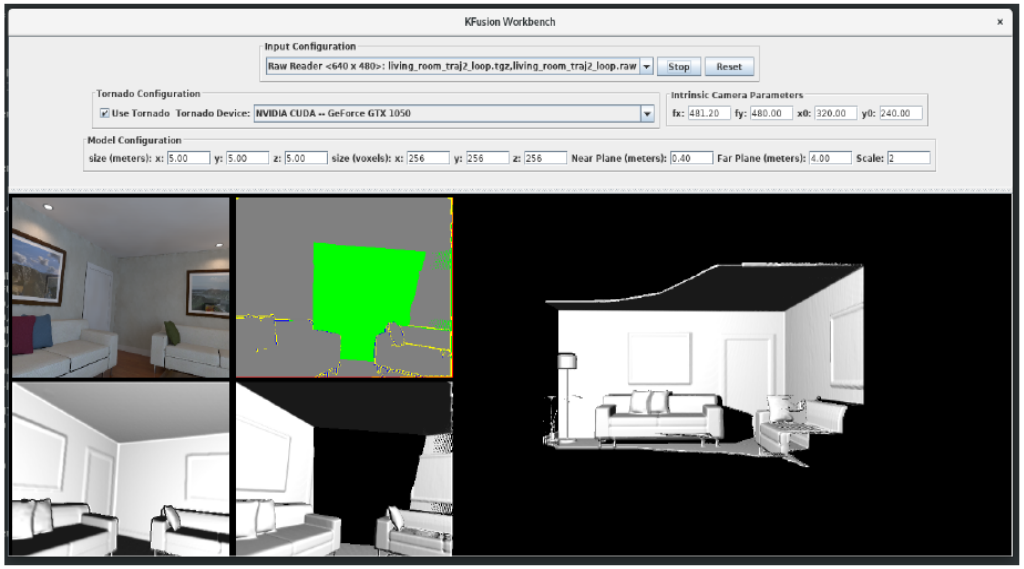


B) Speedup Against Java Multithread (8 threads) ~20x compared to Java multi-threaded

Michail Papadimitriou, Juan Fumero, Athanasios Stratikopoulos, Foivos S. Zakkak, Christos Kotselidis. ***Transparent Compiler and Runtime Specializations for Accelerating Managed Languages on FPGAs.*** Programming 2021 - Volume 5 - Issue 2

Source: https://jjfumero.github.io/talks/

- https://github.com/beehive-lab/kfusion-tornadovm



* Computer Vision Application
* ~7K LOC
* Thousands of OpenCL LOC generated.

Source: https://jjfumero.github.io/talks/

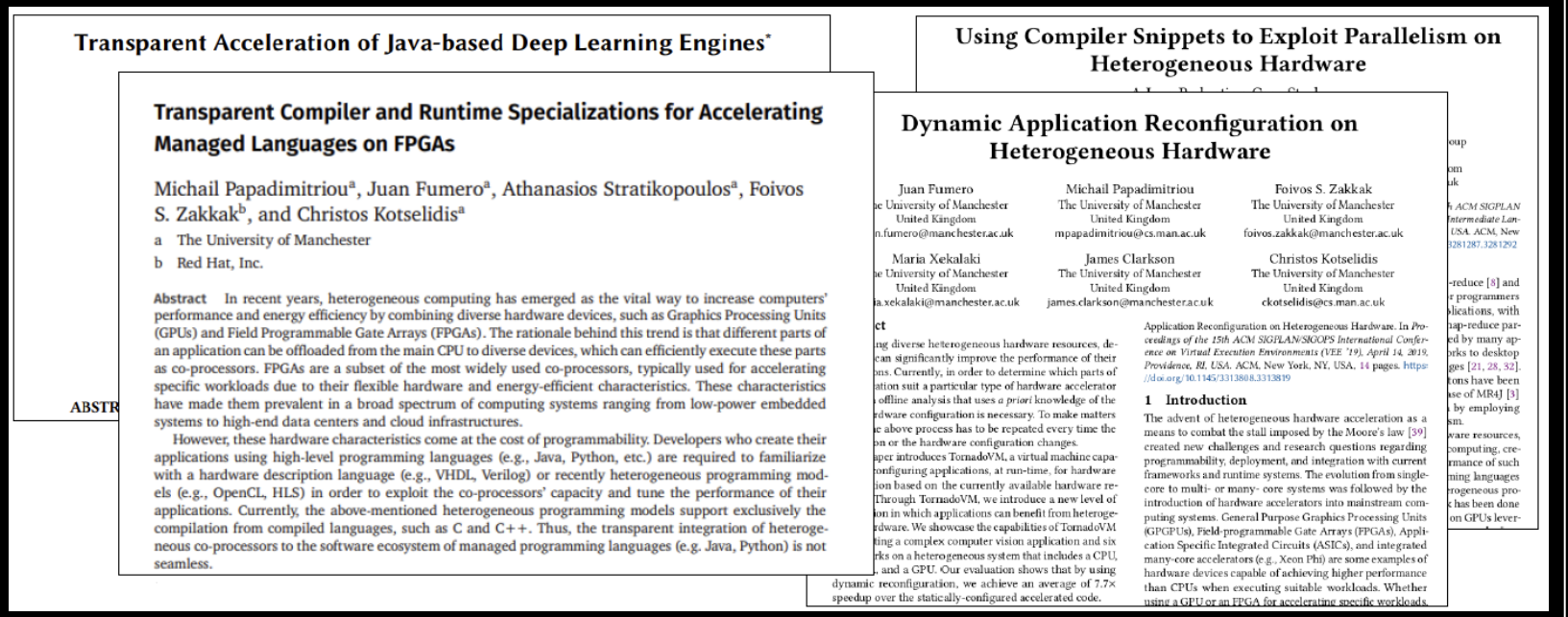
- https://github.com/jjfumero/qconlondon2020-tornadovm

  Running Matrix Multiplication & Node.js example
- https://github.com/jjfumero/nyjavasig-tornadovm-2021

  Live Task Migration –Server/Client App
- https://e2data.eu/blog/java-gpu-accelerated-viola-jones-face-detection-with-tornadovm
- …

1.10 Good Resources

- https://jjfumero.github.io/talks/
- https://github.com/beehive lab/TornadoVM/blob/master/assembly/src/docs/14_PUBLICATIONS.md



Transparent Acceleration of Java-based Deep Learning Engines*

Transparent Compiler and Runtime Specializations for Accelerating Managed Languages on FPGAs

Michail Papadimitriou[a], Juan Fumero[a], Athanasios Stratikopoulos[a], Foivos S. Zakkak[b], and Christos Kotselidis[a]

a The University of Manchester

b Red Hat, Inc.

Abstract In recent years, heterogeneous computing has emerged as the vital way to increase computers' performance and energy efficiency by combining diverse hardware devices, such as Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs). The rationale behind this trend is that different parts of an application can be offloaded from the main CPU to diverse devices, which can efficiently execute these parts as co-processors. FPGAs are a subset of the most widely used co-processors, typically used for accelerating specific workloads due to their flexible hardware and energy-efficient characteristics. These characteristics have made them prevalent in a broad spectrum of computing systems ranging from low-power embedded systems to high-end data centers and cloud infrastructures.

However, these hardware characteristics come at the cost of programmability. Developers who create their applications using high-level programming languages (e.g., Java, Python, etc.) are required to familiarize with a hardware description language (e.g., VHDL, Verilog) or recently heterogeneous programming models (e.g., OpenCL, HLS) in order to exploit the co-processors' capacity and tune the performance of their applications. Currently, the above-mentioned heterogeneous programming models support exclusively the compilation from compiled languages, such as C and C++. Thus, the transparent integration of heterogeneous co-processors to the software ecosystem of managed programming languages (e.g. Java, Python) is not seamless.

Using Compiler Snippets to Exploit Parallelism on Heterogeneous Hardware

Dynamic Application Reconfiguration on Heterogeneous Hardware

Juan Fumero, The University of Manchester, United Kingdom

Michail Papadimitriou, The University of Manchester, United Kingdom, mpapadimitriou@cs.man.ac.uk

Foivos S. Zakkak, The University of Manchester, United Kingdom, foivos.zakkak@manchester.ac.uk

Maria Xekalaki, The University of Manchester, United Kingdom

James Clarkson, The University of Manchester, United Kingdom, james.clarkson@manchester.ac.uk

Christos Kotselidis, The University of Manchester, United Kingdom, ckotselidis@cs.man.ac.uk

Application Reconfiguration on Heterogeneous Hardware. In *Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE '19), April 14, 2019, Providence, RI, USA.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3313808.3313819

1 Introduction

The advent of heterogeneous hardware acceleration as a means to combat the stall imposed by the Moore's law [39] created new challenges and research questions regarding programmability, deployment, and integration with current frameworks and runtime systems. The evolution from single-core to multi- or many- core systems was followed by the introduction of hardware accelerators into mainstream computing systems. General Purpose Graphics Processing Units (GPGPUs), Field-programmable Gate Arrays (FPGAs), Application Specific Integrated Circuits (ASICs), and integrated many-core accelerators (e.g., Xeon Phi) are some examples of hardware devices capable of achieving higher performance than CPUs when executing suitable workloads. Whether using a GPU or an FPGA for accelerating specific workloads

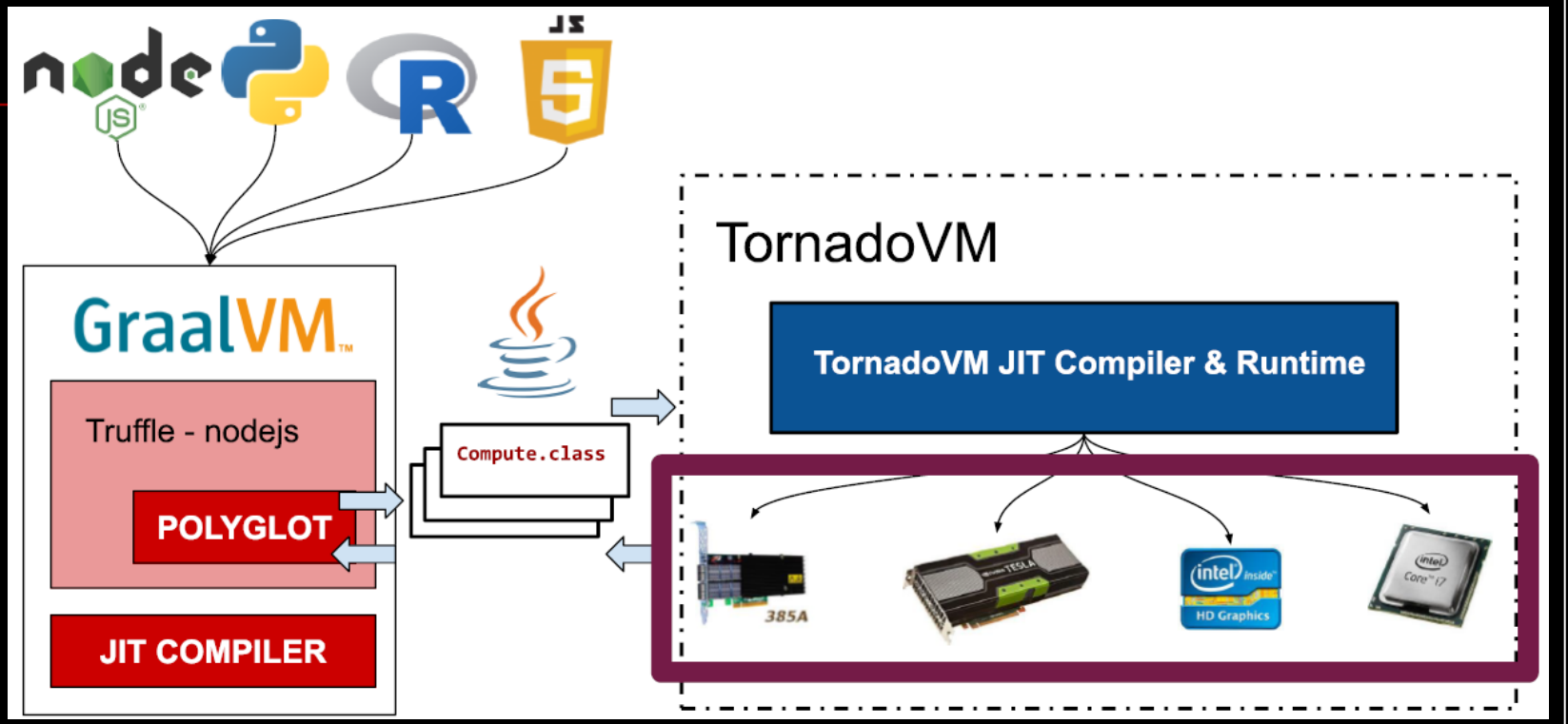
- https://github.com/beehive lab/TornadoVM/blob/master/assembly/src/docs/16_RESOURCES.md
- https://www.infoq.com/articles/tornadovm-java-gpu-fpga/
- …

1.11 Related Work

Project	Production-Ready	Supported Devices	Live Task Migration	Compiler Specializations	Dynamic Languages
Sumatra	No	AMD GPUs	No	No	No
Marawacc	No	Multi-core, GPUs	No	Yes	Yes
JaBEE	No	NVIDIA GPUs	No	No	No
RootBeer	No	NVIDIA GPUs	No	No	No
Aparapi	Yes	GPUs, multi-core	No	No	No (*)
IBM GPU J9	Yes	NVIDIA GPUs	No	No	No
grCUDA	No (*)	NVIDIA GPUs	No	No	Yes
TornadoVM	No (*)	Multi-core, GPUs,FPGAs	Yes	Yes	Yes

Source: https://jjfumero.github.io/talks/

2) Acceleration of JVM Apps
2.1 TornadoVM & Dynamic/Untyped Languages



Source: https://jjfumero.github.io/talks/

Polyglot

```
public class MyCompute {
  public static float[] compute() {
    new TaskSchedule("s0")
      .streamIn(a, b)
      .task("t0", obj::compute, a, b, c)
      .streamOut(c)
      .execute();
    return c;
  }
}
```

```
myCompute <- java.type('MyCompute');
output <- myCompute.compute();
```

```
import java
myclass = java.type('MyCompute')
output = myclass.compute()
```

```
var myclass = Java.type('MyCompute')
var output = myclass.compute()
```

Code reusability from high-level Programming Languages thanks to Truffle Polyglot

Live demo with NodeJS -> then Check the talk for QConLondon 2020:
https://www.infoq.com/presentations/tornadovm-java-gpu-fpga/

Source: https://jjfumero.github.io/talks/

3) Speed up data processing

3.1 E2Data

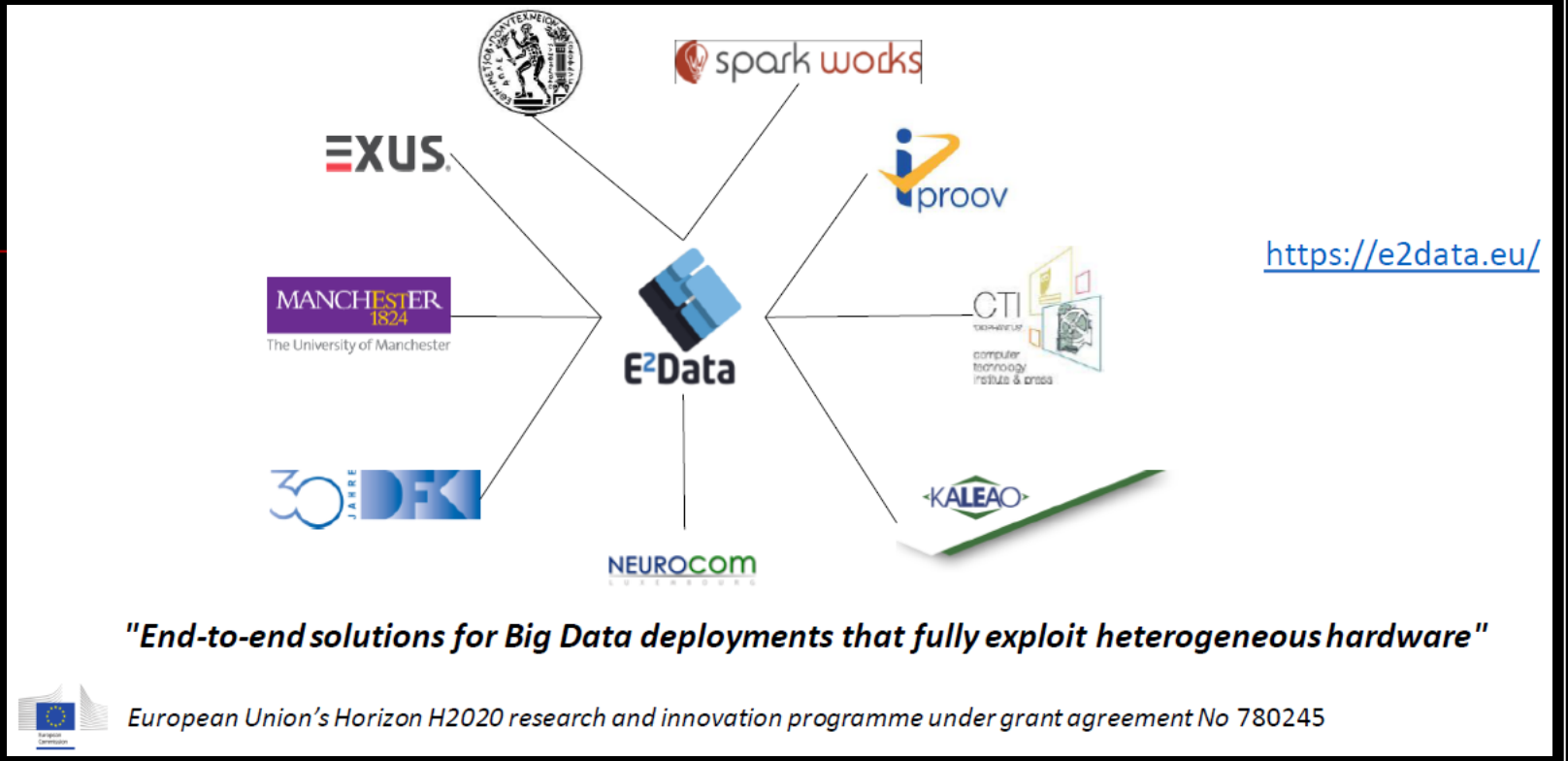
- https://e2data.eu/

In the current paradigm, when a company needs to run a critical component of their business logic at optimal speed in the cloud, they have three options:

1. Scale-up (by upgrading processors at the node level)
2. Scale-out (by adding nodes)
3. Manually implement code optimisations specific to the underlying hardware.

E2Data proposes an end-to-end solution for Big Data deployments that will fully exploit and advance the state-of-the-art in infrastructure services by delivering more performance from fewer resources. The E2Data stack will achieve this by dynamically profiling, compiling and optimising code for execution on chosen devices, such as CPUs, GPUs, FPGAs, and others. By removing the need for developers to craft specific device code in languages like CUDA or OpenCL, E2Data will create substantial savings in developer time, while still exploiting the power of diverse device architectures, such as are currently offered by Microsoft, Amazon and others.

Over the course of the project, testing will be conducted on both high-performing x86 and low-power ARM cluster architectures representing realistic execution scenarios in four resource-demanding applications : finance, healthcare, green architecture, and security.

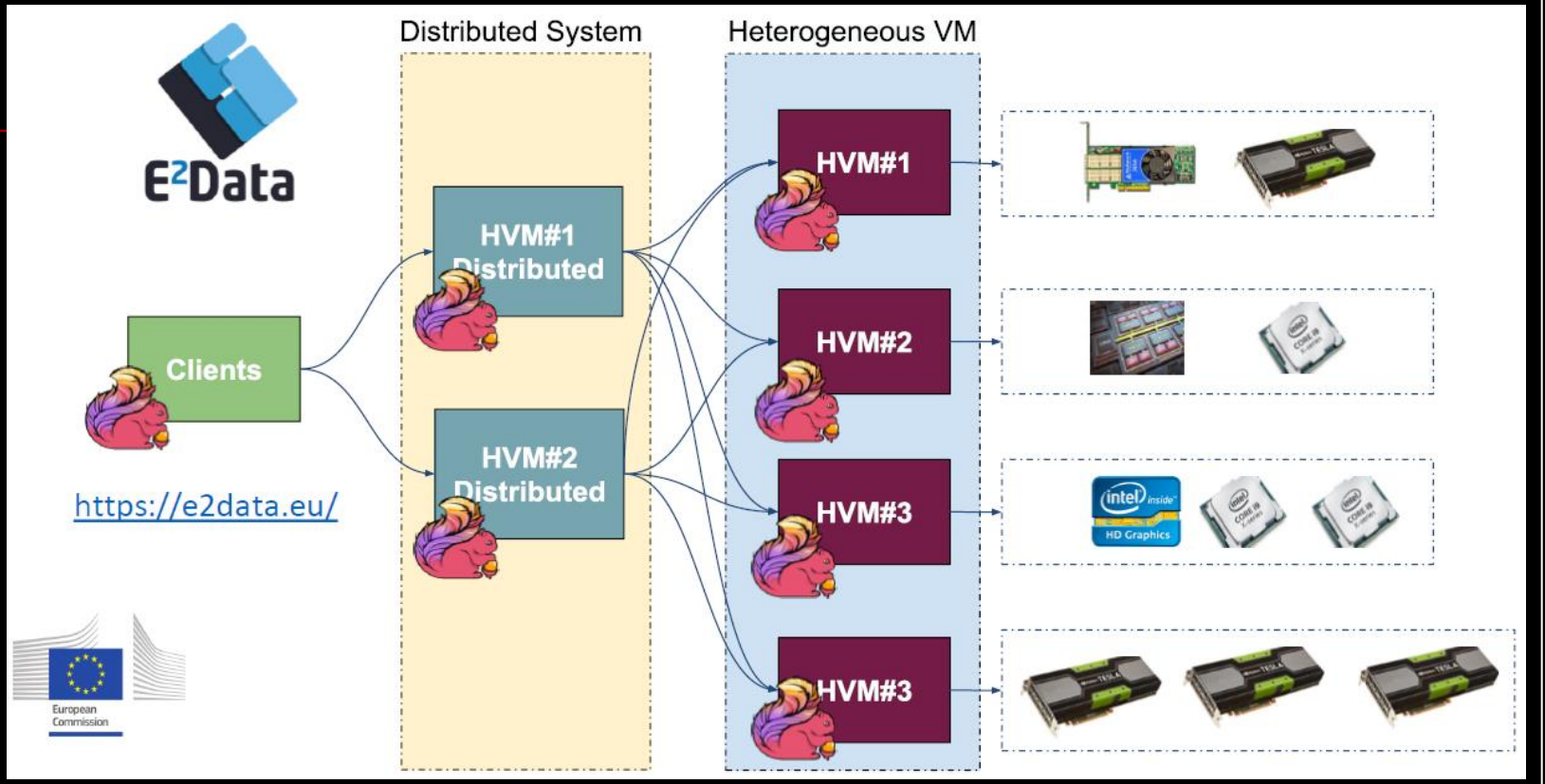


Source: https://jjfumero.github.io/talks/

- **https://github.com/E2Data**
- **…**

Architecture

- **Distributed H. System with Apache Flink & TornadoVM**



Source: https://jjfumero.github.io/talks/

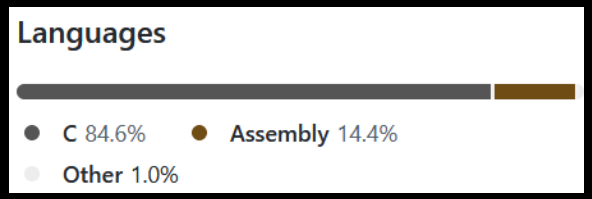
4) Mambo and ProtonVM

4.1 Mambo

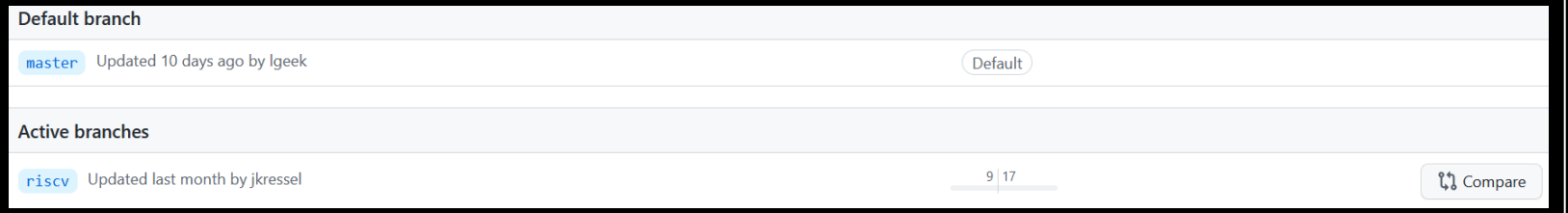
- https://github.com/beehive-lab/mambo

~~A low-overhead dynamic binary instrumentation and modification tool~~ for ARM (now with both AArch32 and AArch64 support).

- Src



https://github.com/beehive-lab/mambo/branches



- 2021-09-21: We've released a partial port of MAMBO to RISC-V in the riscv branch, where development is continuing. The initial porting was done by Guillermo Callaghan and Cosmin Gorgovan.

...

Performance

System	DBM configuration	SPEC CPU2006 SPECint	SPECfp	CPU
Pine A64+	DynamoRIO	48.8%	14.4%	27.5%
	MAMBO	23.8%	6.4%	13.3%
	Contiguous Traces	17.5%	4.7%	9.8%
	MAMBO RAIBI	20.2%	5.3%	11.2%
	MAMBO TRIBI 1	16.2%	4.4%	9.3%
	MAMBO TRIBI 6	**16.2%**	**4.0%**	**8.9%**
Applied Micro X-Gene 2	DynamoRIO	60.4%	15.7%	32.4%
	MAMBO	36.5%	10.2%	20.4%
	Contiguous Trace	22.3%	5.4%	12.1%
	MAMBO RAIBI	22.7%	4.7%	11.8%
	MAMBO TRIBI 1	21.4%	5.0%	11.5%
	MAMBO TRIBI 6	**20.6%**	**4.2%**	**10.7%**
Jetson TX1	DynamoRIO	58.7%	15.0%	31.4%
	MAMBO	34.6%	9.6%	19.3%
	Contiguous Trace	20.2%	5.3%	11.2%
	MAMBO RAIBI	23.3%	4.8%	12.1%
	MAMBO TRIBI 1	20.0%	4.4%	10.6%
	MAMBO TRIBI 6	**18.7%**	**3.6%**	**9.6%**
Rock 960	DynamoRIO	62.7%	18.5%	35.1%
	MAMBO	25.3%	7.2%	14.4%
	Contiguous Trace	17.2%	4.1%	9.3%
	MAMBO RAIBI	16.7%	3.5%	8.8%
	MAMBO TRIBI 1	14.9%	2.7%	7.6%
	MAMBO TRIBI 6	**14.3%**	**2.5%**	**7.2%**
Hikey 960	DynamoRIO	47.0%	12.2%	25.5%
	MAMBO	23.3%	4.8%	12.1%
	Contiguous Trace	19.1%	4.0%	10.0%
	MAMBO RAIBI	23.4%	4.1%	11.7%
	MAMBO TRIBI 1	**16.1%**	**2.2%**	**7.7%**
	MAMBO TRIBI 6	16.4%	2.6%	8.1%

The lowest overhead for each benchmark appears in **bold**.

Source: https://www.research.manchester.ac.uk/portal/files/160211940/callaghan_MAMBO_VEE2020_authorsversion.pdf

Considering new use case

- **How to leverage Mambo in TornadoVM?**

4.2 ProtonVM

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

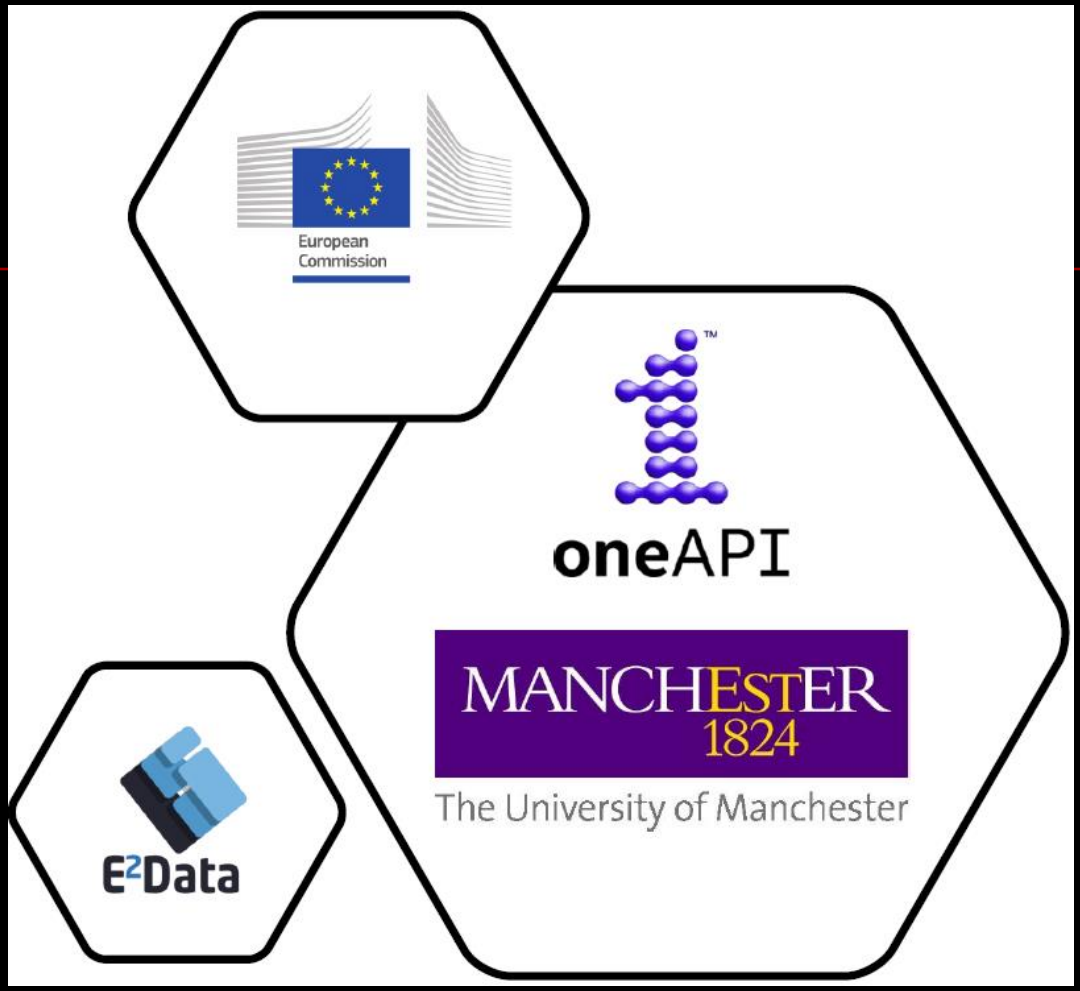
Parallel Bytecode Interpreter For Heterogeneous Hardware.

In a nutshell, ProtoVM is a tiny Parallel Java Bytecode Interpreter (BC) implemented in OpenCL capable of executing on heterogeneous architectures (multi-core CPUs, GPUs and FPGAs).

ProtonVM is a proof-of-concept for running a subset of the Java bytecodes in C++ and OpenCL. The bytecodes defined correspond to an extension of the small-subset of Java bytecode explained by Terence Parr, from the University of San Francisco of how to build a simple Virtual Machine. This project extends this simple bytecode interpreter to study the feasibility of running, as efficiently as possible, parallel bytecode interpreters on heterogeneous computer architectures.

ProtonVM currently runs integer arithmetic only. A possible extension is to include `fp32` and `fp64` math arithmetic and support complex math operations. This way ProtonVM could take advantage of the compute-power of modern GPUs, and DSPs available on FPGAs.

ProtonVM has been presented at MoreVMs 2020 and it has been executed on NVIDIA GPUs, Intel HD Graphics and Xilinx FPGAs.



Source: https://jjfumero.github.io/talks/

...

Design

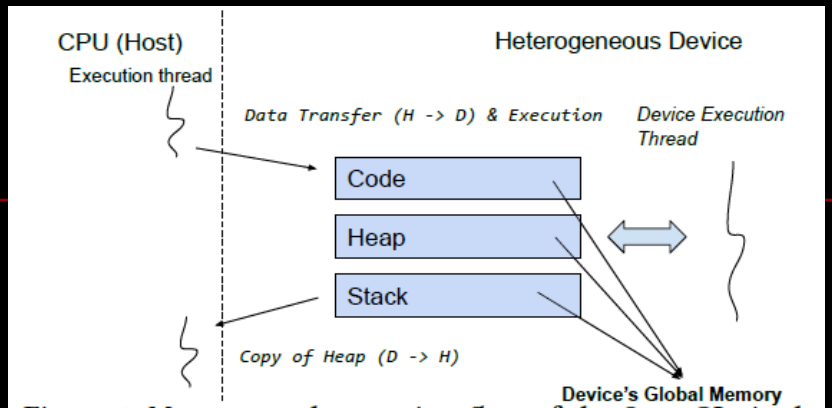


Figure 1: Memory and execution flow of the OpenCL single threaded interpreter.

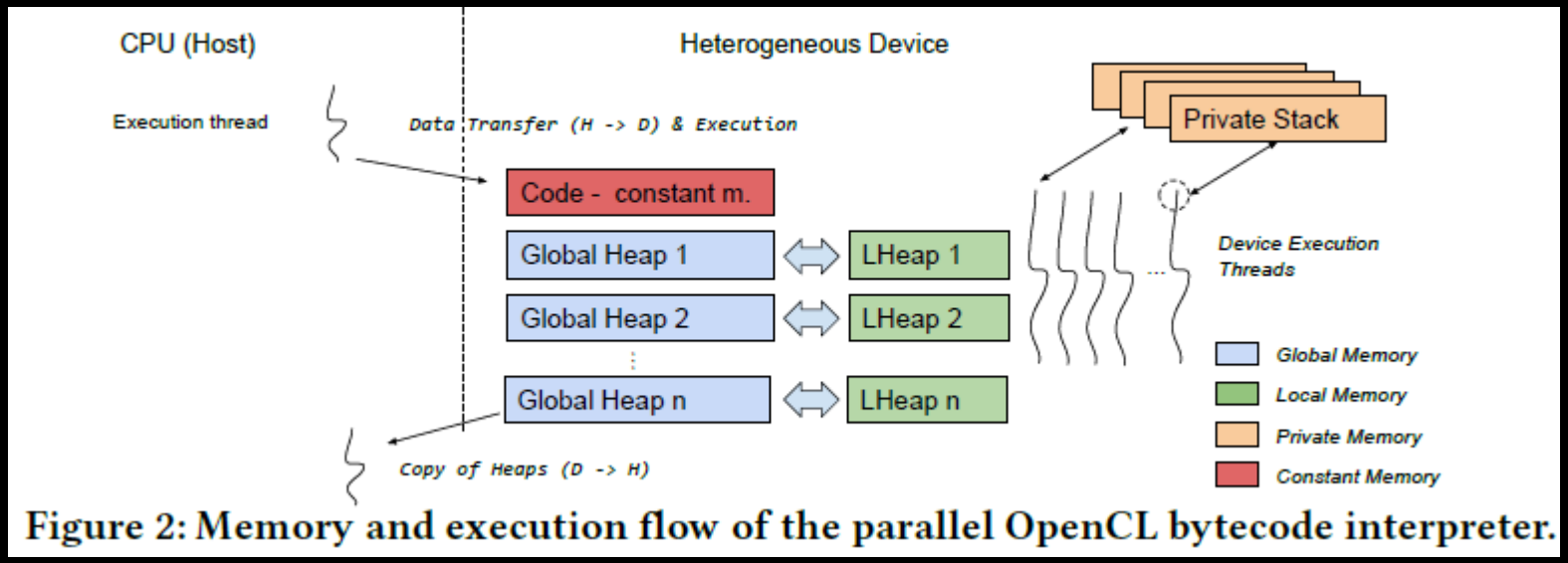


Figure 2: Memory and execution flow of the parallel OpenCL bytecode interpreter.

Source: https://jjfumero.github.io/files/JuanFumero-MoreVMs2020-Preprint.pdf

Considering future improvement

- Re-implement with SYCL or OpenCL C++?
- Use in Serverless scenarios?

5) Practice
5.1 Getting Started

- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/1_INSTALL.md

Supported Platforms

The following table includes the platforms that TornadoVM can be executed.

OS	OpenCL Backend	PTX Backend	SPIR-V Backend
CentOS >= 7.3	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Level-Zero >= 1.1.2
Fedora >= 21	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Level-Zero >= 1.1.2
Ubuntu >= 16.04	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Level-Zero >= 1.1.2
Mac OS X Mojave 10.14.6	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Not supported
Mac OS X Catalina 10.15.3	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Not supported
Mac OS X Big Sur 11.5.1	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Not supported
Windows 10	OpenCL for GPUs and CPUs >= 1.2, OpenCL for FPGAs >= 1.0)	CUDA 9.0+	Not supported/tested

Note: The SPIR-V backend is only supported for Linux OS. Besides, the SPIR-V backend with Level Zero runs on Intel HD Graphics (integrated GPUs).

- TornadoVM can be built with three compiler backends and is able to generate OpenCL, PTX and SPIR-V code.

Important [SPIR-V Backend Configuration] Prior to the built with the SPIR-V backend, users have to ensure that Level Zero is installed in their system. Please follow the guidelines here.

At least one backend must be specified at build time to the `make` command:

```
$ make BACKENDS=opencl,ptx,spirv
```

Install with GraalVM 21.3.0

- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/10_INSTALL_WITH_GRAALVM.md

Pre-requisites

- Maven Version 3.6.3
- CMake 3.6 (or newer)
- At least one of:
  - OpenCL: GPUs and CPUs >= 1.2, FPGAs >= 1.0
  - CUDA 9.0 +
- GCC or clang/LLVM (GCC >= 5.5)
- Python (>= 2.7)

...

- Example for GraalVM based on JDK 17:

```
$ wget https://github.com/graalvm/graalvm-ce-builds/releases/download/vm-21.3.0/graalvm-ce-java17-linux-amd64-21.3.0.tar.gz
$ tar -xf graalvm-ce-java17-linux-amd64-21.3.0.tar.gz
```

...

The first time you need to create the `etc/sources.env` file and add the following code in it **(after updating the paths to your correct ones)**:

```
#!/bin/bash
export JAVA_HOME=<path to GraalVM 21.3.0 jdk> ## This path is produced in Step 1
export PATH=$PWD/bin/bin:$PATH    ## This directory will be automatically generated during Tornado compilation
export TORNADO_SDK=$PWD/bin/sdk   ## This directory will be automatically generated during Tornado compilation
export CMAKE_ROOT=/usr            ## or <path/to/cmake/cmake-3.10.2> (see step 4)
```

This file should be loaded once after opening the command prompt for the setup of the required paths:

```
$ source ./etc/sources.env
```

```
$ cd ~/tornadovm
$ . etc/sources.env
```

To build with GraalVM and JDK 11 and JDK 17:

```
$ make graal-jdk-11-plus BACKEND={ptx,opencl}
```

...

- **RPi4 is obviously not supported byTornadoVM, but let's take a try!**

Test Env

```
[mydev@fedora /]$ uname -a
Linux fedora 5.15.7-100.fc34.aarch64 #1 SMP Wed Dec 8 19:00:11 UTC 2021 aarch64 aarch64 aarch64 GNU/Linux
[mydev@fedora /]$
[mydev@fedora /]$ gcc -v
Using built-in specs.
COLLECT_GCC=gcc
COLLECT_LTO_WRAPPER=/usr/libexec/gcc/aarch64-redhat-linux/11/lto-wrapper
Target: aarch64-redhat-linux
Configured with: ../configure --enable-bootstrap --enable-languages=c,c++,fortran,objc,obj-c++,ada,go,lto --prefix=/usr --mand
ir=/usr/share/man --infodir=/usr/share/info --with-bugurl=http://bugzilla.redhat.com/bugzilla --enable-shared --enable-threads
=posix --enable-checking=release --enable-multilib --with-system-zlib --enable-__cxa_atexit --disable-libunwind-exceptions --e
nable-gnu-unique-object --enable-linker-build-id --with-gcc-major-version-only --with-linker-hash-style=gnu --enable-plugin --
enable-initfini-array --with-isl=/builddir/build/BUILD/gcc-11.2.1-20210728/obj-aarch64-redhat-linux/isl-install --enable-gnu-i
ndirect-function --build=aarch64-redhat-linux
Thread model: posix
Supported LTO compression algorithms: zlib zstd
gcc version 11.2.1 20210728 (Red Hat 11.2.1-1) (GCC)
[mydev@fedora /]$
[mydev@fedora /]$ clang -v
clang version 12.0.1 (Fedora 12.0.1-1.fc34)
Target: aarch64-unknown-linux-gnu
Thread model: posix
InstalledDir: /bin
Found candidate GCC installation: /bin/../lib/gcc/aarch64-redhat-linux/11
Found candidate GCC installation: /usr/lib/gcc/aarch64-redhat-linux/11
Selected GCC installation: /usr/lib/gcc/aarch64-redhat-linux/11
Candidate multilib: .;@m64
Selected multilib: .;@m64
[mydev@fedora /]$
```

```
[mydev@fedora /]$ mvn -v
Apache Maven 3.6.3 (Red Hat 3.6.3-8)
Maven home: /usr/share/maven
Java version: 17.0.1, vendor: GraalVM Community, runtime: /opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java17-21.3.0
Default locale: en_US, platform encoding: UTF-8
OS name: "linux", version: "5.15.7-100.fc34.aarch64", arch: "aarch64", family: "unix"
[mydev@fedora /]$
[mydev@fedora /]$ cmake --version
cmake version 3.20.5

CMake suite maintained and supported by Kitware (kitware.com/cmake).
[mydev@fedora /]$
[mydev@fedora /]$ python -V
Python 3.9.9
[mydev@fedora /]$
[mydev@fedora /]$ java --version
openjdk 17.0.1 2021-10-19
OpenJDK Runtime Environment GraalVM CE 21.3.0 (build 17.0.1+12-jvmci-21.3-b05)
OpenJDK 64-Bit Server VM GraalVM CE 21.3.0 (build 17.0.1+12-jvmci-21.3-b05, mixed mode, sharing)
[mydev@fedora /]$
```

edit etc/sources.env

```
[mydev@fedora TornadoVM-master]$ cat etc/sources.env
#export JAVA_HOME=/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java17-21.3.0
export PATH=$PWD/bin/bin:$PATH
export TORNADO_SDK=$PWD/bin/sdk
#export CMAKE_ROOT=
[mydev@fedora TornadoVM-master]$
```

```
[mydev@fedora TornadoVM-master]$ source etc/sources.env
```

start building

```
[mydev@fedora TornadoVM-master]$ make graal-jdk-11-plus BACKEND=opencl
./bin/compile.sh graal-jdk-11-plus opencl
mvn -Popencl-backend,ptx-backend,spirv-backend clean
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for tornado:tornado-runtime:jar:0.13-dev
[WARNING] The expression ${version} is deprecated. Please use ${project.version} instead.
[WARNING]
[WARNING] Some problems were encountered while building the effective model for tornado:tornado-matrices:jar:0.13-dev
[WARNING] The expression ${version} is deprecated. Please use ${project.version} instead.
[WARNING]
[WARNING] Some problems were encountered while building the effective model for tornado:tornado-drivers-common:jar:0.13-dev
[WARNING] The expression ${version} is deprecated. Please use ${project.version} instead.
[WARNING]
[WARNING] Some problems were encountered while building the effective model for tornado:tornado-drivers-opencl:jar:0.13-dev
[WARNING] The expression ${version} is deprecated. Please use ${project.version} instead.
[WARNING]
[WARNING] Some problems were encountered while building the effective model for tornado:tornado-drivers-opencl-jni:jar:0.13-de
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for tornado 0.13-dev:
[INFO]
[INFO] tornado ............................................ SUCCESS [01:15 min]
[INFO] tornado-api ........................................ SUCCESS [ 45.299 s]
[INFO] tornado-runtime .................................... SUCCESS [ 14.948 s]
[INFO] tornado-matrices ................................... SUCCESS [ 12.863 s]
[INFO] tornado-drivers .................................... SUCCESS [  3.742 s]
[INFO] tornado-drivers-common ............................. SUCCESS [ 32.870 s]
[INFO] tornado-drivers-opencl ............................. SUCCESS [ 18.293 s]
[INFO] tornado-drivers-opencl-jni ......................... SUCCESS [01:02 min]
[INFO] tornado-examples ................................... SUCCESS [ 41.436 s]
[INFO] tornado-benchmarks ................................. SUCCESS [ 52.628 s]
[INFO] tornado-unittests .................................. SUCCESS [ 54.370 s]
[INFO] tornado-annotation ................................. SUCCESS [  6.544 s]
[INFO] tornado-assembly ................................... SUCCESS [01:58 min]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  05:21 min (Wall Clock)
[INFO] Finished at: 2021-12-17T08:55:07-08:00
[INFO] ------------------------------------------------------------------------

#########################################################
Tornado build success
Updating PATH and TORNADO_SDK to tornado-sdk-0.13-dev-256f715
Binaries: /opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin
Commit  : 256f715e8
#########################################################
[mydev@fedora TornadoVM-master]$
```

That's amazing!

test it

```
mydev@fedora TornadoVM-master]$ which tornado
opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/bin/tornado
mydev@fedora TornadoVM-master]$
mydev@fedora TornadoVM-master]$ tree -L 4 dist
ist
── tornado-sdk
   └── tornado-sdk-0.13-dev-256f715
       ├── bin
       │   ├── aws_post_processing.sh
       │   ├── cleanFpga.sh
       │   ├── convert2csv.sh
       │   ├── createJsonFile.py
       │   ├── javac.py
       │   ├── readJsonFile.py
       │   ├── runBenchmarks.sh
       │   ├── test-native.sh
       │   ├── tornado
       │   ├── tornado-benchmarks.py
       │   ├── tornadoDeviceInfo
       │   ├── tornadoLocalInstallMaven
       │   ├── tornado-test.py
       │   └── txt2csv.py
       ├── CHANGELOG.md
       ├── docs
       │   ├── 10_INSTALL_WITH_GRAALVM.md
       │   ├── 11_INSTALL_WITH_JDK8.md
       │   ├── 12_INSTALL_WITH_JDK11_PLUS.md
       │   ├── 13_INSTALL_WITH_DOCKER.md
       │   ├── 14_PUBLICATIONS.md
       │   ├── 15_FAQ.md
       │   ├── 16_RESOURCES.md
       │   ├── 17_AWS.md
       │   ├── 18_MALI.md
       │   ├── 19_CUDA.md
       │   ├── 1_INSTALL.md
       │   ├── 20_INSTALL_WINDOWS_WITH_GRAALVM.md
       │   ├── 21_KERNEL_CONTEXT.md
       │   ├── 22_SPIRV_BACKEND_INSTALL.md
       │   ├── 2_EXAMPLES.md
       │   ├── 3_INTELLIJ.md
       │   ├── 4_BENCHMARKS.md
       │   ├── 5_REDUCTIONS.md
       │   ├── 6_TORNADO_FLAGS.md
       │   ├── 7_FPGA.md
       │   ├── 8_BATCH.md
       │   ├── 9_PROFILER.md
       │   ├── CHANGELOG.md
       │   ├── Releases.md
       │   ├── TORNADOVM_TESTED_DRIVERS.md
       │   └── Unsupported.md
       ├── etc
       │   ├── exportLists
       │   ├── tornado.backend
       │   ├── tornado.properties
       │   └── tornado.release
       ├── examples
       │   ├── generated
       │   ├── TestTornado.java
       │   ├── virtual-device-CPU.json
       │   └── virtual-device-GPU.json
       ├── lib
       │   └── libtornado-opencl.so
       ├── LICENSE_APACHE2
       ├── LICENSE_GPLv2CE
       └── share
           └── java
── tornado-sdk.tar.gz
```

```
[mydev@fedora TornadoVM-master]$ tornado --devices
WARNING: Using incubator modules: jdk.incubator.foreign, jdk.incubator.vector
[TornadoVM-OCL-JNI] ERROR : clGetPlatformIDs -> Returned: -1001
[TornadoVM-OCL-JNI] ERROR : clGetPlatformIDs -> Returned: -30
Exception in thread "main" java.lang.ExceptionInInitializerError
        at tornado.drivers.common@0.13-dev/uk.ac.manchester.tornado.drivers.TornadoDeviceQuery.main(TornadoDeviceQuery.java:73
)
Caused by: uk.ac.manchester.tornado.api.exceptions.TornadoBailoutRuntimeException: [WARNING] No OpenCL platforms found. Deopti
mizing to sequential execution.
        at tornado.drivers.opencl@0.13-dev/uk.ac.manchester.tornado.drivers.opencl.OCLDriver.<init>(OCLDriver.java:59)
        at tornado.drivers.opencl@0.13-dev/uk.ac.manchester.tornado.drivers.opencl.OCLTornadoDriverProvider.createDriver(OCLTo
rnadoDriverProvider.java:50)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.loadDrivers(TornadoCoreRuntime.java:17
3)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.<init>(TornadoCoreRuntime.java:151)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.<clinit>(TornadoCoreRuntime.java:99)
        ... 1 more
[mydev@fedora TornadoVM-master]$
```

Aha!

run the test case “Vector Addition”in https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/2_EXAMPLES.md

Below you can find a snapshot of the `TestArrays` example code in TornadoVM (full code listing can be found in the `examples` directory).

In this example, we will run the `vectorAdd` method on a heterogeneous device. TornadoVM will dynamically compile and run the Java code (of the `vectorAdd` method) to an OpenCL/CUDA device. During the execution process, the code will be compiled from Java bytecode to OpenCL C/PTX and afterwards it will run on the OpenCL/CUDA compatible device, transparently.

As you can see in the example below, the accelerated `vectorAdd` method performs a double vector addition. Furthermore, it does not differ at all from a vanilla sequential Java implementation of the method. The only difference is the addition of the `@Parallel` annotation that instructs TornadoVM that the loop has to be computed in parallel (i.e. using the global identifier in OpenCL).

The `testVectorAddition` method prepares the input data and creates a TornadoVM `task`. TornadoVM `tasks` cannot execute directly; instead they must be part of a `TaskSchedule`. This is a design choice allowing a number of optimizations, such as task pipelining and parallelism, to be performed. Furthermore, `TaskSchedules` define which parameters are copied in and out from a device.

Once the method `execute` is invoked, TornadoVM builds the data dependency graph, compiles the referenced Java method to OpenCL C/PTX, and executes the generated application on the available OpenCL/CUDA device.

mkdir -p tests;vim tests/VectorAddFloat.java

```java
import java.util.Arrays;

import uk.ac.manchester.tornado.api.TaskSchedule;
import uk.ac.manchester.tornado.api.annotations.Parallel;

public class VectorAddFloat {

 private static void vectorAdd(float[] a, float[] b, float[] c) {
        for (@Parallel int i = 0; i < c.length; i++) {
            c[i] = a[i] + b[i];
        }
    }

    public static void main(String[] args) {
        int size = Integer.parseInt(args[0]);

        float[] a = new float[size];
        float[] b = new float[size];
        float[] c = new float[size];
        float[] result = new float[size];

        Arrays.fill(a, 450f);
        Arrays.fill(b, 20f);

        //@formatter:off
        TaskSchedule task = new TaskSchedule("s0")
                .task("t0", VectorAddFloat::vectorAdd, a, b, c)
        .streamOut(c);
        //@formatter:on

        task.execute();
        vectorAdd(a, b, result);
        boolean wrongResult = false;
        for (int i = 0; i < c.length; i++) {
            if (c[i] != result[i]) {
                wrongResult = true;
                break;
            }
        }
        if (!wrongResult) {
            System.out.println("Test success");
        } else {
            System.out.println("Result is wrong");
        }
    }
}
```

build VectorAddFloat.java

```
[mydev@fedora TornadoVM-master]$ cd tests;javac.py VectorAddFloat.java
/opt/MyWorkSpace/DevSW/Java/JDK/GraalVM/CE/java17-21.3.0/bin/javac  -classpath ".:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Bee
hive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/asm-9.2.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoV
M-master/bin/sdk/share/java/tornado/commons-lang3-3.3.2.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-mas
ter/bin/sdk/share/java/tornado/commons-math3-3.2.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin
/sdk/share/java/tornado/ejml-core-0.38.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share
/java/tornado/ejml-ddense-0.38.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/to
rnado/ejml-dsparse-0.38.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/e
jml-simple-0.38.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/hamcrest-
core-1.3.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/jmh-core-1.29.ja
r:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/jopt-simple-4.6.jar:/opt/My
WorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/jsr305-3.0.2.jar:/opt/MyWorkSpace/My
Projs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/junit-4.13.1.jar:/opt/MyWorkSpace/MyProjs/Runtim
e/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/log4j-api-2.16.0.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalV
M/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/log4j-core-2.16.0.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehi
ve-Lab/TornadoVM-master/bin/sdk/share/java/tornado/lucene-core-8.2.0.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/
TornadoVM-master/bin/sdk/share/java/tornado/tornado-annotation-0.13-dev.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-L
ab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-api-0.13-dev.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-Lab/T
ornadoVM-master/bin/sdk/share/java/tornado/tornado-benchmarks-0.13-dev.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Beehive-La
b/TornadoVM-master/bin/sdk/share/java/tornado/tornado-drivers-common-0.13-dev.jar:/opt/MyWorkSpace/MyProjs/Runtime/GraalVM/Bee
hive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-drivers-opencl-0.13-dev.jar:/opt/MyWorkSpace/MyProjs/Runtime/Graa
lVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-drivers-opencl-jni-0.13-dev-libs.jar:/opt/MyWorkSpace/MyPr
ojs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-examples-0.13-dev.jar:/opt/MyWorkSpace/MyP
rojs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-matrices-0.13-dev.jar:/opt/MyWorkSpace/My
Projs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-runtime-0.13-dev.jar:/opt/MyWorkSpace/My
Projs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/tornado-unittests-0.13-dev.jar:/opt/MyWorkSpace/
MyProjs/Runtime/GraalVM/Beehive-Lab/TornadoVM-master/bin/sdk/share/java/tornado/" VectorAddFloat.java
[mydev@fedora tests]$
```

run VectorAddFloat will result in expected failure

```
[mydev@fedora tests]$ tornado --printKernel --debug VectorAddFloat 5
WARNING: Using incubator modules: jdk.incubator.vector, jdk.incubator.foreign
[TornadoVM-OCL-JNI] ERROR : clGetPlatformIDs -> Returned: -1001
[TornadoVM-OCL-JNI] ERROR : clGetPlatformIDs -> Returned: -30
Exception in thread "main" java.lang.ExceptionInInitializerError
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.analyzer.TaskUtils.resolveMethodHandle(TaskUtils.java:165
)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.tasks.TornadoTaskSchedule.addTask(TornadoTaskSchedule.jav
a:1909)
        at tornado.api@0.13-dev/uk.ac.manchester.tornado.api.TaskSchedule.task(TaskSchedule.java:130)
        at VectorAddFloat.main(VectorAddFloat.java:27)
Caused by: uk.ac.manchester.tornado.api.exceptions.TornadoBailoutRuntimeException: [WARNING] No OpenCL platforms found. Deopti
mizing to sequential execution.
        at tornado.drivers.opencl@0.13-dev/uk.ac.manchester.tornado.drivers.opencl.OCLDriver.<init>(OCLDriver.java:59)
        at tornado.drivers.opencl@0.13-dev/uk.ac.manchester.tornado.drivers.opencl.OCLTornadoDriverProvider.createDriver(OCLTo
rnadoDriverProvider.java:50)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.loadDrivers(TornadoCoreRuntime.java:17
3)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.<init>(TornadoCoreRuntime.java:151)
        at tornado.runtime@0.13-dev/uk.ac.manchester.tornado.runtime.TornadoCoreRuntime.<clinit>(TornadoCoreRuntime.java:99)
        ... 4 more
[mydev@fedora tests]$
```

- Can't wait to get an OpenCL implementation ready for Raspberry Pi 4!
- Considering install TornadoVM to GraalVM via gu
  https://www.graalvm.org/reference-manual/graalvm-updater/

…

SPIR-V

- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/22_SPIRV_BACKEND_INSTALL.md
- Currently only Intel oneAPI Level Zero Compute Runtime is supported
- …

FPGA

- **https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/7_FPGA.md**

We have currently tested with an Intel Nallatech-A385 FPGA (Intel Arria 10 GT1150) and a Xilinx KCU1500 FPGA card. We have also tested it on the AWS EC2 F1 instance with `xilinx_aws-vu9p-f1-04261818_dynamic_5_0 device`.

- HLS Versions: Intel Quartus 17.1.0 Build 240, Xilinx SDAccel 2018.2, Xilinx SDAccel 2018.3, Xilinx Vitis 2020.2
- TornadoVM Version: >= 0.9
- AWS AMI Version: 1.6.0

- **Currently support three execution modes**
  1. **Full JIT Mode**
  2. **Ahead of Time Execution Mode**
  3. **Emulation Mode**
- **…**

ARM Mali

- **https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/18_MALI.md**

Installation

The installation of TornadoVM to run on ARM Mali GPUs requires JDK11 with GraalVM. See the INSTALL WITH GRAALVM document for details about the installation.

The OpenCL driver for Mali GPUs on Linux that has been tested is:

- OpenCL C 2.0 v1.r9p0-01rel0.37c12a13c46b4c2d9d736e0d5ace2e5e: link

Testing

We have tested TornadoVM on the following ARM Mali GPUs:

- Mali-G71, which implements the Bifrost architecture: link

Some of the unittests in TornadoVM run with `double` data types. To enable double support, TornadoVM includes the following extension in the generated OpenCL code:

```
cl_khr_fp64
```

However, this extension is not available on Bifrost GPUs.

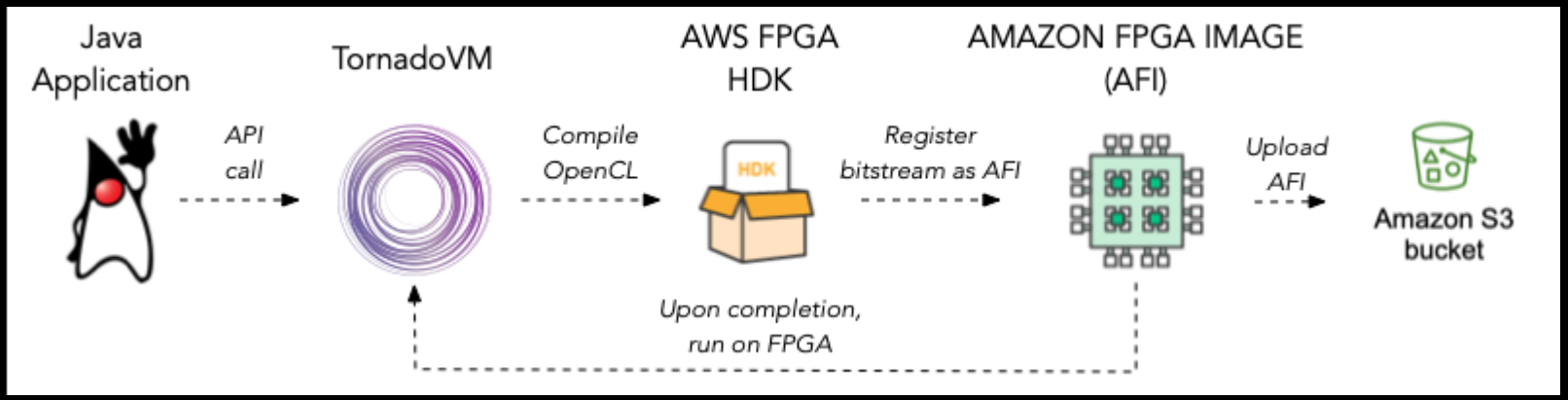
The rest of the unittests should pass.

- **https://developer.arm.com/tools-and-software/graphics-and-gaming/mali-drivers/bifrost-kernel**

…

Cloud

- **https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/17_AWS.md**
- **AWS EC2 F1 Xilinx FPGAs**



- **…**

Cloud-Native

- hhttps://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/13_INSTALL_WITH_DOCKER.md
- https://github.com/beehive-lab/docker-tornado

dockerFiles	Links to Graal 21.2.0 updated for Intel HD Graphics	3 months ago
example	LABEL for maintainer	3 years ago
settings	built JDK for iGPU docker images updated	2 years ago
.gitignore	gitignore added	3 years ago
LICENSE	LICENSE file added	3 years ago
README.md	Update README.md	11 months ago
build.sh	Tag version 0.12	last month
clean.sh	Initial Tornado Docker on NVIDIA GPU	3 years ago
demo.sh	Images updated for GraalVM	2 years ago
run_intel.sh	intel GPU tag image and dockerhub name fixed	3 years ago
run_intel_graalvm_jdk11.sh	Fix image docker URLs	14 months ago
run_intel_graalvm_jdk8.sh	Fix image docker URLs	14 months ago
run_nvidia.sh	Script names updated	3 years ago
run_nvidia_graalvm-jdk11.sh	Images updated for GraalVM	2 years ago
run_nvidia_graalvm-jdk8.sh	Images updated for GraalVM	2 years ago

- …

Miscs

- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/TORNADOVM_TESTED_DRIVERS.md
- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/12_INSTALL_WITH_JDK11_PLUS.md
- https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/2_EXAMPLES.md
- …

III. GraalVM as a generic platform for HPC

1) Smart Runtime

1.1 What is it (From our point of view)

Traditional Runtime + AI

- XPU-aware compilation
- Auto-scheduling
- Auto-tuning
- Auto-debugging
- …

GraalVM + TornadoVM provides a good base for "Smart Runtime".

2) Support for OpenCL 3.x and SPIR-V backend

2.1 OpenCL 3.x implementation

OneAPI

- https://www.oneapi.io/compute-runtime-20-41-18123-flips-on-opencl-3-0-for-all-hardware-back-to-broadwell/ (Oct 10, 2020)

  The fastest WebAssembly interpreter, and the most universal runtime.
- https://github.com/intel/compute-runtime/
- …

2.2 OpenCL on RPi

2.2.1 OpenCL on RPi3

- https://github.com/doe300/VC4CL

VC4CL is an implementation of the **OpenCL 1.2** standard for the VideoCore IV GPU (found in Raspberry Pi 1 - 3 models).

The implementation consists of:

- The **VC4CL** OpenCL runtime library, running on the host CPU to compile, run and interact with OpenCL kernels.
- The VC4C compiler, converting OpenCL kernels into machine code. This compiler also provides an implementation of the OpenCL built-in functions.
- The VC4CLStdLib, the platform-specific implementation of the OpenCL C standard library, is linked in with the kernel by **VC4C**

- https://github.com/doe300/VC4C

Compiler for the VC4CL OpenCL-implementation. This compiler supports OpenCL C (via LLVM or SPIRV-LLVM), LLVM-IR and SPIR-V code, depending on the build configuration.

- https://abhitronix.github.io/2019/01/15/VC4CL-1/
- …

2.2.2 OpenCL on RPi4

- **Unfortunately, there is no available OpenCL implementation on RPi4 by now…**

Vulkan 1.1 support is ready!

- **https://www.khronos.org/conformance/adopters/conformant-products/vulkan#submission_596**

Raspberry PI Ltd 2021-10-20 Vulkan_1_1	
Raspberry Pi 4 Model B	CTS Version: 1.2.7.1 Driver Version: Mesa 21.3.0 (v3dv) CTS Version: 1.2.7.1 Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) OS: Raspbian Linux 5.10.63-v8 (aarch64) API version: 1.1.190 GPU: VideoCore VI HW (V3D 4.2)

- **https://en.wikipedia.org/wiki/Vulkan**

Vulkan 1.1 [edit]

At SIGGRAPH 2016, Khronos announced that Vulkan would be getting support for automatic multi-GPU features, similar to what is offered by Direct3D 12.[38] Multi-GPU support included in-API removes the need for SLI or Crossfire which requires graphics cards to be of the same model. API multi-GPU instead allows the API to intelligently split the workload among two or more completely different GPUs.[39] For example, integrated GPUs included on the CPU can be used in conjunction with a high-end dedicated GPU for a slight performance boost.

On March 7, 2018, Vulkan 1.1 was released by the Khronos Group.[40] This first major update to the API standardized several extensions, such as multi-view, device groups, cross-process and cross-API sharing, advanced compute functionality, HLSL support, and YCbCr support.[41] At the same time, it also brought better compatibility with DirectX 12, explicit multi-GPU support, ray tracing support,[42][43] and laid the groundwork for the next generation of GPUs.[44] Alongside Vulkan 1.1, SPIR-V was updated to version 1.3.[41]

Vulkan 1.2 [edit]

On January 15, 2020, Vulkan 1.2[45] was released by the Khronos Group.[46] This second major update to the API integrates 23 additional commonly-used proven Vulkan extensions into the base Vulkan standard. Some of the most important features are "timeline semaphores for easily managed synchronization", "a formal memory model to precisely define the semantics of synchronization and memory operations in different threads", and "descriptor indexing to enable reuse of descriptor layouts by multiple shaders". The additional features of Vulkan 1.2 improve its flexibility when it comes to implementing other graphics APIs on top of Vulkan, including "uniform buffer standard layout", "scalar block layout", and "separate stencil usage".[47]

Planned features [edit]

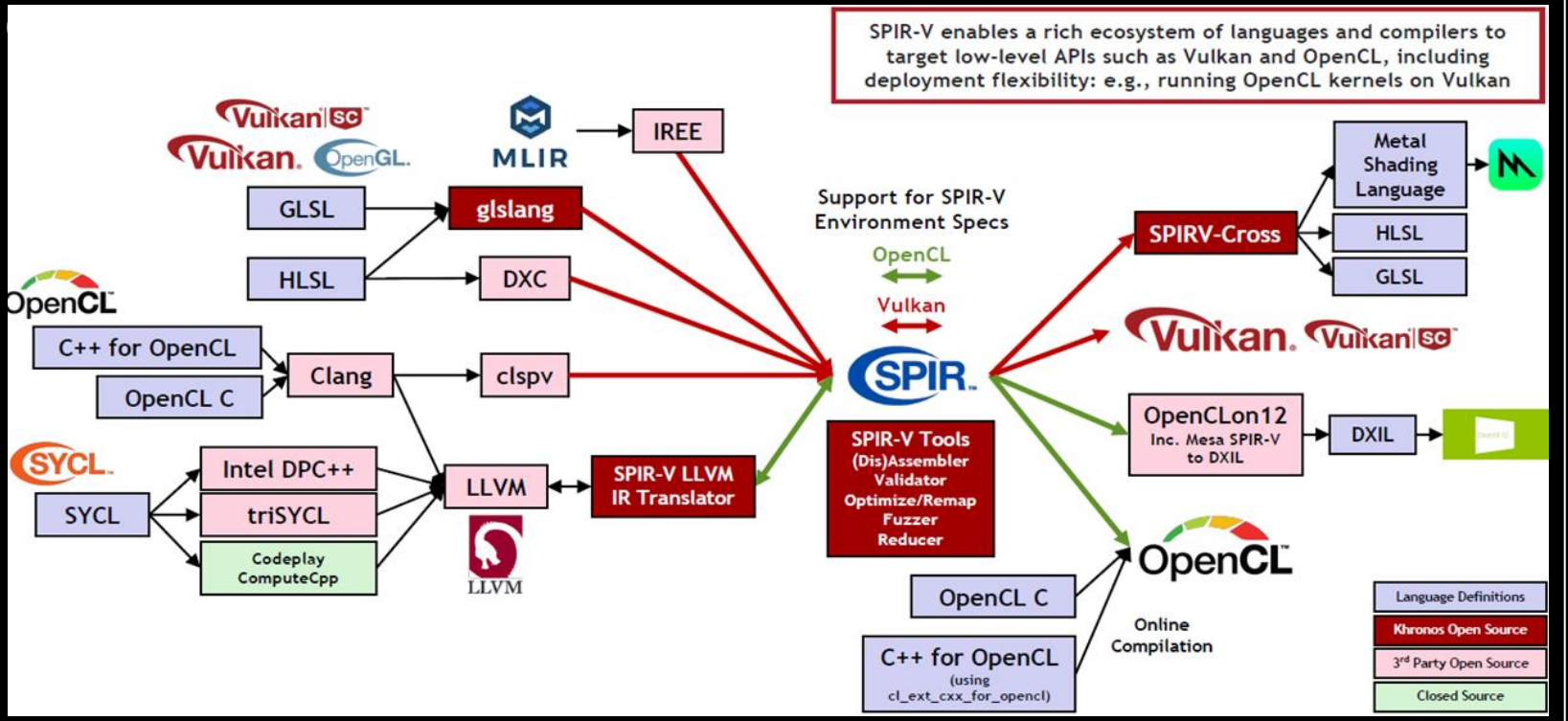
When releasing OpenCL 2.2, the Khronos Group announced that OpenCL would converge where possible with Vulkan to enable OpenCL software deployment flexibility over both APIs.[48][49] This has been now demonstrated by Adobe's Premiere Rush using the clspv[50] open source compiler to compile significant amounts of OpenCL C kernel code to run on a Vulkan runtime for deployment on Android.[51]

- **https://gitlab.freedesktop.org/mesa/mesa/blob/main/docs/features.txt**

2.3 SPIR-V Backend

- https://en.wikipedia.org/wiki/Standard_Portable_Intermediate_Representation
- https://www.khronos.org/spir/

2.3.1 SPIR-V Language Ecosystem



Source: "State of the Union OpenCL Working Group", Neil Trevett, IWOCL 2021.

- ...

SPIR-V backend for LLVM is ready!

- https://lists.llvm.org/pipermail/llvm-dev/2021-December/154270.html
- https://www.phoronix.com/scan.php?page=news_item&px=LLVM-Ready-For-SPIR-V-Backend
- https://www.phoronix.com/scan.php?page=news_item&px=Intel-2021-LLVM-SPIR-V-Backend
- https://www.phoronix.com/scan.php?page=news_item&px=LLVM-HIPSPV-SPIR-V
- https://github.com/KhronosGroup/LLVM-SPIRV-Backend/llvm/lib/Target/SPIRV
- …

LLVM/SPIR-V Bi-Directional Translator

- https://github.com/KhronosGroup/SPIRV-LLVM-Translator
  A tool and a library for bi-directional translation between SPIR-V and LLVM IR.

SPIRV-Cross

- https://github.com/KhronosGroup/SPIRV-Cross
  A practical tool and library for performing reflection on SPIR-V and disassembling SPIR-V back to high level languages.

2.3.2 SPIR-V support in Tornando

SPIR-V Driver

```
[mydev@fedora TornadoVM-master]$ tree -L 2 -d drivers/
drivers/
├── drivers-common
│   └── src
├── opencl
│   └── src
├── opencl-jni
│   └── src
├── ptx
│   └── src
├── ptx-jni
│   └── src
├── spirv
│   └── src
└── spirv-levelzero-jni
    └── src
```

SPIR-V Beehive Toolkit

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

  Prototype for a SPIR-V assembler and dissasembler. It provides a composable Java interface for generating SPIR-V code at runtime..

- https://github.com/beehive-lab/spirv-beehivetoolkit/blob/master/docs/EXAMPLE.md
- ...

3) Common Runtime for FOSS EDA

3.1 GaaS(GraalVM as a Service)

- **Top 3 Polyglot Runtimes**
  Considering GraalVM, .Net, Wasm

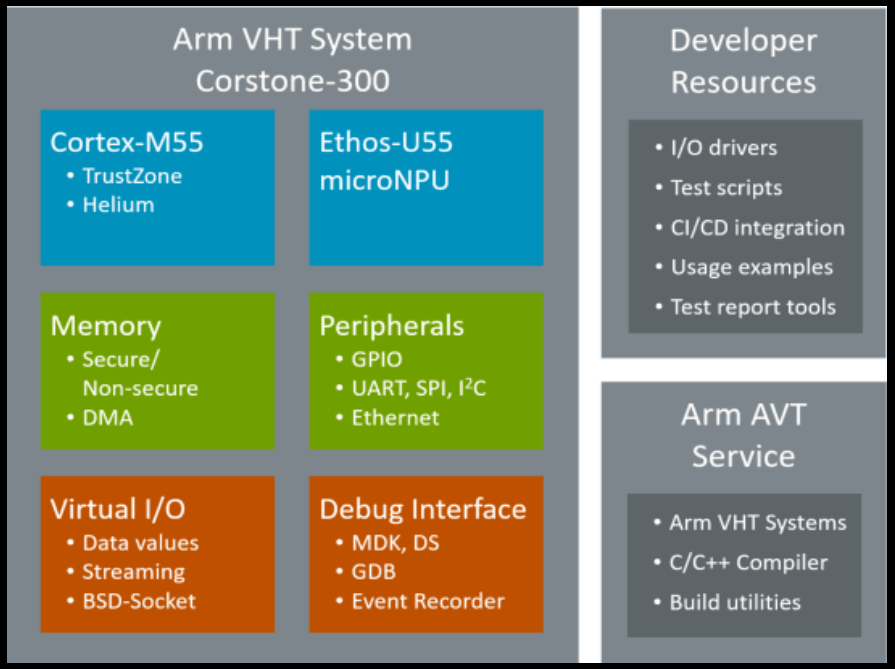
- **uBPF and Wasm on GraalVM**
  In addition, Lua, C#, Go, and some DSLs on GraalVM…
- **For details, please refer to my previous talk "Revisiting GraalVM-based unified runtime for eBPF & WebAssembly" at OpenInfra Days China 2021(Beijing) and upcoming follow-ups.**
- **For details, you may refer to my previous talks as below:**
  "Scala-based FOSS EDA on ARM" at the 5th China Functional Programming Meetup 2021(Shanghai).
  "Python-based Open Source Toolchain for RISC-V Development" at the 1st RISC-V World Conference China 2021 (Shanghai).
  And the upcoming follow-ups:
  "Rust for FOSS EDA", "Revisiting Python for FOSS EDA" , "Revisiting Scala-based FOSS EDA"…

3.2 Serverless Architecture

- GraalVM is Serverless friendly

EDA in the Cloud

- https://www.arm.com/company/news/2020/12/arm-moves-production-level-electronic-design-automation-to-the-cloud-with-the-help-of-aws
- https://www.arm.com/company/news/2021/10/arm-transforms-the-economics-of-iot-with-virtual-hardware-and-new-solutions-led-offering



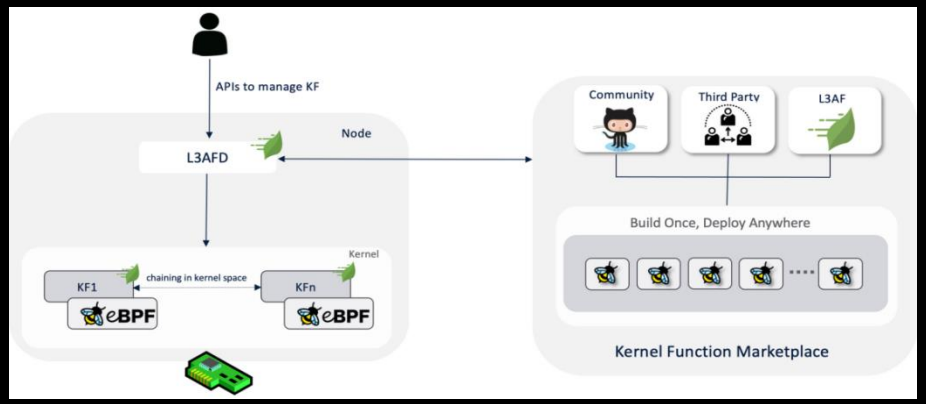
- …

KFaaS(Kernel Functions as a Service)

- https://www.linuxfoundation.org/press-release/walmart-moves-production-grade-networking-project-l3af-to-the-linux-foundation/
- https://l3af.io/
- https://github.com/l3af-project
- Features

Innovation	Empowerment	Flexibility
Industry-first "Kernel Functions as a Service"	Simple API to add, remove, and reorder kernel functions on the fly	Distributed model to manage and configure kernel functions on a per-node basis
Multiple independent kernel functions executing in a chain	Configurable metrics are gathered for each kernel function	Compose kernel functions to fit business needs
More to come, including a community-driven kernel function marketplace	Replace proprietary applications and hardware with blazing fast eBPF code	Cloud and vendor agnostic

- Platform



- …

3.3 HW/SW Co-development

3.3.1 Renode

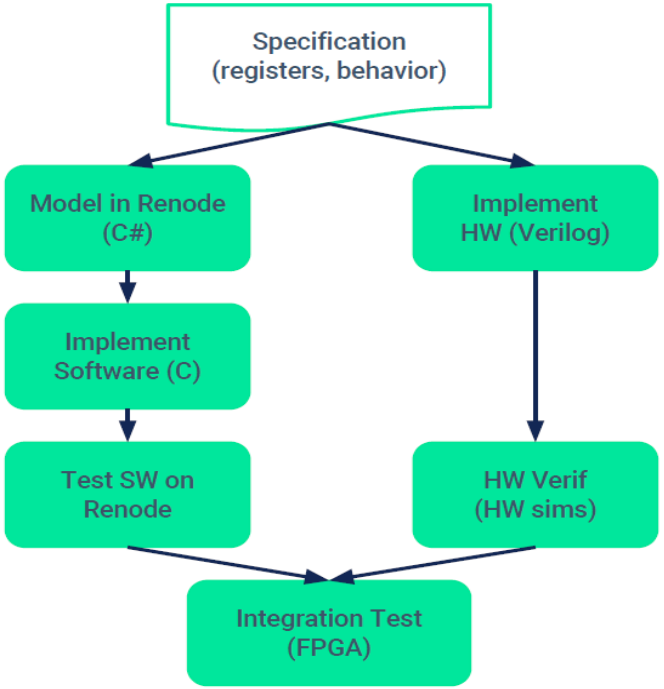
- https://renode.io/
- https://github.com/renode

Renode-assisted Parallel HW/SW Development

- HW spec developed between HW and SW teams
- SW team implements spec in Renode and writes firmware against spec, testing on Renode
- In parallel, HW is implementing and testing HW design
- Integration test via FPGA and/or HW simulator

EXAMPLE

HDMI device. We had SW working against spec, under Renode, in advance of HW.

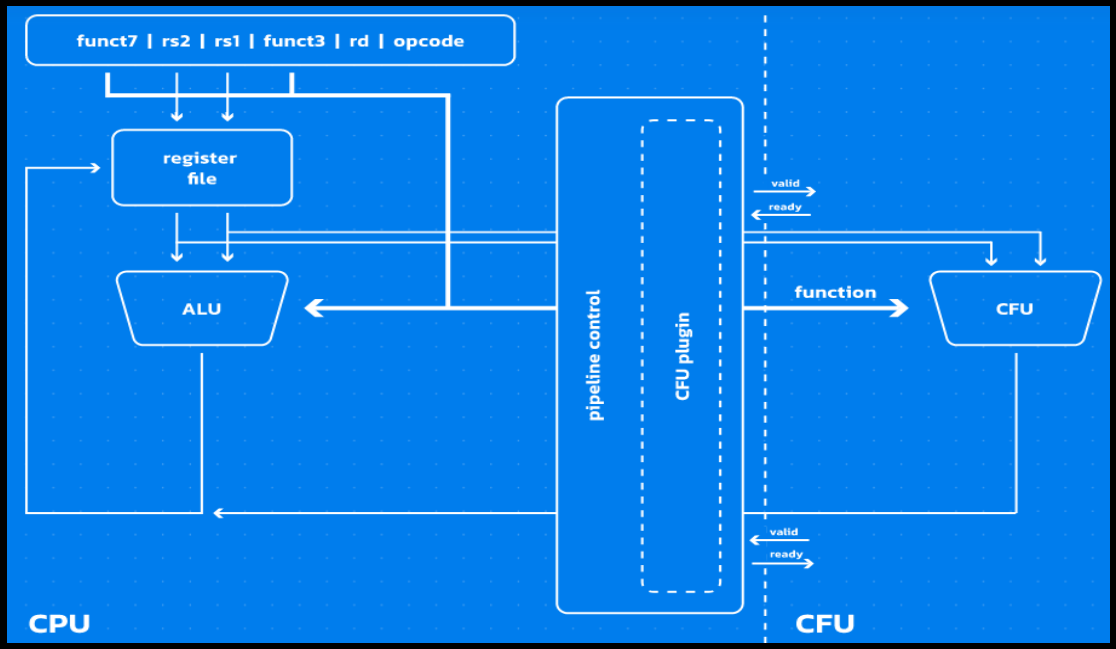


Source: "Hardware/Software Co-Design with the Open Source Renode Framework and RISC-V", Michael Gielda, Getting Started With RISC-V NA Tour 2019.

3.3.1.1 CFU

- Custom Function Units
- https://antmicro.com/blog/2021/09/cfu-support-in-renode/

RISC-V is also excellent for FPGA-based ML development, offering a multitude of FPGA-friendly softcore options such as VexRiscv and specialized ML-oriented extensions called CFU - which you can experiment with both in cheap, easily accessible hardware as well as, you guessed it, with Renode, using Verilator co-simulation capabilities that we have described a few times already.



CFU Playground

- **https://cfu-playground.readthedocs.io/en/latest/**

  Accelerate ML models on FPGAs.

"CFU" stands for Custom Function Unit: accelerator hardware that is tightly coupled into the pipeline of a CPU core, to add new custom function instructions that complement the CPU's standard functions (such as arithmetic/logic operations).

The CFU Playground is a collection of software, gateware and hardware configured to make it easy to:

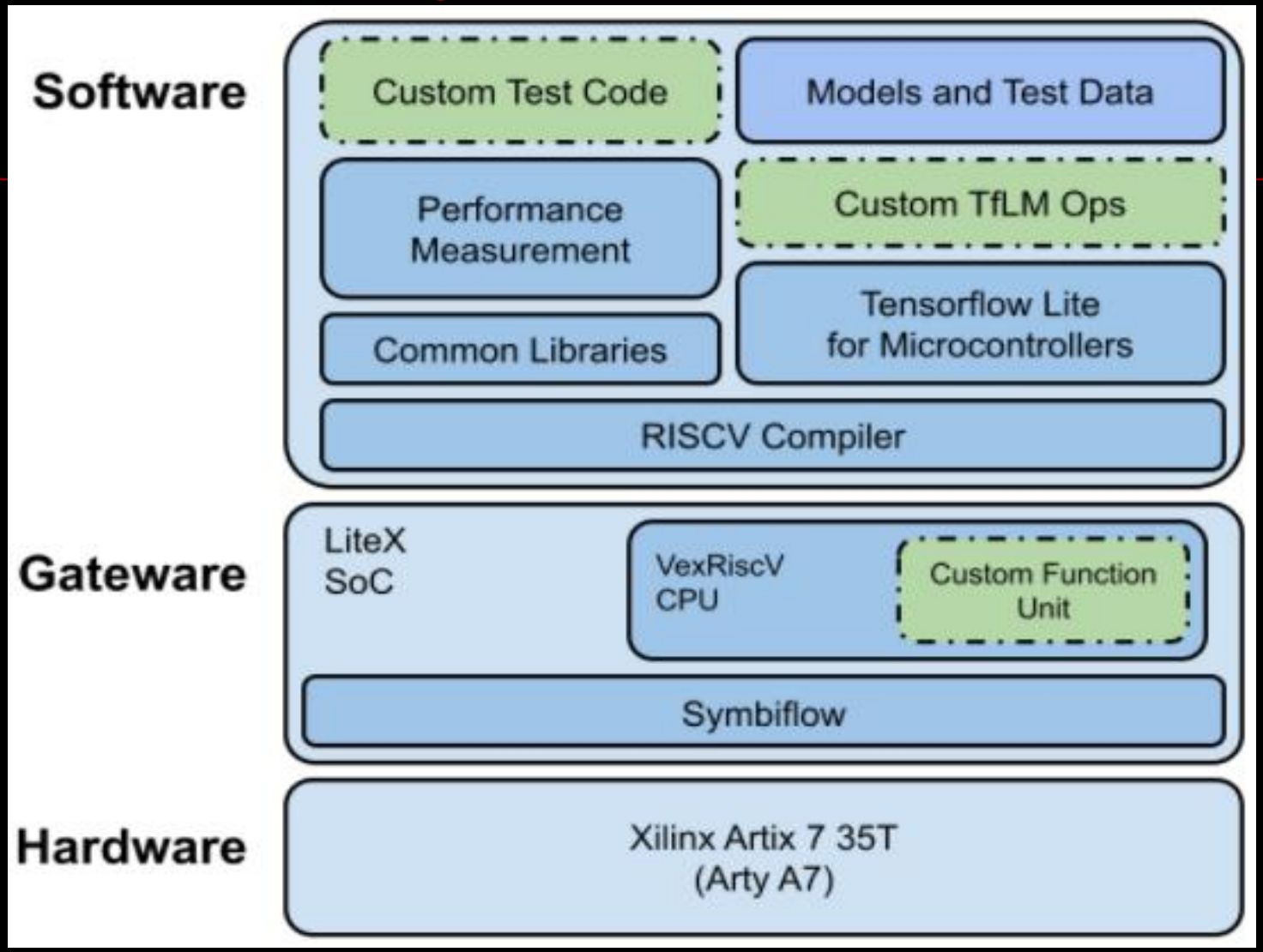
- Run ML models
- Benchmark and profile performance
- Make incremental improvements
  - In software by modifying source code
  - In gateware with a CFU
- Measure the results of changes

ML acceleration on microcontroller-class hardware is a new area, and one that, due to the expense of building ASICs, is currently dominated by hardware engineers. In order to encourage software engineers to join in the innovation, the CFU-Playground aims to make experimentation as simple, fast and fun as possible.

As well as being a useful tool for accelerating ML inferencing, the CFU Playground is a relatively gentle introduction to using FPGAs for computation.

- **A new kind of ASIP?**
- **…**

Architecture & Design



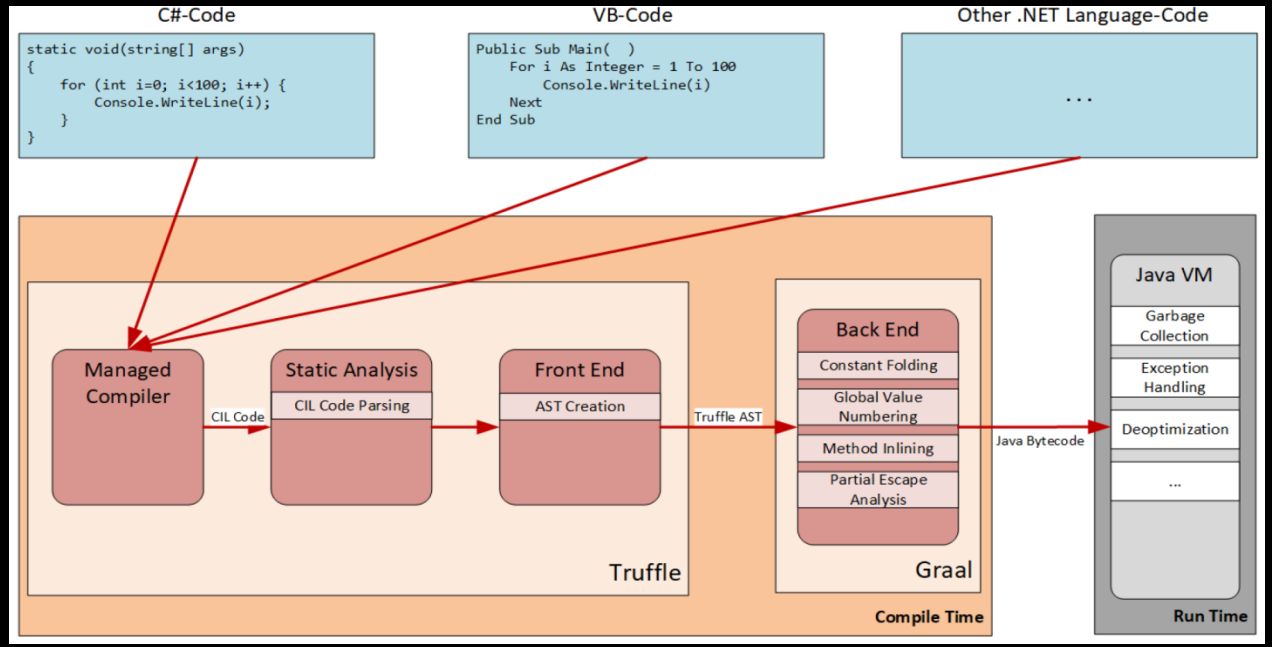
3.3.1.2 Renode on GraalVM

Scheme 1

- Running Renode on GraalVM
- Truffle CIL Interpreter

https://githubmemory.com/repo/jagotu/BACIL

https://epub.jku.at/obvulihs/download/pdf/5473678?originalFilename=true(not open-sourced, and poor performance…)



Scheme 2

- Rewriting Renode with a GraalVM-native language…

3.4 Unified runtime for eBPF and Wasm in Userland

- For details, you may refer to my previous talks and upcoming follow-ups:
  1. "GraalVM-based unified runtime for eBPF & Wasm" at GOTC 2021(Shenzhen)
  2. "Revisiting GraalVM-based unified runtime for eBPF & Wasm" at OpenInfra Days China 2021(Beijing).

3.5 Ideas and design

- For more details, you may refer to my previous talks:
  1. "Rethinking Hyper-Converged Infrastructure for Edge Computing" at OpenInfra Days China 2019(Shanghai).
  2. "Revisiting the eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing" at K+ Summit 2021(Shanghai).

  and upcoming follow-ups…

IV. Wrap-up

- Heterogeneous Parallel Computing is the future!
- GraalVM + TornadoVM is awesome!
- How about GasS(GraalVM as a Service)?
- There is still a lot of work to be done to bring TornadoVM to ARM and other popular hardware platforms or products…
- "One VM to Rule Them All" is not a dream!
- Please look forward to our follow-ups "Revisiting GraalVM for Heterogeneous Parallel Computing", "GraalVM plus TornadoVM as a common runtime for FOSS EDA", "The 3rd round discussion on GraalVM-based unified runtime for eBPF & Wasm" and "The 3rd round discussion on eBPF-centric new approach for Hyper-Converged Infrastructure & Edge Computing"(will be divided into "eBPF 101", "Rust 101", "GraalVM 101" , HW/SW Tech Stack 101", and itself will only focus on design and implementation of our schemes)…

Q & A

Thanks!

Reference

Slides/materials from many and varied sources:

- http://en.wikipedia.org/wiki/
- http://www.slideshare.net/
- https://www.forbes.com/sites/oracle/2019/05/08/meet-the-team-that-built-graalvm-an-energy-saving-multilingual-compiler-written-entirely-in-java
- https://www.slideshare.net/VadymKazulkin/adopting-java-for-the-serverless-world-at-serverless-meetup-singapore
- https://github.com/KhronosGroup/SPIRV-Guide/blob/master/chapters/what_is_spirv.md
- https://www.khronos.org/blog/offline-compilation-of-opencl-kernels-into-spir-v-using-open-source-tooling
- …