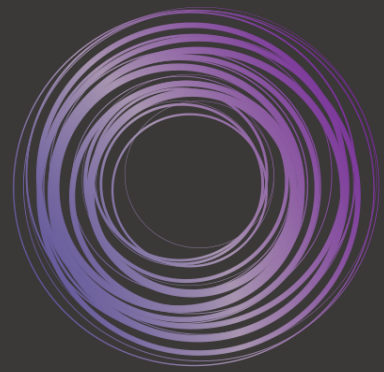


Transparent Heterogeneous Computing for Java via TornadoVM

Juan Fumero

Research Associate @ The University of Manchester
http://jjfumero.github.io

Twitter: @snatverk

NYJavaSIG, 10th February 2021

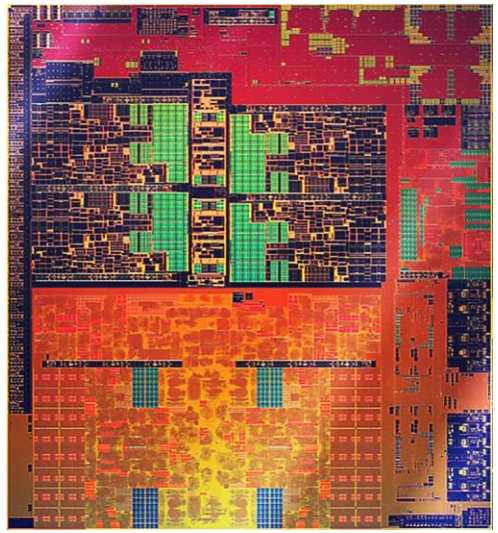
Agenda

1. Background and Terminology
2. TornadoVM
   - API - examples
   - Runtime & Just In Time Compiler
   - Live Task Migration
   - Demos on GPU and FPGAs
   - Some of the TornadoVM-Graal JIT Internals
3. Performance Results
4. How TornadoVM is being used in Industry?
5. Related Work & Future Directions
6. Conclusions

Motivation

Why should we care about GPUs/FPGAs, etc.?

CPU



Intel Ice Lake (10nm)
8 cores HT, AVX(512 SIMD)
~1TFlops* (including the iGPU)
~ TDP 28W
Source: Intel docs

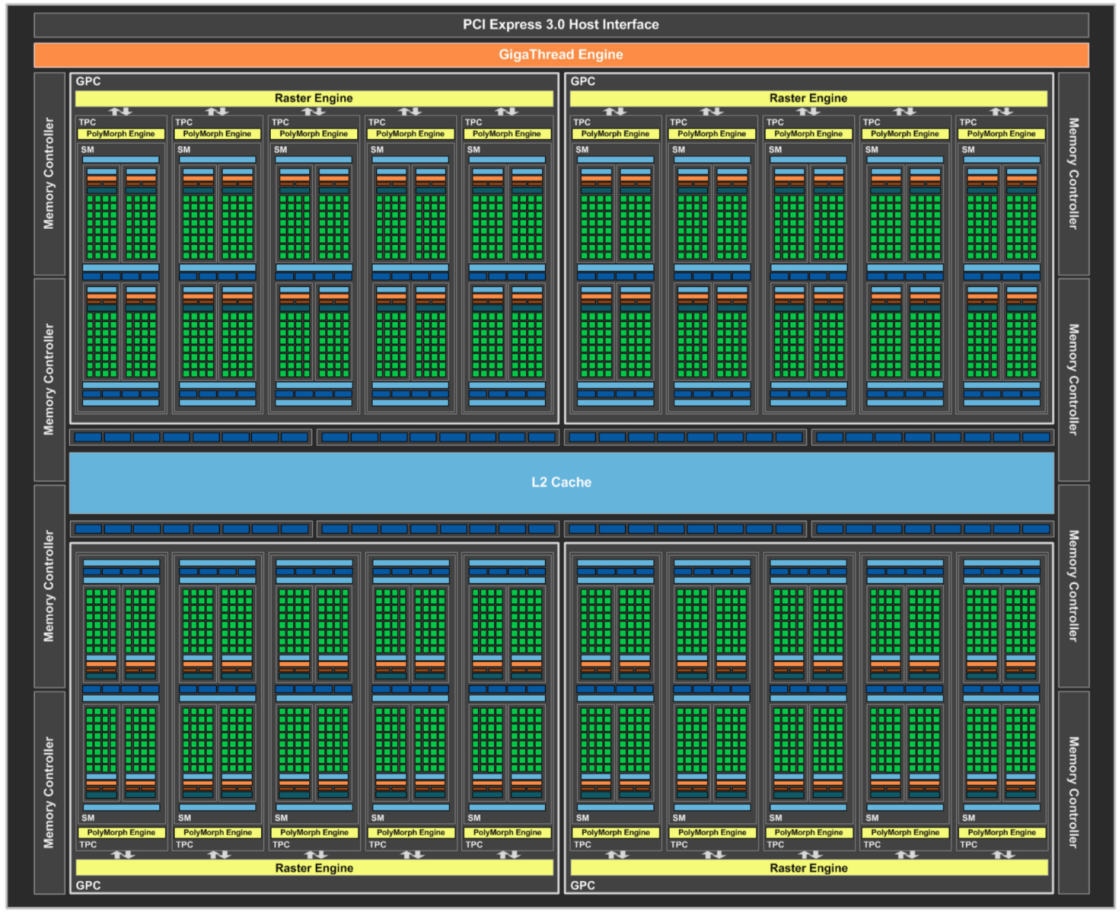
GPU

NVIDIA GP 100 – Pascal - 16nm
60 SMs, 64 cores each
3584 FP32 cores
10.6 TFlops (FP32)
TDP ~300 Watts
Source: NVIDIA docs

FPGA

Intel FPGA Stratix 10 (14nm)
Reconfigurable Hardware
~ 10 TFlops
TDP ~225Watts
Source: Intel docs

What is a GPU? Graphics Processing Unit



Source: NVIDIA docs

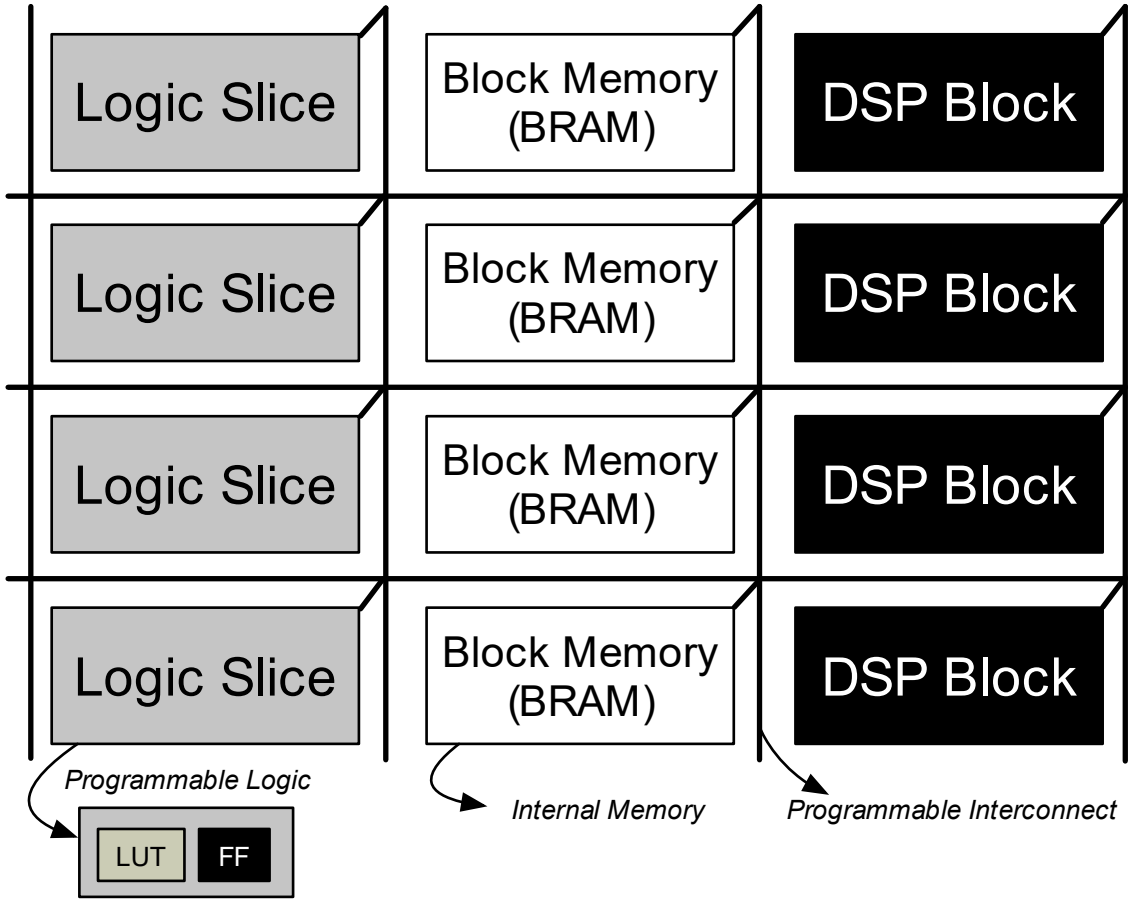
Contains a set of Stream Multiprocessor cores (SMx)
* Pascal arch. 60 SMx
* ~3500 CUDA cores

Users need to know:

A) Programming model (normally CUDA or OpenCL)

B) Details about the architecture are essential to achieve performance
* Non sequential consistency, manual barriers, etc.

What is an FPGA? Field Programmable Gate Array

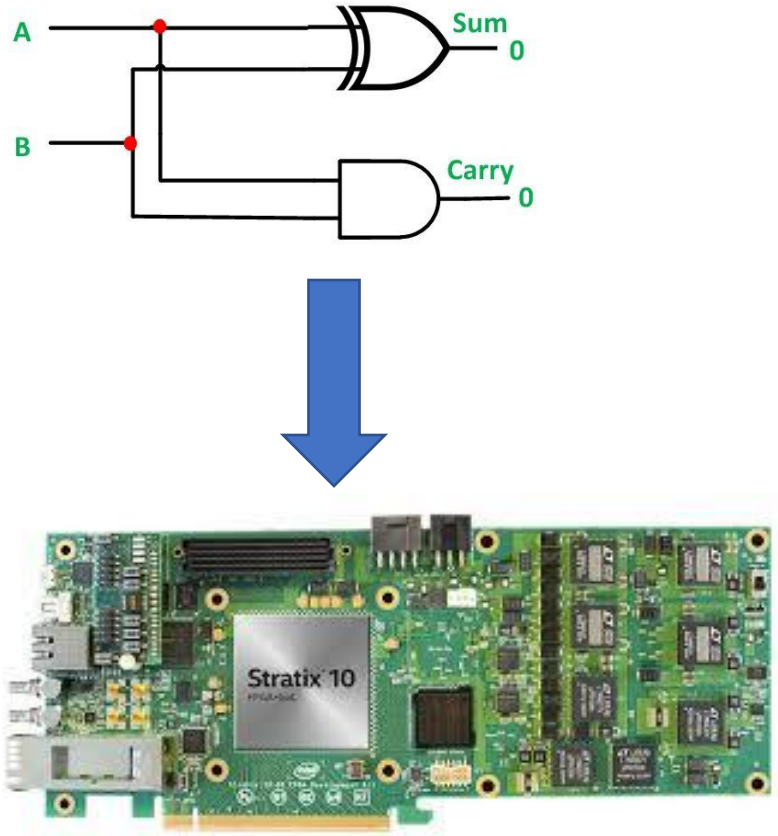


You can configure the design of your hardware after manufacturing

It is like having "*your algorithms directly wired on hardware*" with only the parts you need

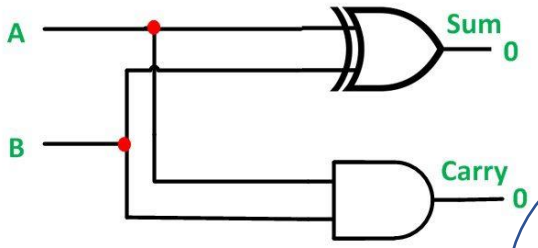
Example in VHDL (using structural modelling)

```
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;
```



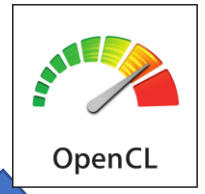
Using OpenCL instead

```
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;
```

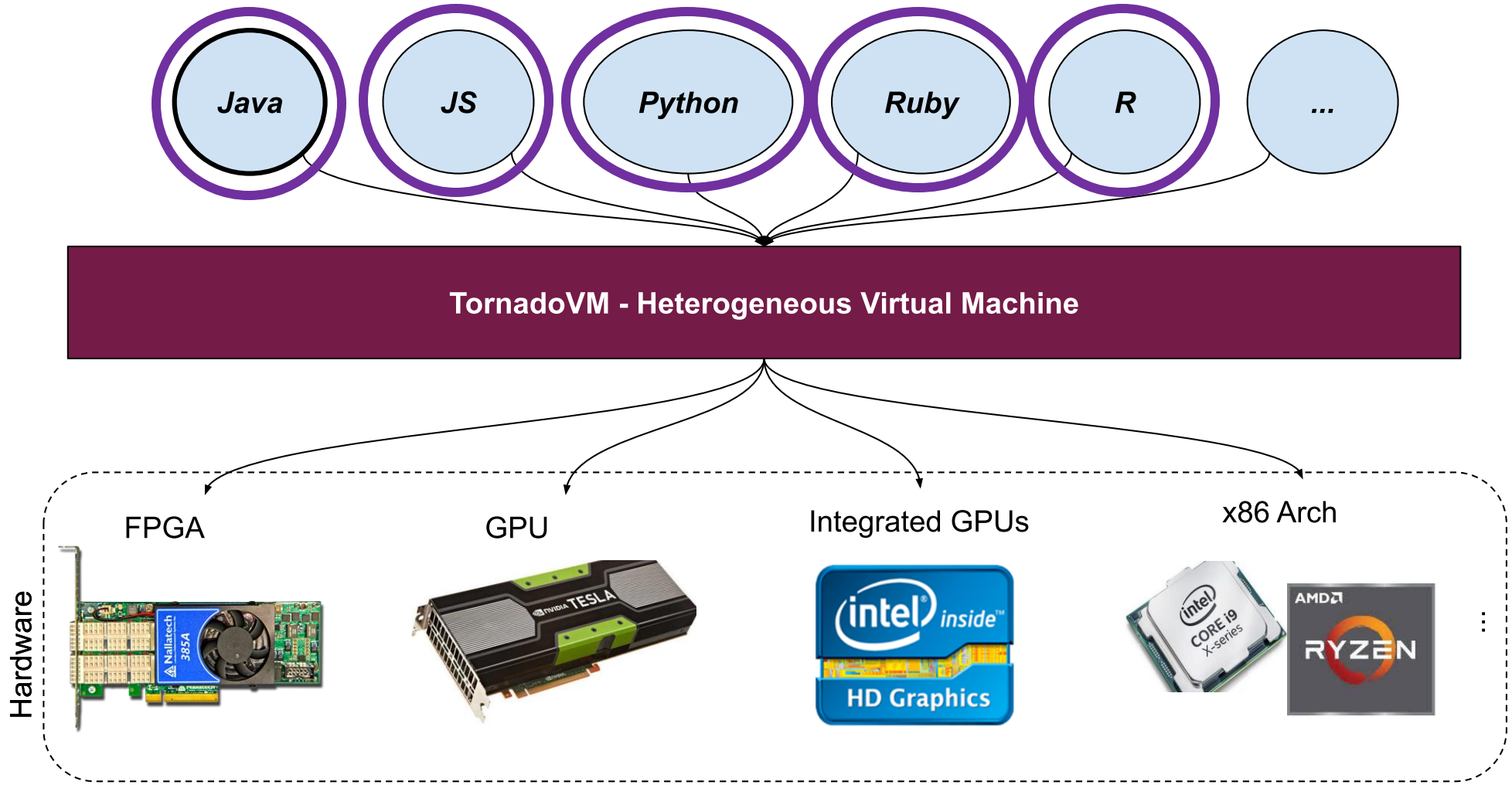


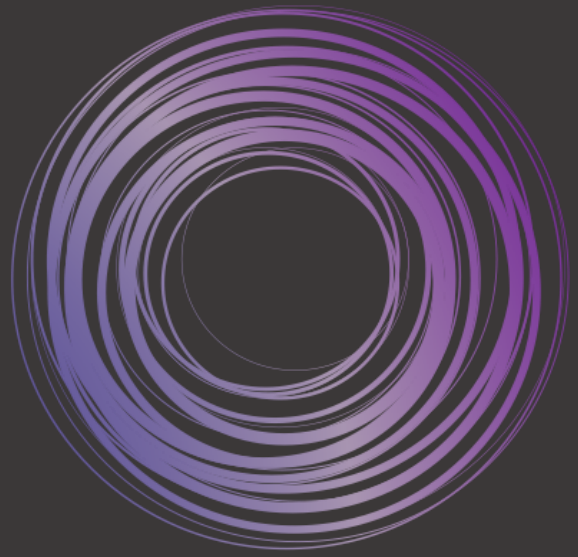

```
kernel void sum
(global float a,
 global float b,
 global float *result)
{
  result[0] = a + b;
}
```

Industry is pushing for OpenCL on FPGAs!



Our Vision: Ideal System for Managed Languages

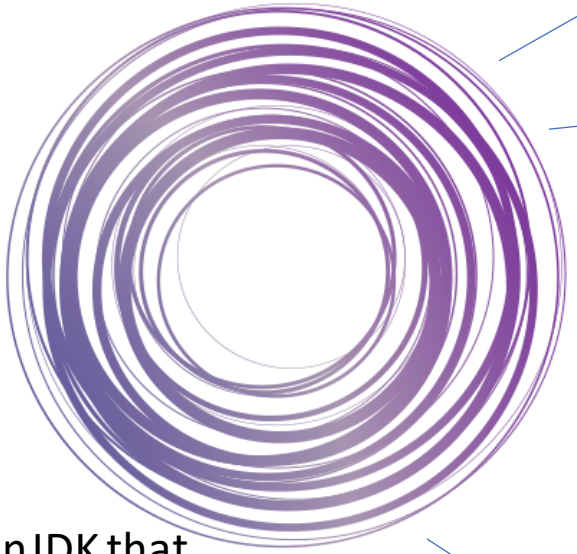




TornadoVM

www.tornadovm.org

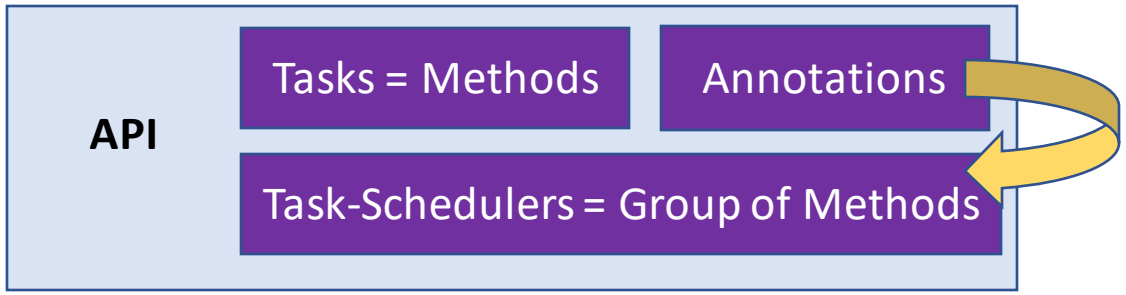
TornadoVM Overview



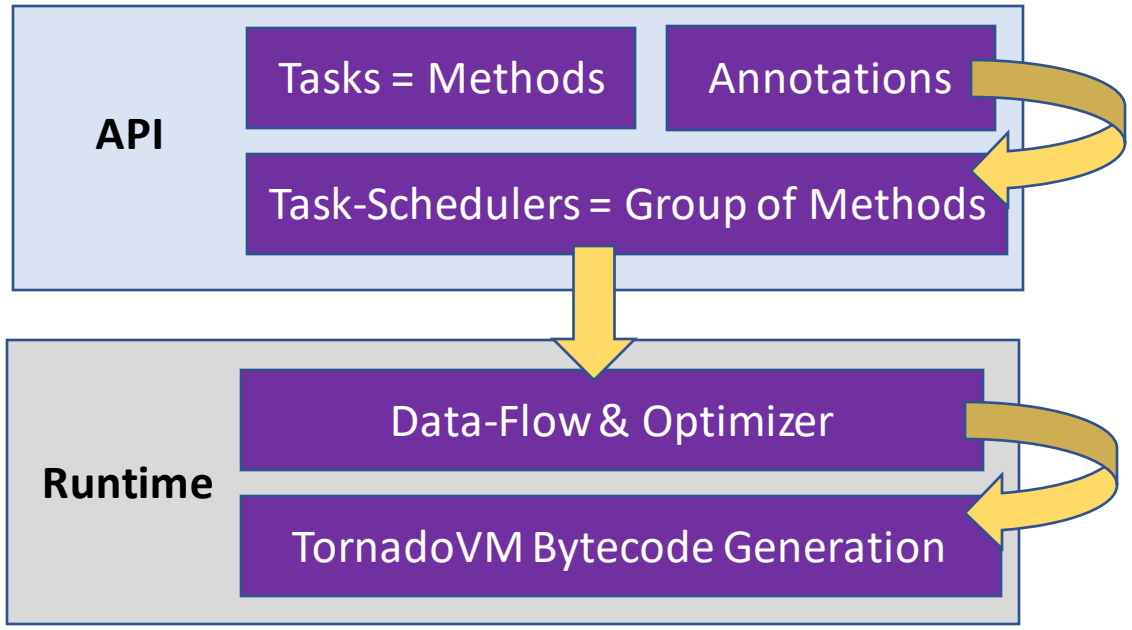
Plug-in to OpenJDK that allows developers to run JVM based programs on heterogeneous hardware

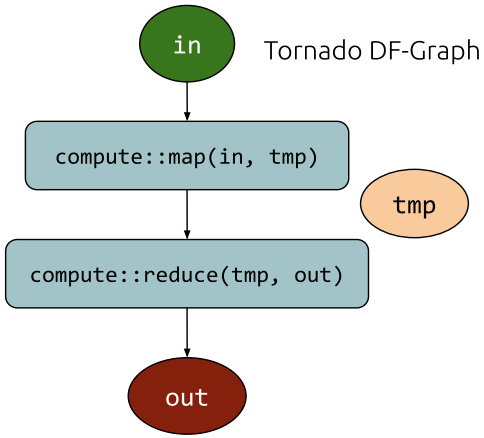
- OpenJDK 8 + JVMCI
- OpenJDK 11-14
- GraalVM
- RH Mandrel
- Amazon Corretto

TornadoVM Overview

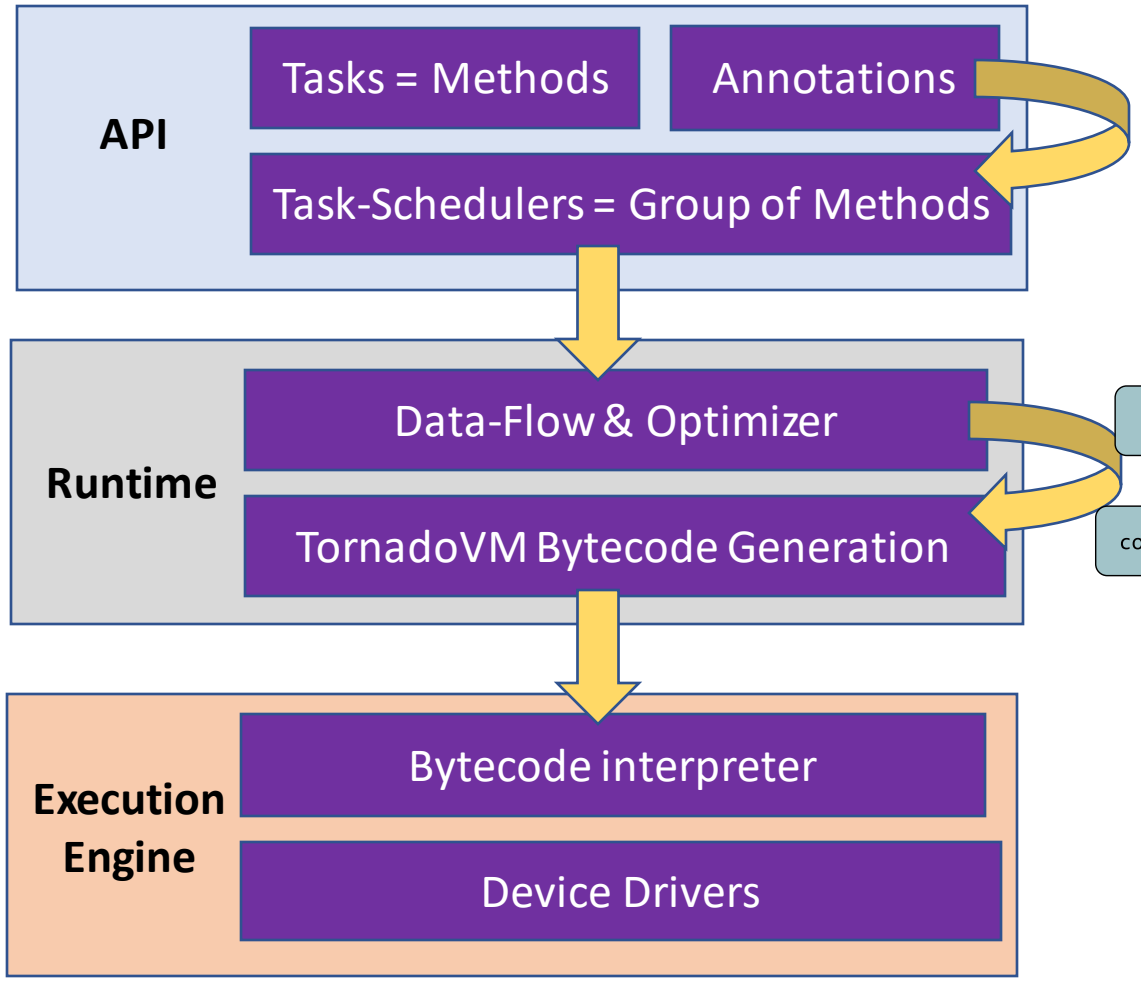


TornadoVM Overview

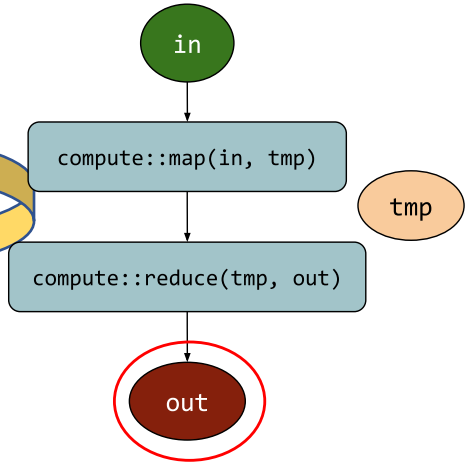




TornadoVM Overview



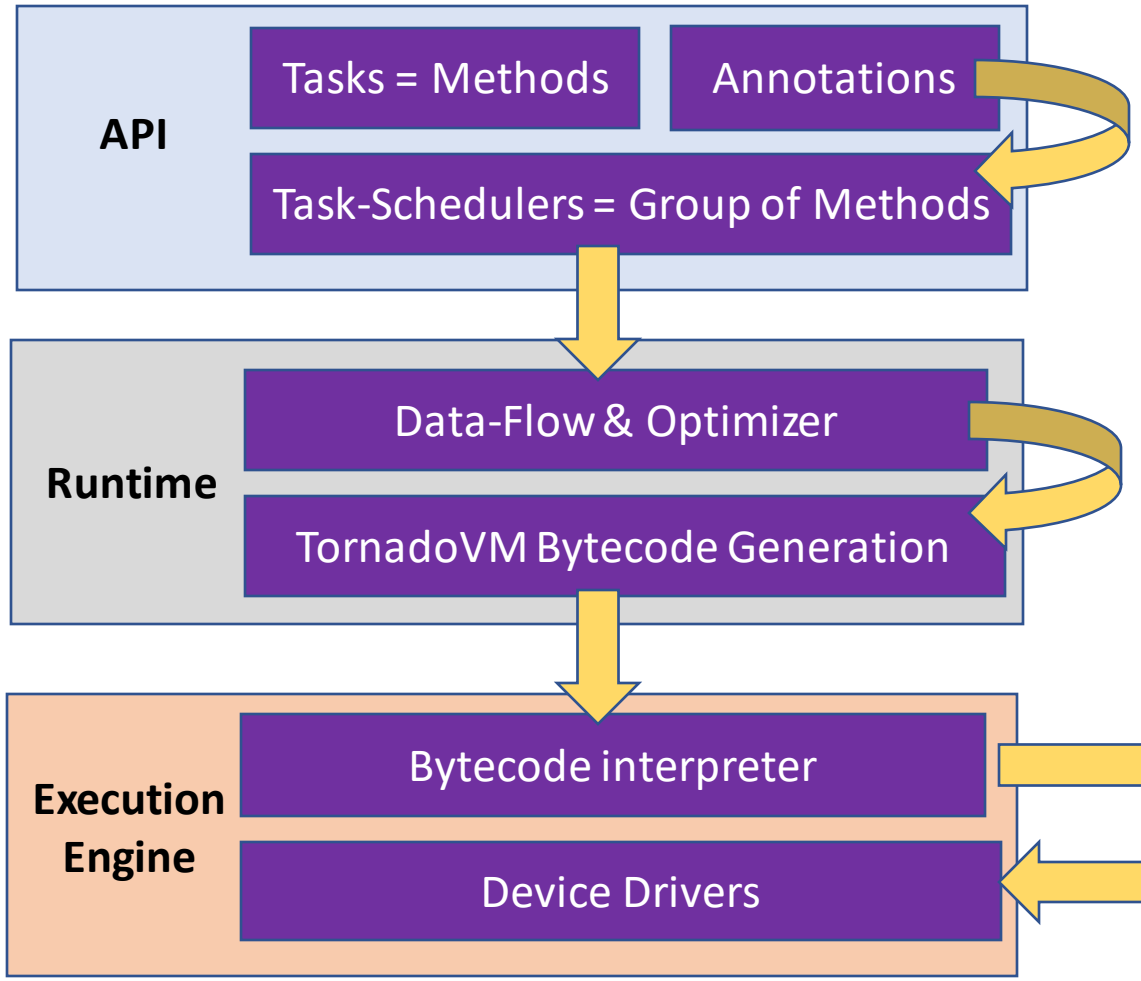
Tornado DF-Graph



TornadoVM Bytecodes

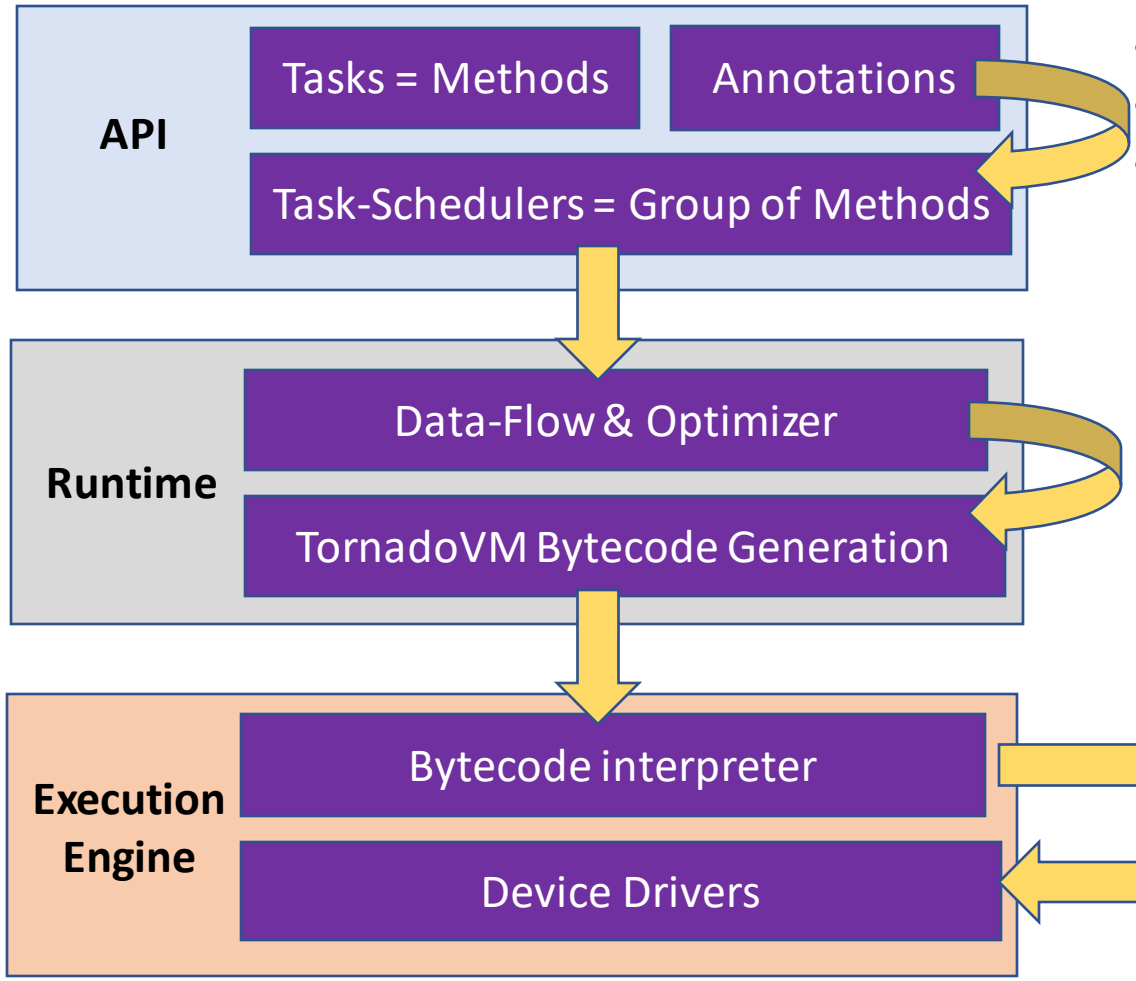
```
BEGIN    <0>              // Starts a new context
COPY_IN <0, bi1, in>      // Allocates and copies <in>
ALLOC    <0, bi2, tmp>    // Allocates <tmp> on device
ADD_DEP <0, bi1, bi2>     // Waits for copy and alloc
LAUNCH   <0, bi3, @map, in, tmp> // Runs map
ALLOC    <0, bi4, out>    // Allocates <out> on device
ADD_DEP <0, bi3, bi4>     // Waits for alloc and map
LAUNCH   <0, bi5, @reduce, tmp, out> // Runs reduce
ADD_DEP <0, bi5>          // Wait for reduce
COPY_OUT_BLOCK <0, bi6, out> // Copies <out> back
END      <0>              // Ends context
```

TornadoVM Overview




```
.version 7.0
.target sm_61
.address_size 64

.visible .entry
computeDft_float65534_float65534_float65534_float65534(.param .u64
stack_pointer) {
.reg .f32 rfi<18>;
.reg .s64 rsd<7>;
.reg .pred rpb<3>;
.reg .s32 rsi<7>;
.reg .u64 rud<11>;
.reg .u32 rui<5>;

BLOCK_0:
ld.param.u64 rud0, [stack_pointer];
ldu.global.u64 rud1, [rud0+24];
ldu.global.u64 rud2, [rud0+32];
ldu.global.u64 rud3, [rud0+40];
ldu.global.u64 rud4, [rud0+48];
mov.u32 rui0, %tid.x;
mov.u32 rui1, %ntid.x;
mov.u32 rui2, %ctaid.x;
mad.lo.s32 rsi0, rui2, rui1, rui0;
...
```

TornadoVM Overview



- OpenJDK 8 + JVMCI >= 262
- OpenJDK 11 - 14
- GraalVM 20.2.0, Mandrel 20.2.0, Corretto 11
- OpenCL >= 1.2
- Support for:
  - OpenCL
    - NVIDIA GPUs
    - Intel HD Graphics
    - AMD GPUs
    - Intel Altera FPGAs
    - Xilinx FPGAs
    - Multi-core CPUs
    - ARM Mali GPUs

PTX:

- NVIDIA GPUs

Example - Running DFT (Discrete Fourier Transform)

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn}$$
$$= \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right)\right], \quad \text{(Eq.1)}$$

Image from Wikipedia:
https://en.wikipedia.org/wiki/Discrete_Fourier_transform

DFT Applications used in:

- Digital Signal Processing
- Spectral Analysis
- Data Compression
- Filters

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

Tornado API – DFT Example

```
public class App {
  private static void computeDft(float[] inreal, float[] inimag,
                                 float[] outreal, float[] outimag) {
   for (int k = 0; k < n; k++) { // For each output element
     float sumreal = 0;
     float sumimag = 0;
     for (int t = 0; t < n; t++) { // For each input element
        float angle = ((2 * PI * t * k) / n);
        sumreal += (inreal[t] * cos(angle)) + inimag[t] * sin(angle));
        sumimag += -(inreal[t] * sin(angle) + inimag[t] * cos(angle));
     }
     outreal[k] = sumreal;
     outimag[k] = sumimag;
     }
  }
}
```

Tornado API – DFT Example

```
public class App {
  private static void computeDft(float[] inreal, float[] inimag,
                                 float[] outreal, float[] outimag) {
    for (@Parallel int k = 0; k < n; k++) { // For each output element
      float sumreal = 0;
      float sumimag = 0;
      for (int t = 0; t < n; t++) { // For each input element
         float angle = ((2 * PI * t * k) / n);
         sumreal += (inreal[t] * cos(angle)) + inimag[t] * sin(angle));
         sumimag += -(inreal[t] * sin(angle) + inimag[t] * cos(angle));
      }
      outreal[k] = sumreal;
      outimag[k] = sumimag;
      }
  }
}
```

Tornado API – DFT Example

```
public class App {
  private static void computeDft(float[] inreal, float[] inimag,
                                 float[] outreal, float[] outimag) {
   for (@Parallel int k = 0; k < n; k++) { // For each output element
     float sumreal = 0;
     float sumimag = 0;
     for (int t = 0; t < n; t++) { // For each input element
        float angle = ((2 * PI * t * k) / n);
        sumreal += (inreal[t] * cos(angle)) + inimag[t] * sin(angle));
        sumimag += -(inreal[t] * sin(angle) + inimag[t] * cos(angle));
     }
     outreal[k] = sumreal;
     outimag[k] = sumimag;
     }
  }
}
```

```
var ts = new TaskSchedule("s0")
 .task("t0", App::computeDft, inReal, inImag, outReal, outImag)
 .streamOut(outReal, outImag).execute();
```

TornadoVM uses single source property (device code and host code in the same language)

Device Code

Host Code

Tornado API – DFT Example

```
public class App {
  private static void computeDft(float[] inreal, float[] inimag,
                                 float[] outreal, float[] outimag) {
   for (@Parallel int k = 0; k < n; k++) { // For each output element
     float sumreal = 0;
     float sumimag = 0;
     for (int t = 0; t < n; t++) { // For each input element
        float angle = ((2 * PI * t * k) / n);
        sumreal += (inreal[t] * cos(angle)) + inimag[t] * sin(angle));
        sumimag += -(inreal[t] * sin(angle) + inimag[t] * cos(angle));
     }
     outreal[k] = sumreal;
     outimag[k] = sumimag;
     }
  }
}
```

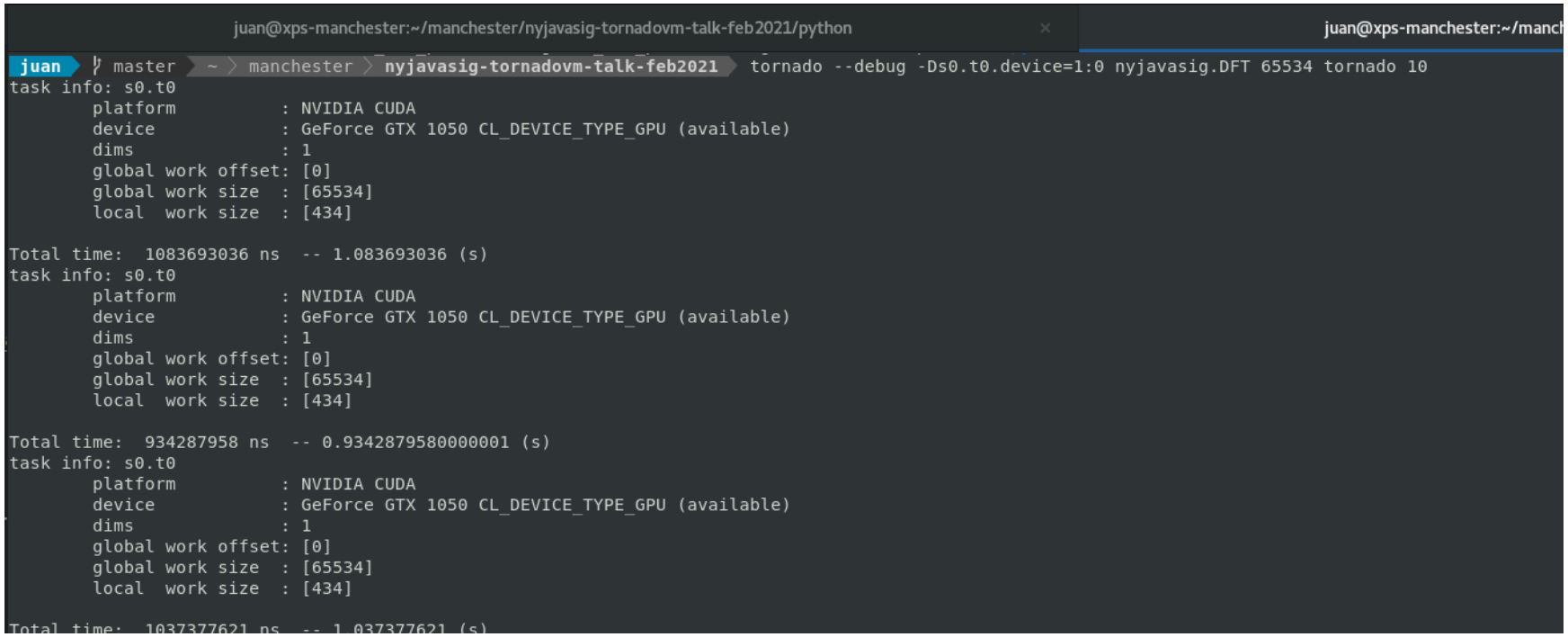
```
var ts = new TaskSchedule("s0")
 .task("t0", App::computeDft, inReal, inImag, outReal, outImag)
 .streamOut(outReal, outImag).execute();
```

To run:

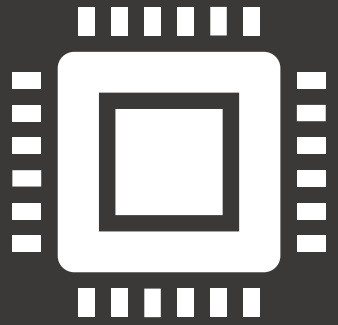
```
$ tornado App
```

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

DEMO - Running DFT



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



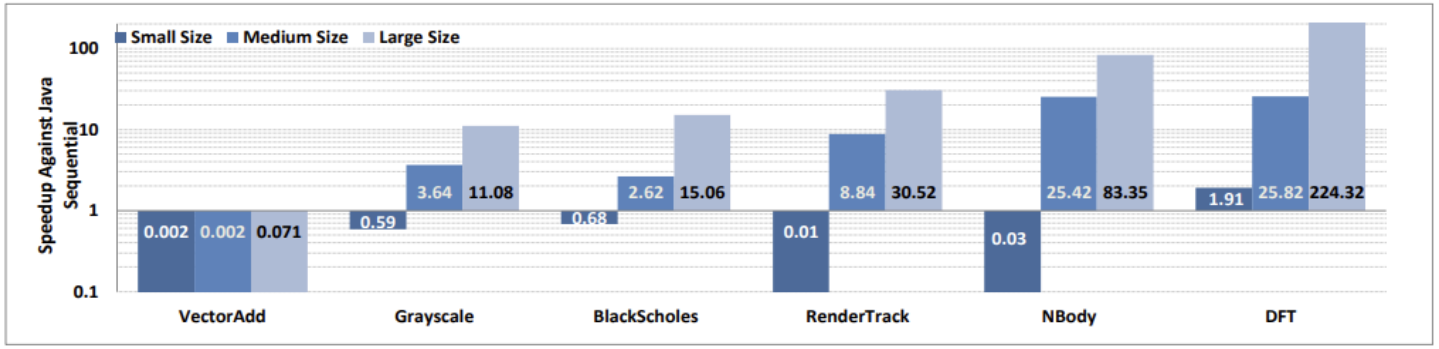
TornadoVM and FPGAs

Demo on FPGAs!

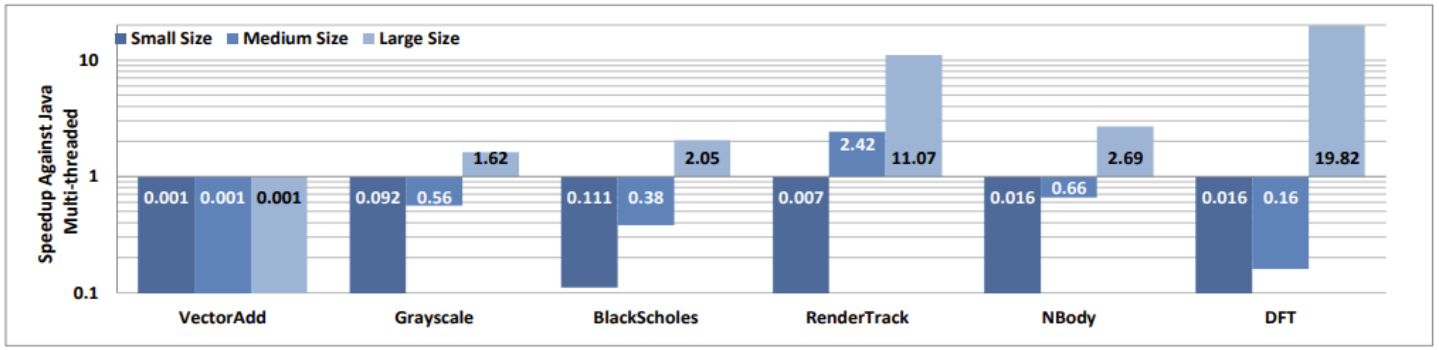
- Demo 1: DFT (same as before)
- Demo 2: BlackScholes (Kernel based on Fintech algorithms)

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

Results for FPGAs



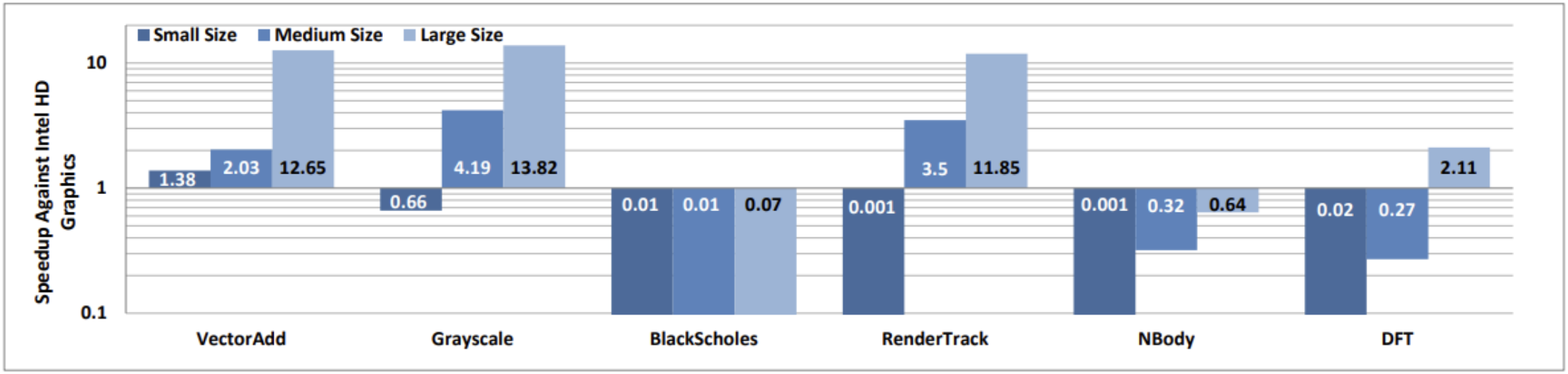
A) Speedup against Sequential Java Hotspot. 224x compared to Java Hotspot



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

TornadoVM on FPGA vs TornadoVM on Intel HD630



(The higher the better)
Up to 13x

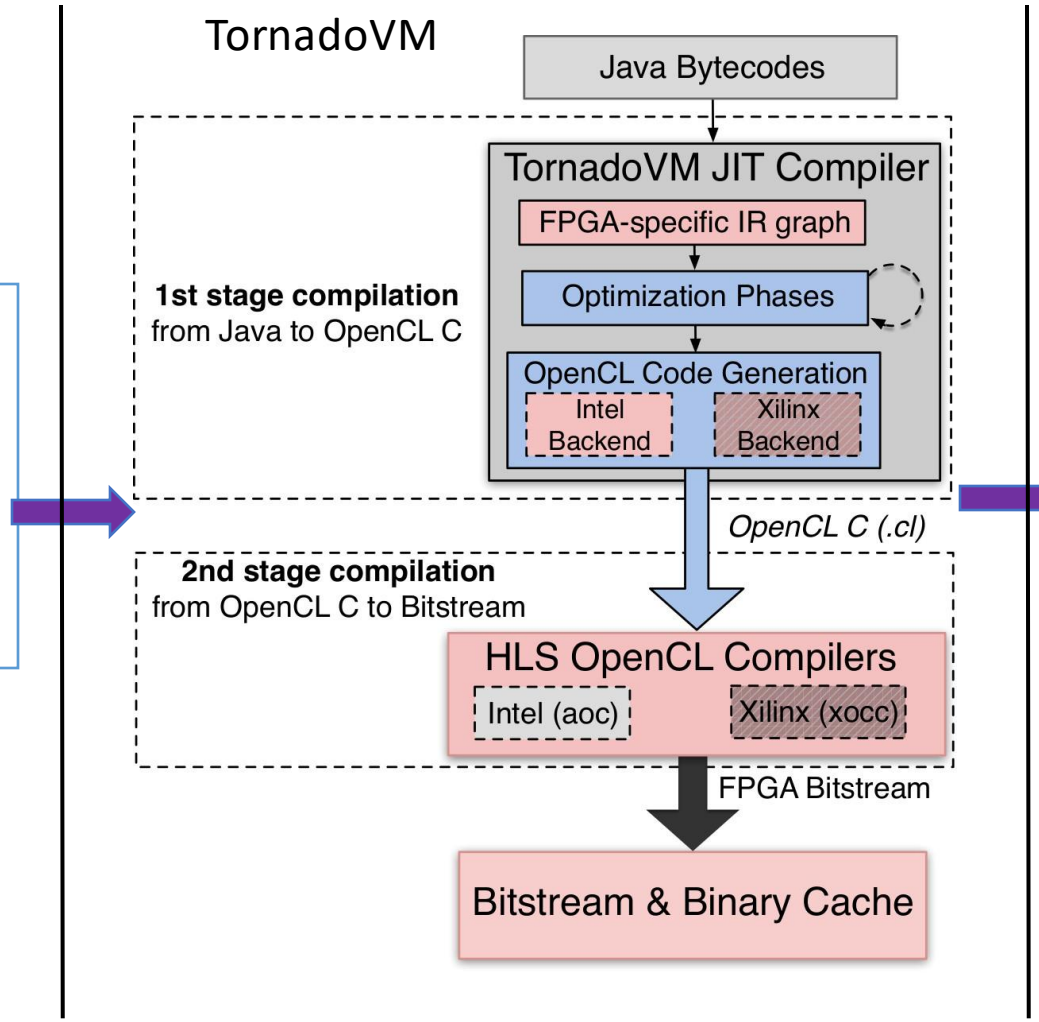
Michail Papadimitriou, Juan Fumero, Athanasios Stratikopoulos, Foivos S. Zakkak, Christos Kotselidis. ***Transparent Compiler and Runtime Specializations for Accelerating Managed Languages on FPGAs.*** Programming 2021 - Volume 5 - Issue 2

FPGA Workflow within TornadoVM

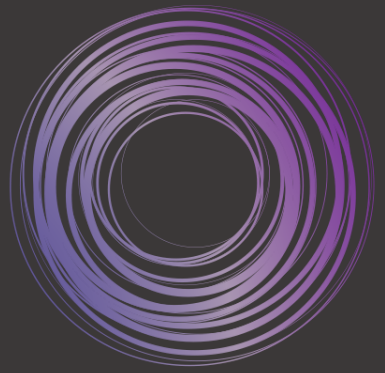
Java

```
void compute(float[] input,
             float[] output) {
 for (@Parallel int i = 0; …) }
   for (@Parallel int j = 0; ...)
{
       // Computation
     }
  }
}
```

Program FPGAs within your favourite IDE: Eclipse, IntelliJ, …



Physical hardware



API and New Features

Expanding the TornadoVM API

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

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**

Expressing Kernel Parallelism within TornadoVM

TornadoVM's default API

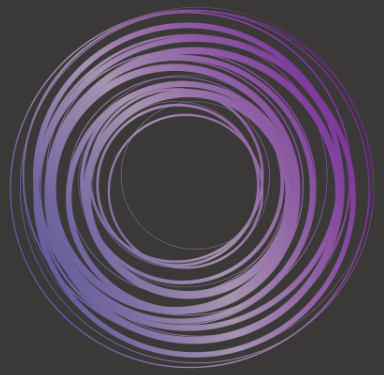
```
public void vectorAdd(float[] a,
                      float[] b,
                      float[] c) {
  for (@Parallel int i = 0; i < c.length; i++){
    c[i] = a[i] + b[i];
  }
}
```

It exploits loop parallelism

TornadoVM's New API

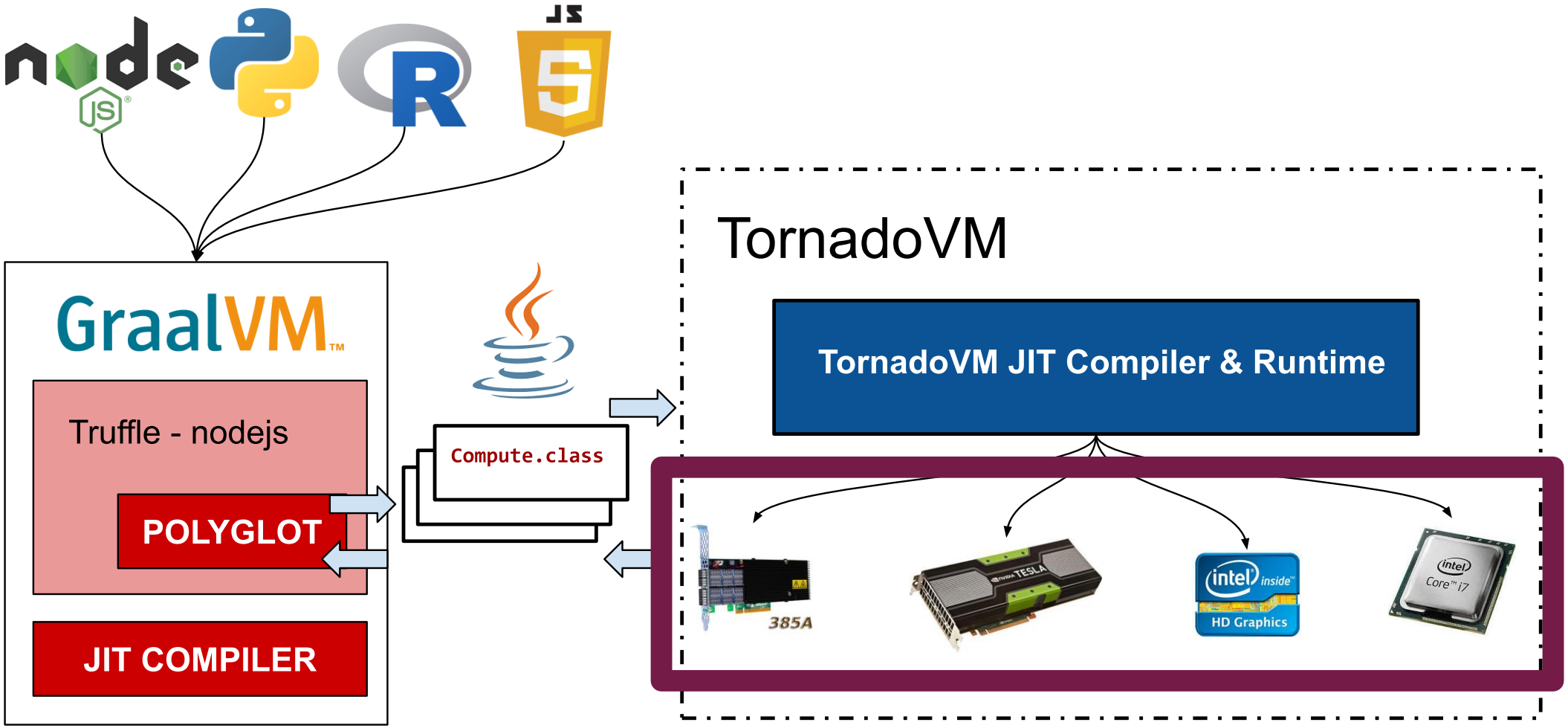
```
public void vectorAdd(float[] a,
                      float[] b,
                      float[] c,
                      TornadoVMContext context) {
  int i = context.threadIdx;
  c[i] = a[i] + b[i];
}
```

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.



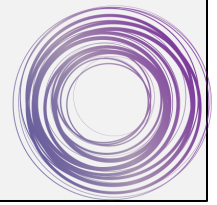
TornadoVM for JVM based Programming Languages

TornadoVM & Dynamic/Untyped Programming Languages



Using TornadoVM with Truffle 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;
  }
}
```



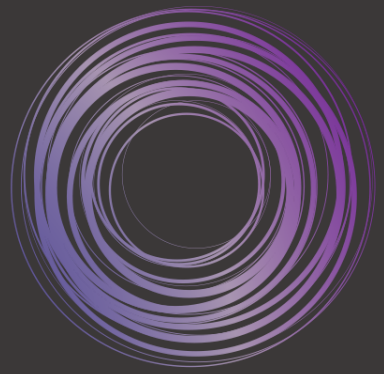
```
myCompute <- java.type('MyCompute');
output <- myCompute.compute();
```

```
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/

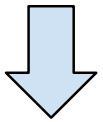


TornadoVM Internals of the JIT Compiler

TornadoVM Compiler & Runtime Overview

Input: Java Method

```
public void compute(int[] a, int[] b) {
  for (@Parallel int i = 0; i < n; i++) {
    b[i] = a[i] + b[i];
  }
}
```



TornadoVM JIT Compiler & Runtime

Output: OpenCL Kernel

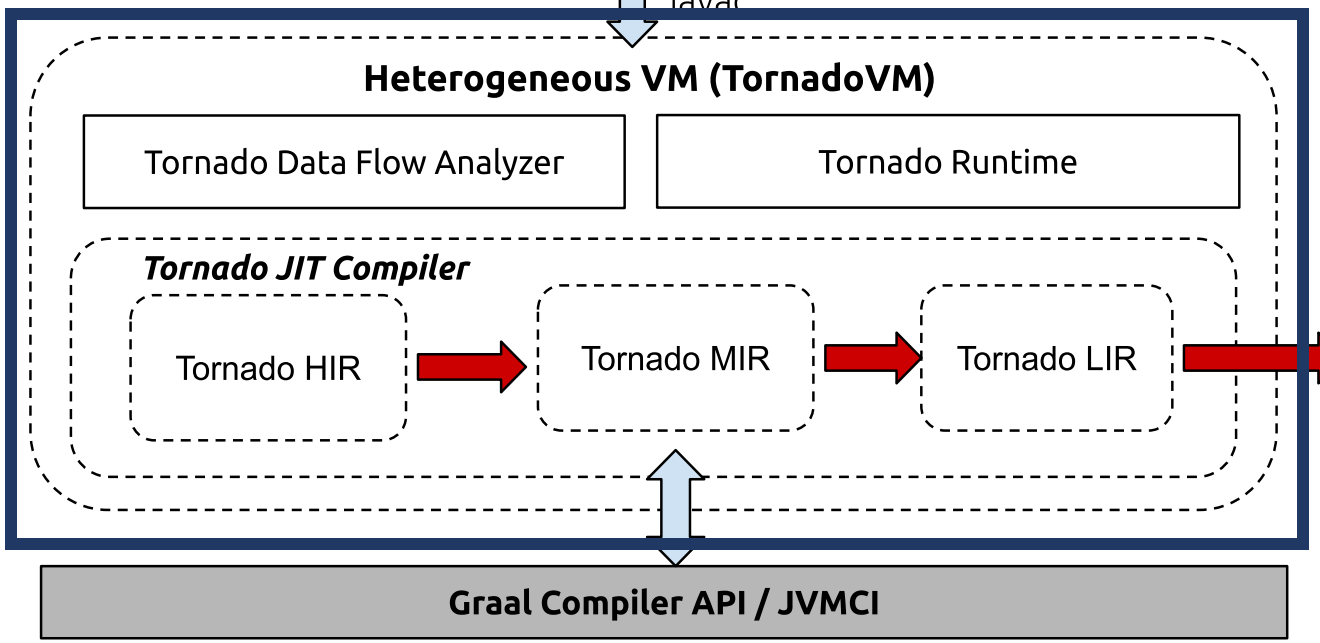
```
kernel void compute(global uchar *frame, ...) {
  // data types declaration
  ...
  ul_0  =  (ulong) frame[7];
  ul_1  =  (ulong) frame[8];
  i_2   =  get_global_id(0);
  i_3   =  i_2;

  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;
  }
}
```

TornadoVM Compiler & Runtime Overview

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



Output: OpenCL Kernel

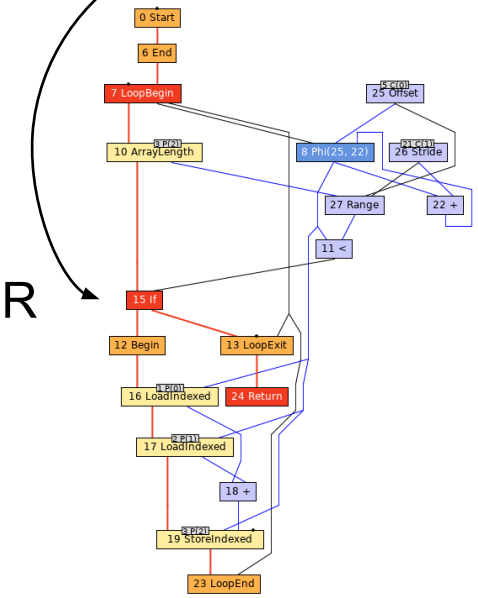
```
kernel void compute(global uchar *frame, ...) {
  // data types declaration
  ...
  ul_0  =  (ulong) frame[7];
  ul_1  =  (ulong) frame[8];
  i_2  =  get_global_id(0);
  i_3  =  i_2;

  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;
  }
}
```

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



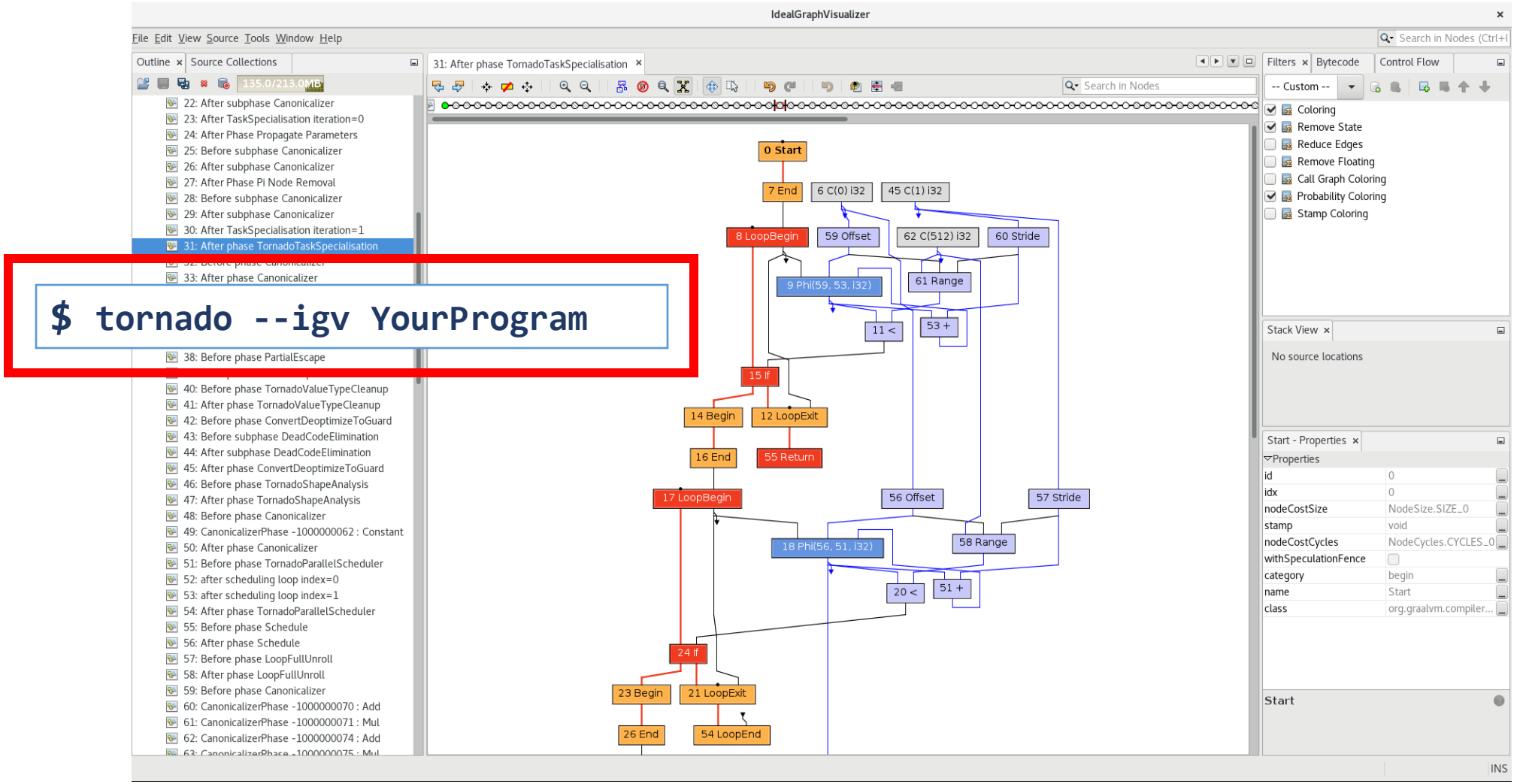
GPU Specialization


```
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];
}
```

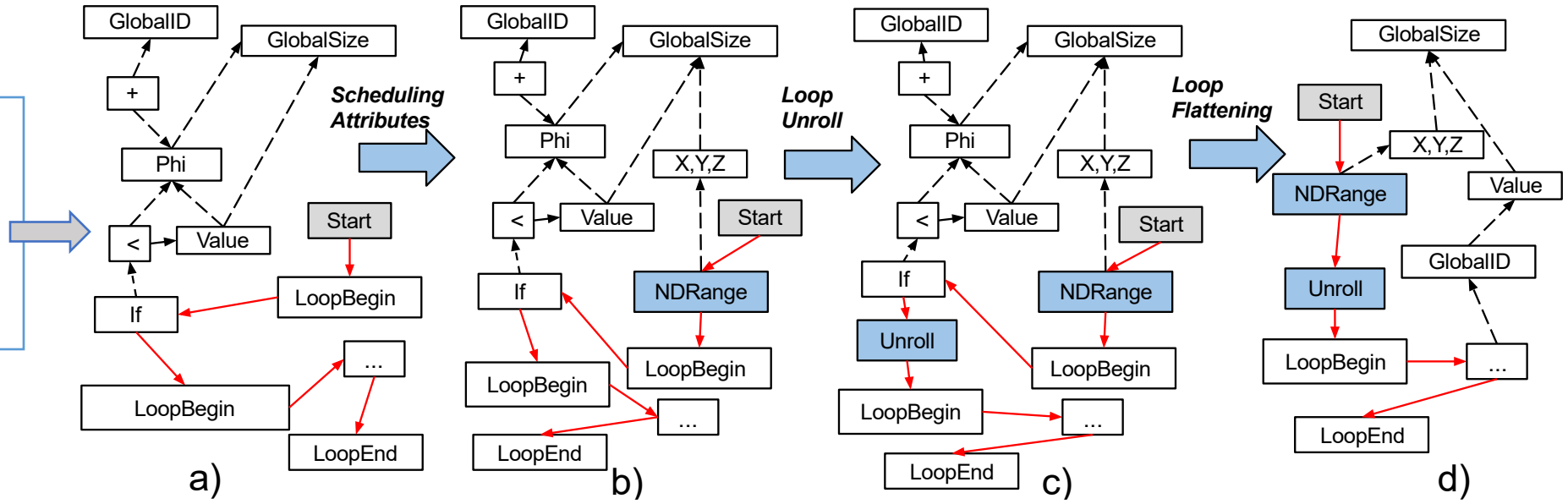
Demo - Code Specialization using the Graal JIT Compiler




```
$ tornado --igv YourProgram
```

FPGA Specializations for Achieving Performance

```
void compute(float[] input,
             float[] output) {
 for (@Parallel int i = 0; …) }
   for (int j = 0; ...) {
       // Computation
     }
  }
}
```

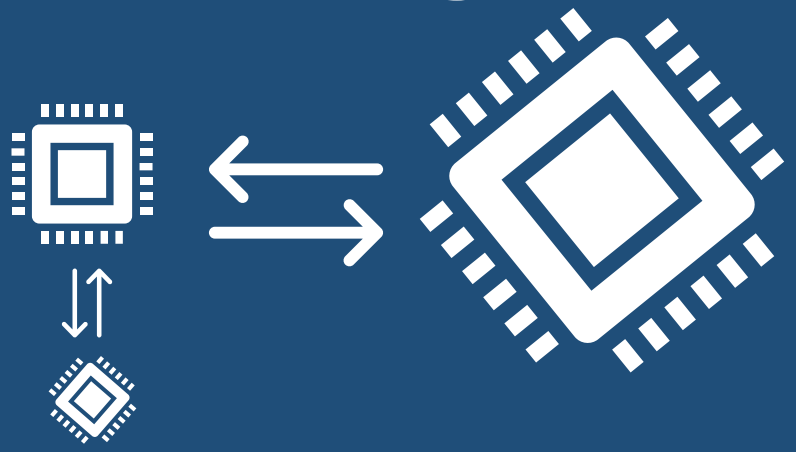


From slowdowns without Specializations to 220x with Automatic Specializations on Intel FPGAs

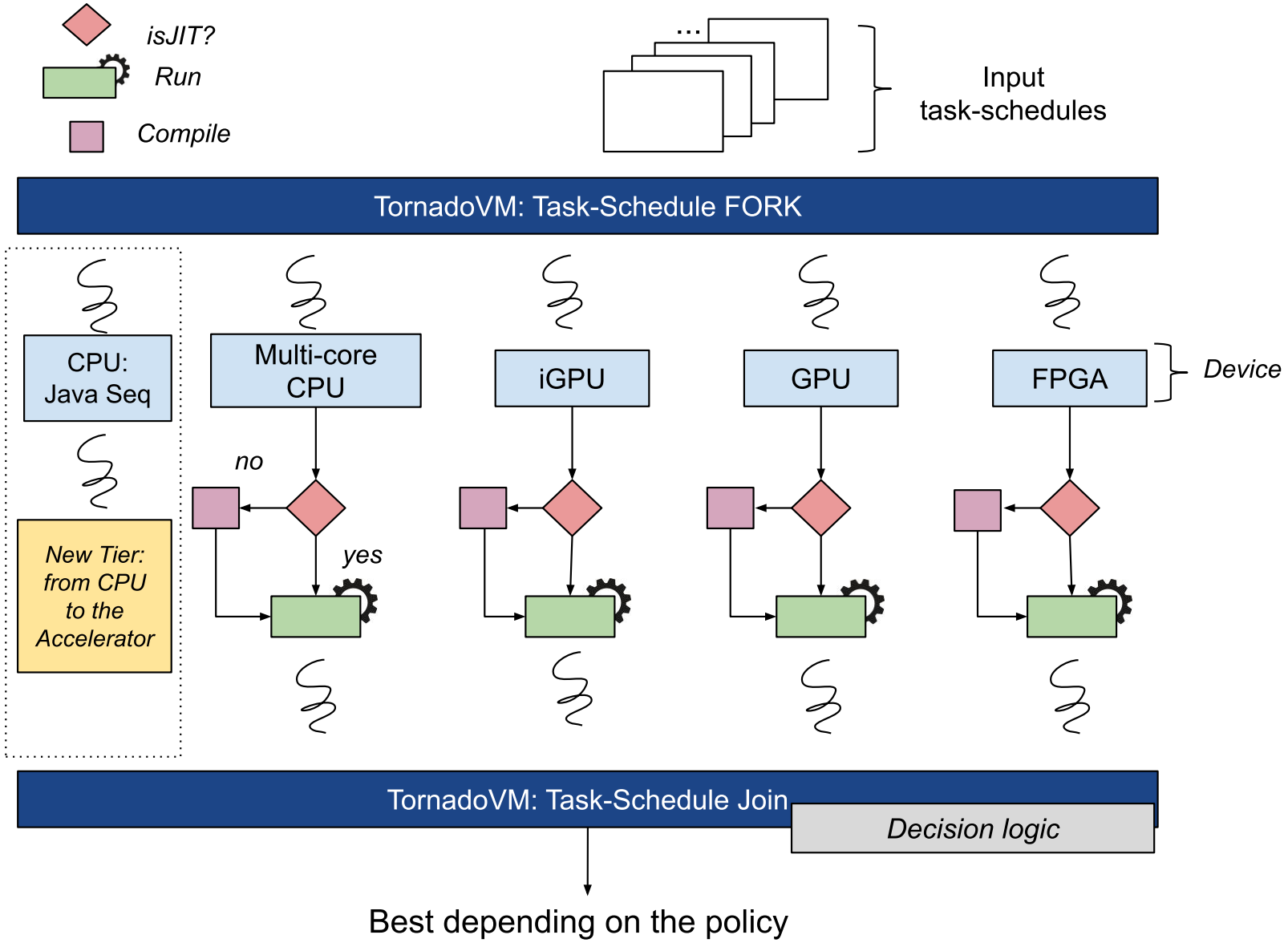
Michail Papadimitriou, Juan Fumero, Athanasios Stratikopoulos, Foivos S. Zakkak, Christos Kotselidis. ***Transparent Compiler and Runtime Specializations for Accelerating Managed Languages on FPGAs.*** Programming 2021 - Volume 5 - Issue 2

MANCHESTER 1824
The University of Manchester

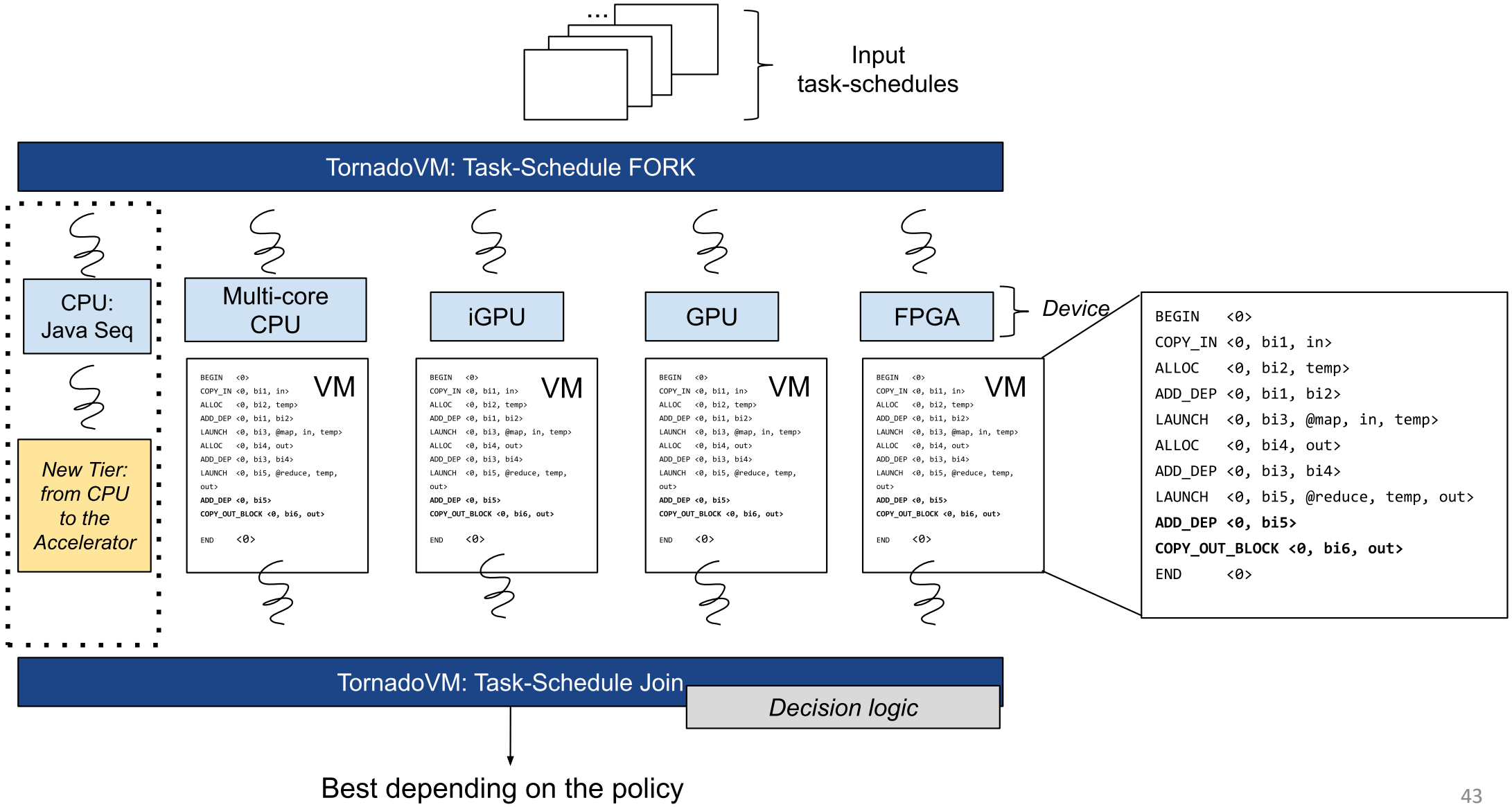
Live Task Migration



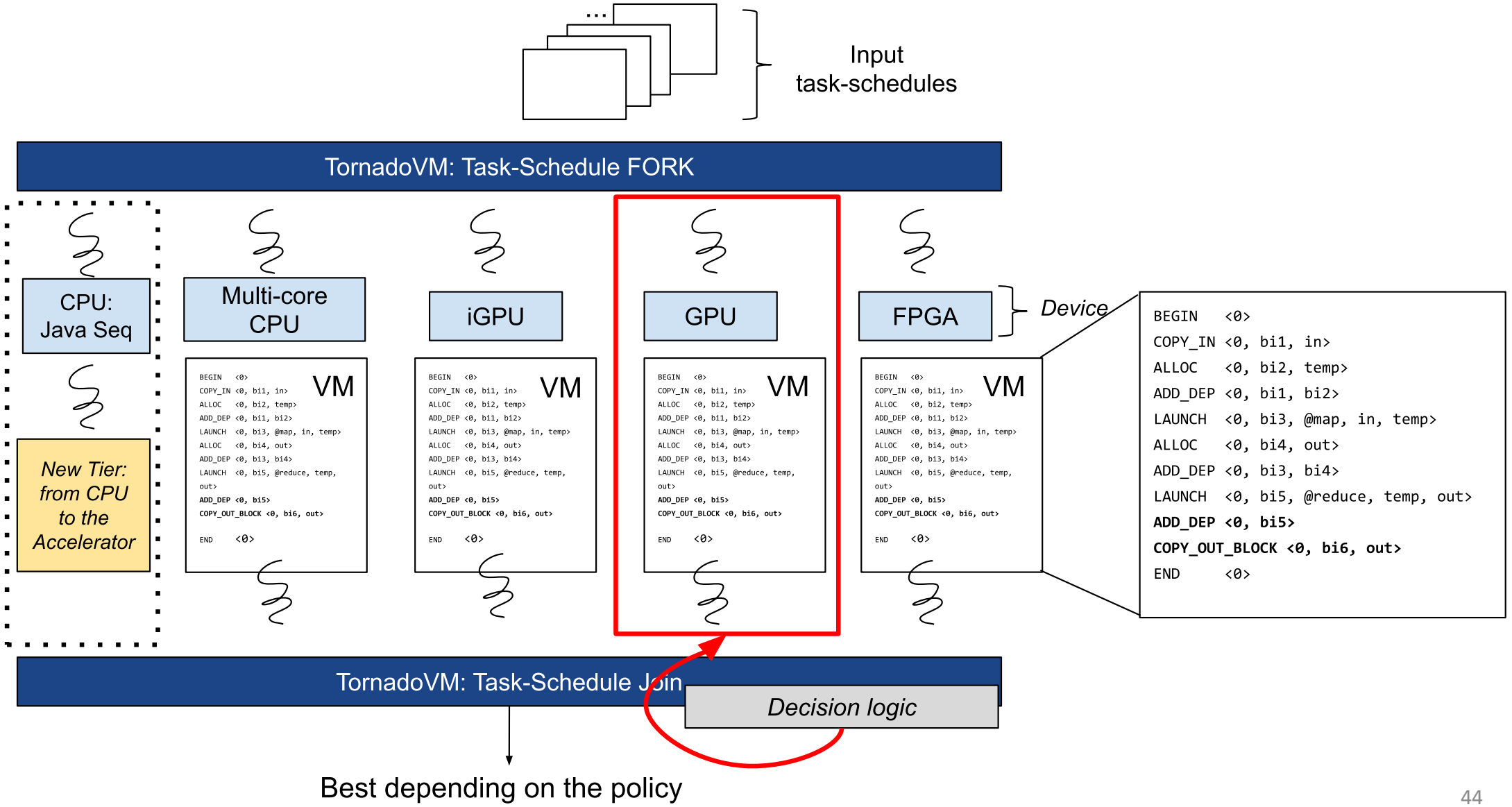
Live Task Migration



Live Task Migration



Live Task Migration



How is the decision made?

- End-to-end: including JIT compilation time
- Peak Performance: without JIT and after warming-up
- Latency: does not wait for all threads to finish

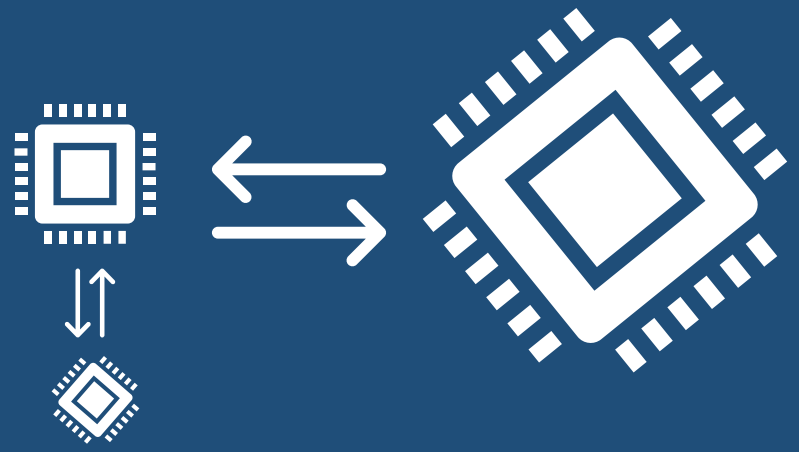
```
// END-TO-END PERFORMANCE
ts.task(Compute::add, a, b, c)
  .streamOut(c)
  .execute(Profiler.END2END);
```

```
// PEAK PERFORMANCE
ts.task(Compute::add, a, b, c)
  .streamOut(c)
  .execute(Profiler.PERFORMANCE);
```

```
// LATENCY
ts.task(Compute::add, a, b, c)
  .streamOut(c)
  .execute(Profiler.LATENCY);
```

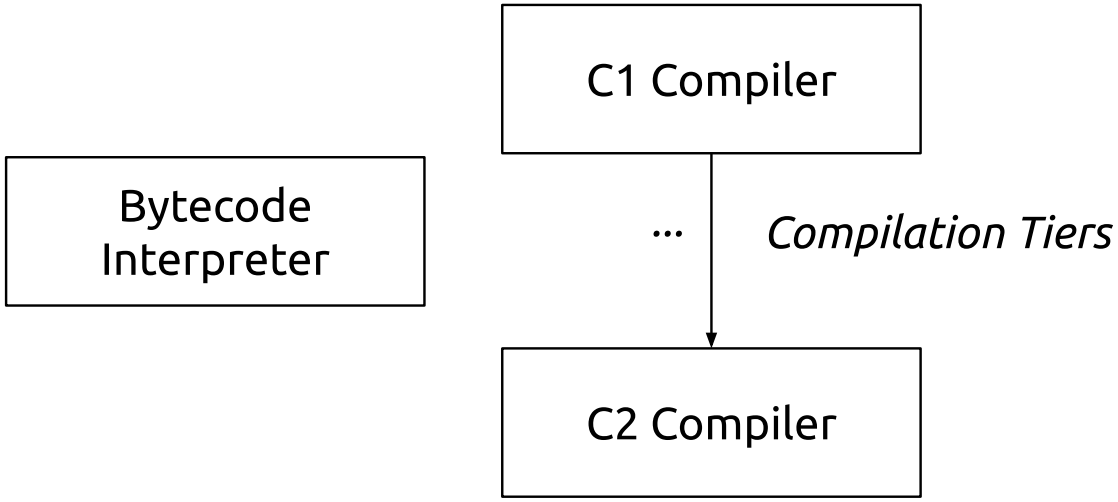
MANCHESTER 1824
The University of Manchester

Demo Live Task Migration – Server/Client App

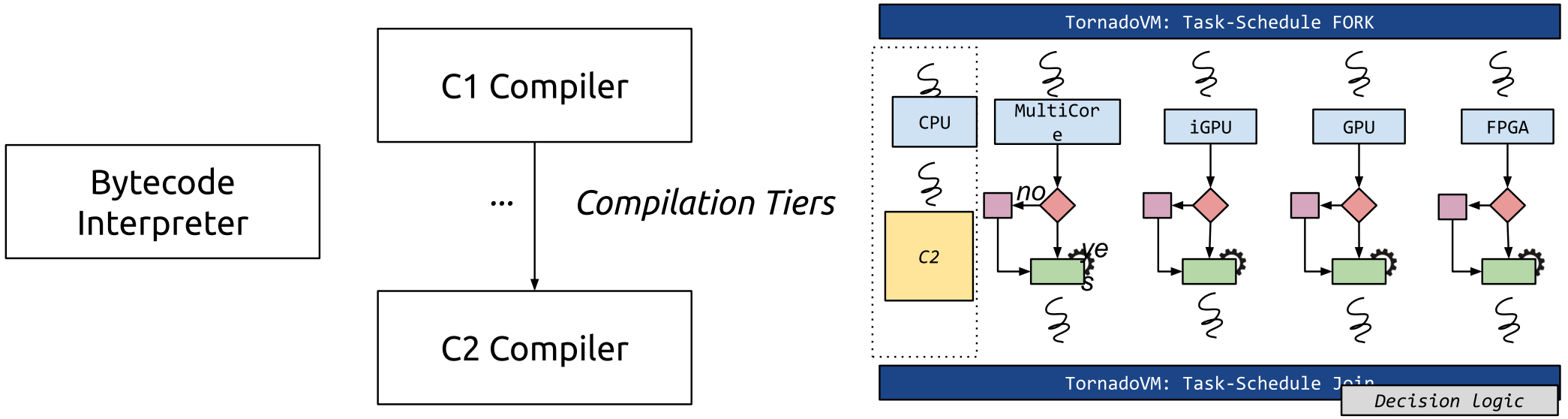


https://github.com/jjfumero/nyjavasig-tornadovm-2021

New compilation tier for Heterogeneous Systems



New compilation tier for Heterogeneous Systems

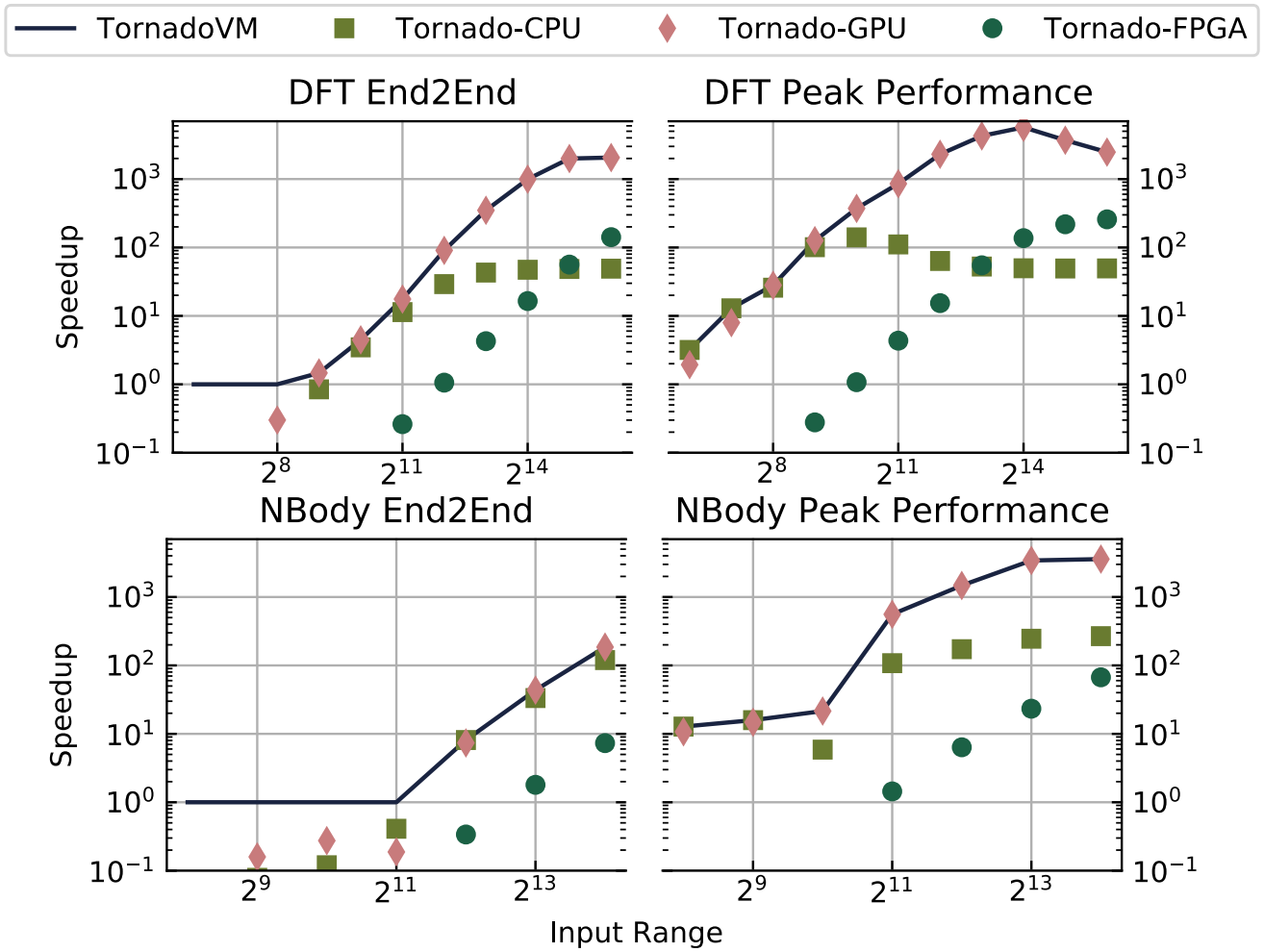


E.g., From C2 -> Multi-core -> GPU

MANCHESTER 1824
The University of Manchester

Ok, cool! What about performance?

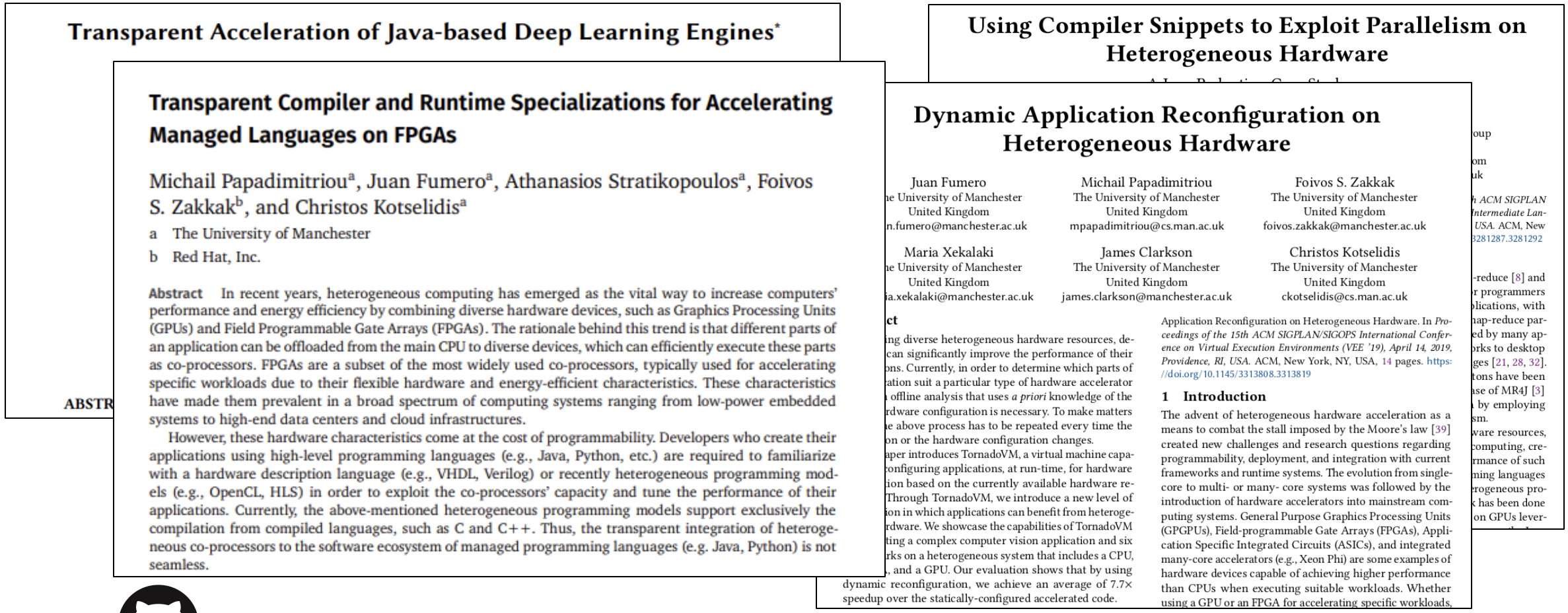
Performance of Dynamic Reconfiguration



* TornadoVM performs up to 7.7x over the best device (statically).
* Up to >**4500x** over Java sequential

- *NVIDIA GTX 1060*
- *Intel FPGA Nallatech 385a*
- *Intel Core* i7-7700K

More details in our papers!



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
n.fumero@manchester.ac.uk

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
ia.xekalaki@manchester.ac.uk

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/14_PUBLICATIONS.md

MANCHESTER 1824
The University of Manchester

Related Work

Related Work (in the Java context)

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

Related Work (in the Java context)

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

MANCHESTER 1824
The University of Manchester

Future Work

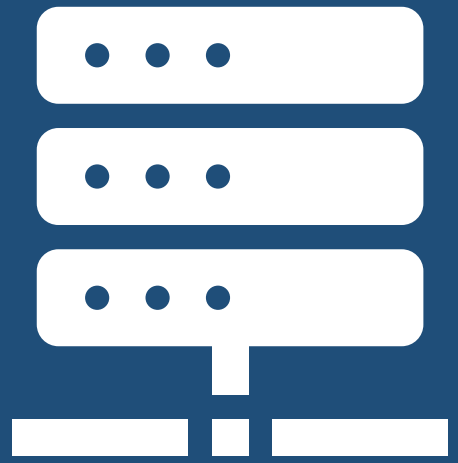
Work in Progress & Future Work

- [In progress] New API for advanced users (Kernel Parallelism)
- [In progress] **SPIR-V Backend** (In collaboration with Intel Labs)

- Policies for energy efficiency
- Multi-device within a task-schedule
- More parallel skeletons (scans, filters, do-WHILE, …)

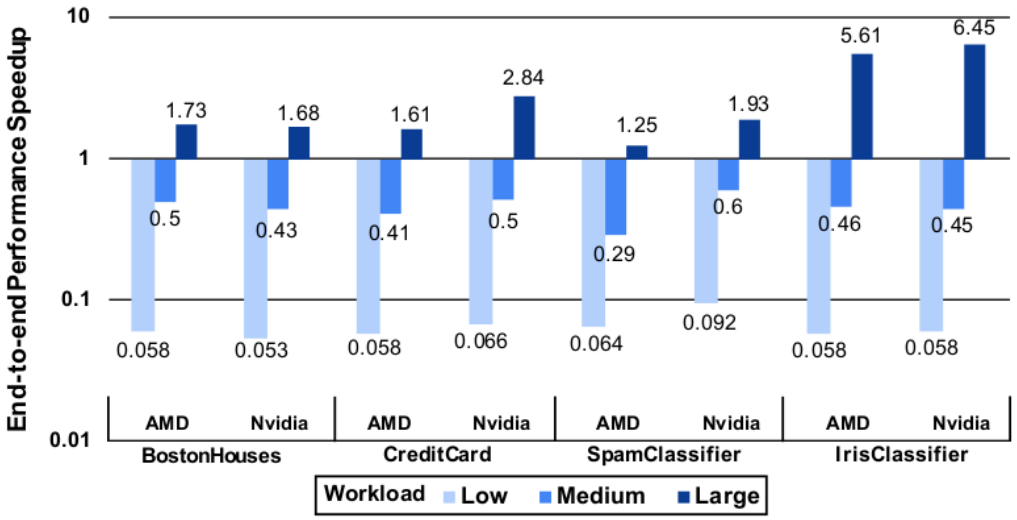
MANCHESTER 1824
The University of Manchester

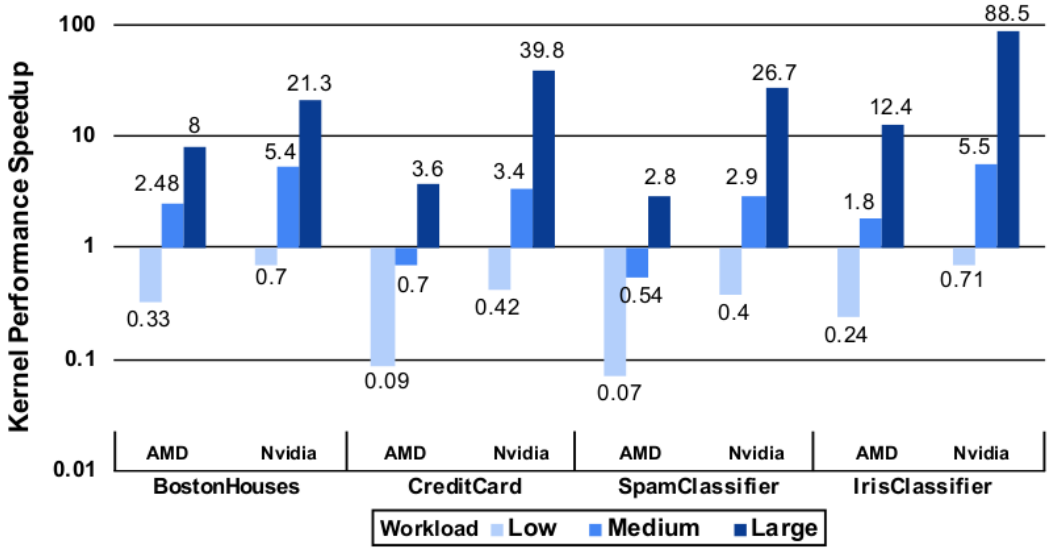
Current Applicability of TornadoVM

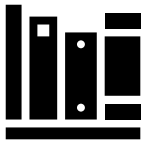


Integration within Existing Frameworks

- E.g., Acceleration of DeepNetts: https://www.deepnetts.com/







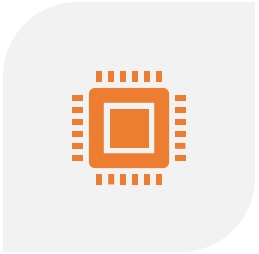
Athanasios Stratikopoulos et al. ***Transparent acceleration of Java-based deep learning engines.*** *Proceedings of the 17th International Conference on Managed Programming Languages and Runtimes. DOI: https://doi.org/10.1145/3426182.3426188*

MANCHESTER 1824
The University of Manchester

Companies Using TornadoVM

EXUS
(MACHINE & DEEP LEARNING)

NEUROCOM
(NATUAL LANGUAGE PROCESSING & FINTECH)



IPROOV LONDON
(COMPUTER VISION)

SPARKWORKS
(SMART BUILDINGS)

OPTRACK
(VEHICLE ROUTING)

MANCHESTER 1824
The University of Manchester

Health Care and Machine Learning

EXUS

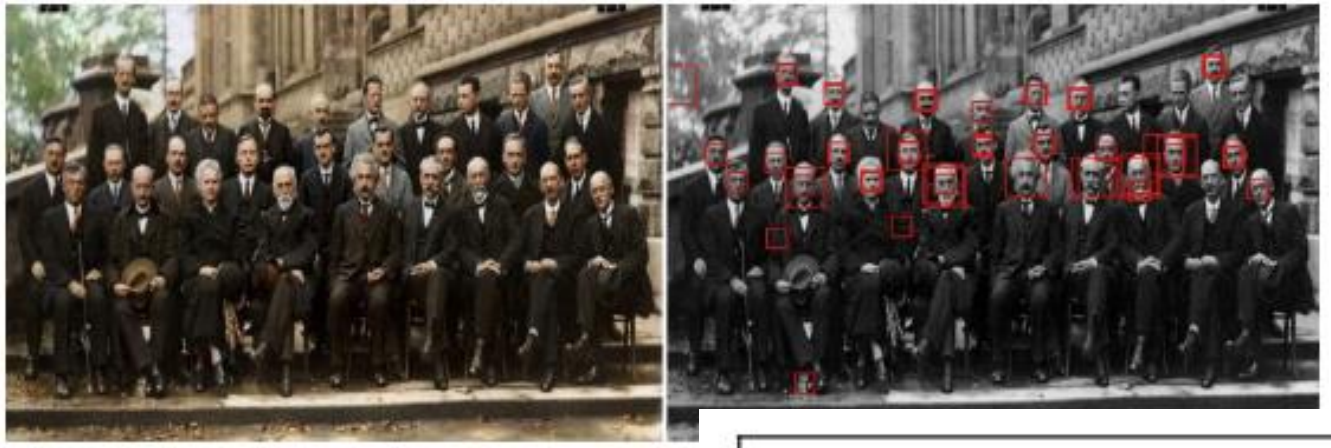
Using TornadoVM for the training phase (2M patients):

From ~2615s to 185s (14x)

Thanks to Gerald Mema from Exus for sharing the numbers and the use case

Iproov London

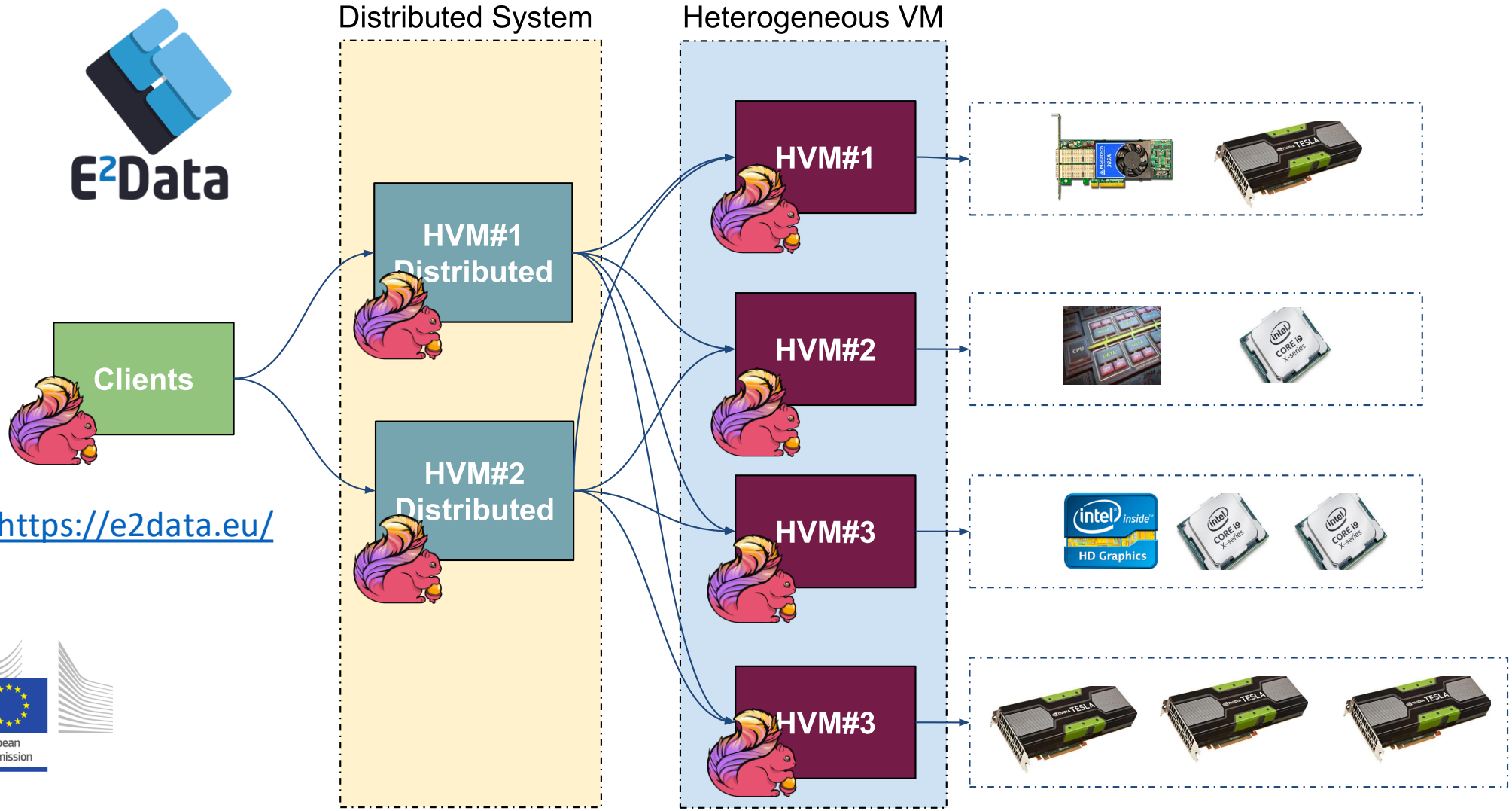
Accelerating Face-Detection with TornadoVM by Gary Frost (Iproov)



	Relative Improvement over Sequential Java (higher is better)			
	600x408	844x612	1578x976	2560x1738
Sequential Java	1	1	1	1
Multithreaded Java	3.4	4.6	5.8	6.7
TornadoVM AMD GPU OpenCL Runtime	15.6	20.4	22.3	20.0
TornadoVM Intel CPU OpenCL Runtime	13.7	17.1	23	22.9

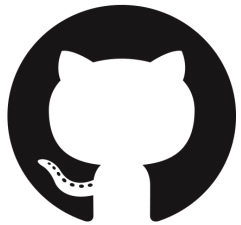
https://e2data.eu/blog/java-gpu-accelerated-viola-jones-face-detection-with-tornadovm

E2Data Project – Distributed H. System with Apache Flink & TornadoVM

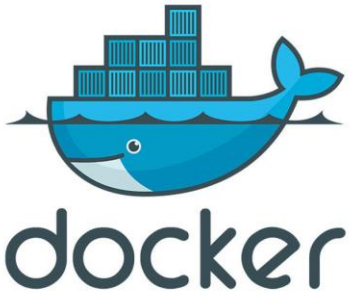


To sum up …

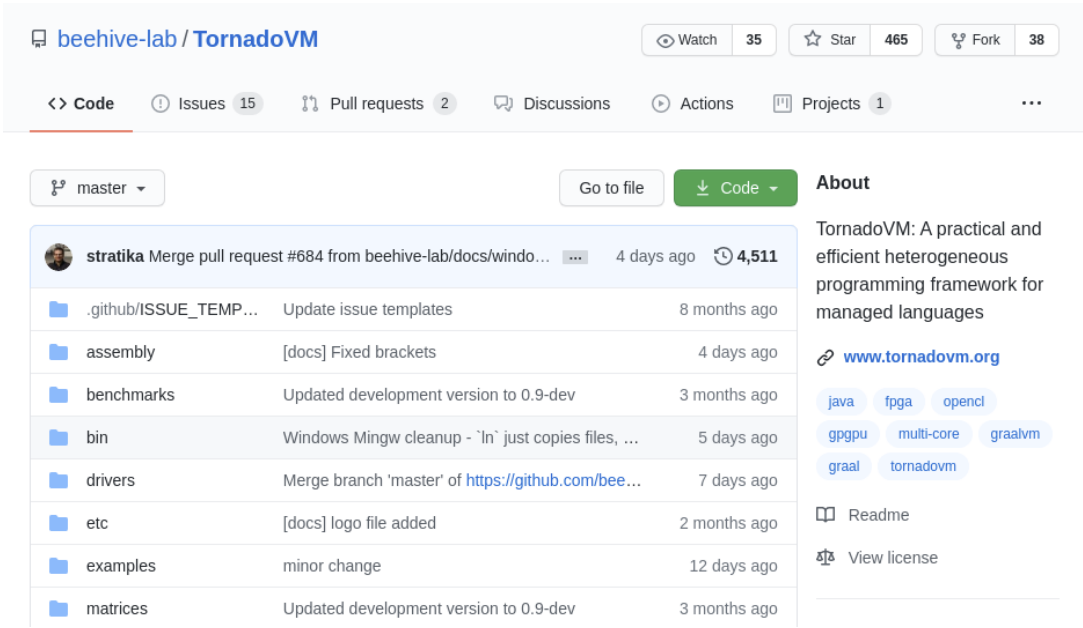
TornadoVM is OpenSource and available on Github and DockerHub



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



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



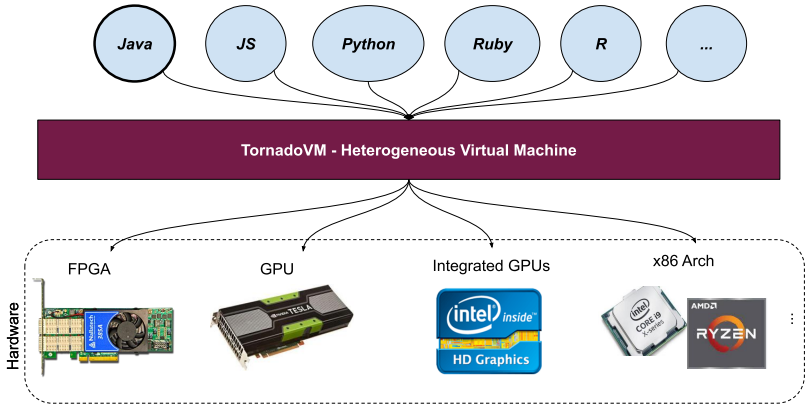

```
$ docker pull beehivelab/tornado-gpu

#And run!
$ ./run_nvidia.sh javac.py YourApp
$ ./run_nvidia.sh tornado YourApp
```

Team

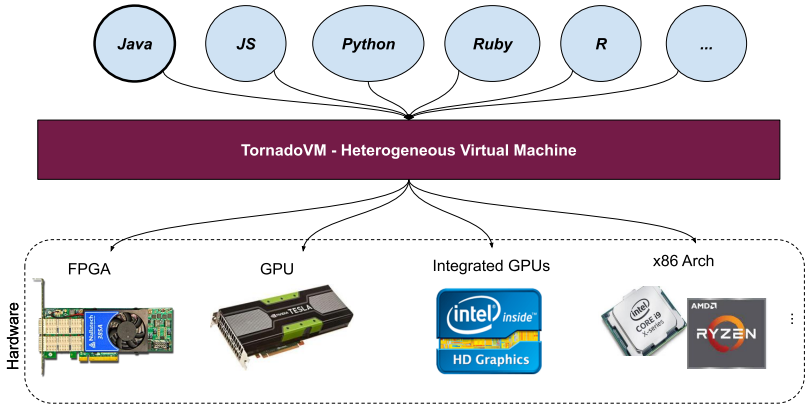
- Academic staff:
  - Christos Kotselidis
- Research staff:
  - Juan Fumero
  - Athanasios Stratikopoulos
  - Nikos Foutris
- PhD Students:
  - Michail Papadimitriou
  - Maria Xekalaki
- Master Students
  - Florin Blanaru
- Alumni:
  - James Clarkson
  - Benjamin Bell
  - Amad Aslam
  - Foivos Zakkak
  - Gyorgy Rethy
  - Mihai-Christian Olteanu
  - Ian Vaughan

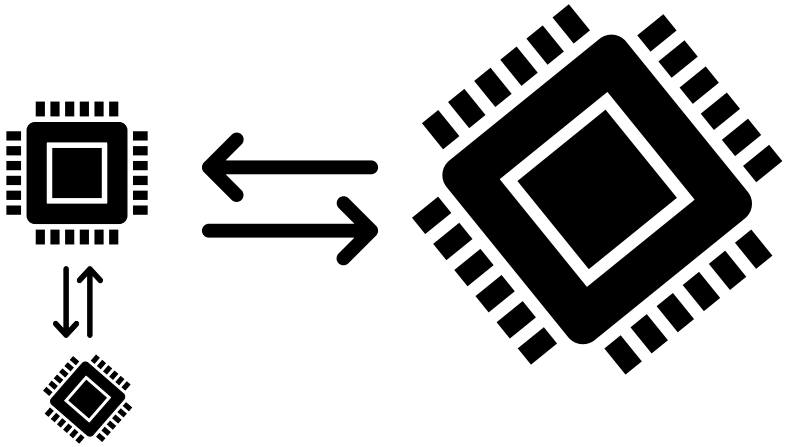
We are looking for collaborations (industrial & academics) -> Let's talk!

Takeaways

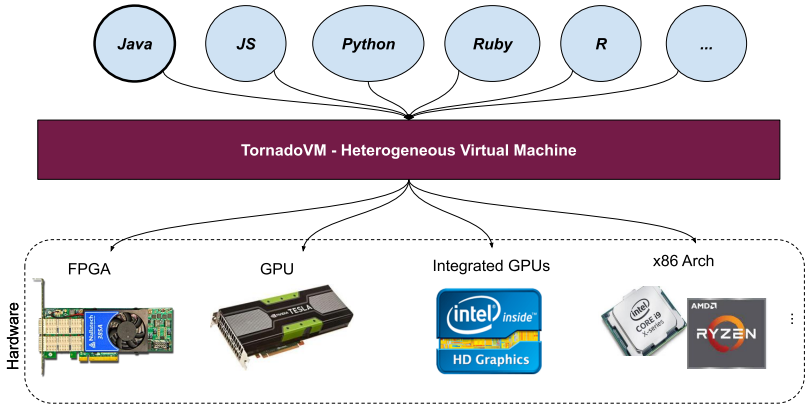

Java
JS
Python
Ruby
R
...
TornadoVM - Heterogeneous Virtual Machine
Hardware
FPGA
GPU
Integrated GPUs
x86 Arch
HD Graphics
RYZEN

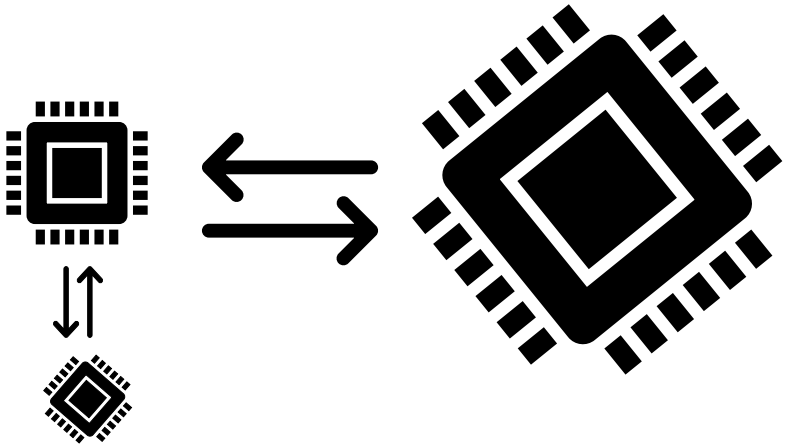
Takeaways

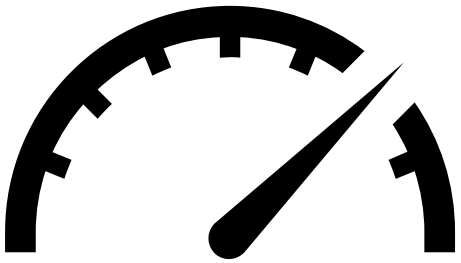

Java
JS
Python
Ruby
R
...
TornadoVM - Heterogeneous Virtual Machine
Hardware
FPGA
GPU
Integrated GPUs
x86 Arch
intel inside
HD Graphics
AMD
RYZEN



Takeaways







> 4500x vs Hotspot

E²Data

https://e2data.eu

SCAN ME

tornadovm.org

Thank you so much for your attention

- This work is partially supported by the EU Horizon 2020 **E2Data** 780245

- Partially supported by **Intel Labs Grant**

Contact: Juan Fumero
<juan.fumero@manchester.ac.uk>