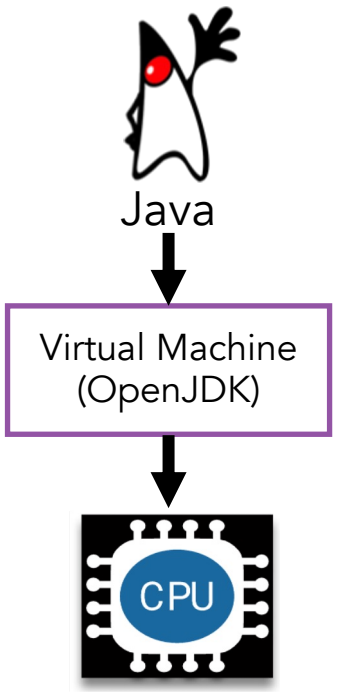
Write Once, Run Anywhere... Well, What About Heterogeneous Hardware?

Thanos Stratikopoulos

Friends of OpenJDK devroom
Sunday 5th February 2023

Write Once, Run Anywhere

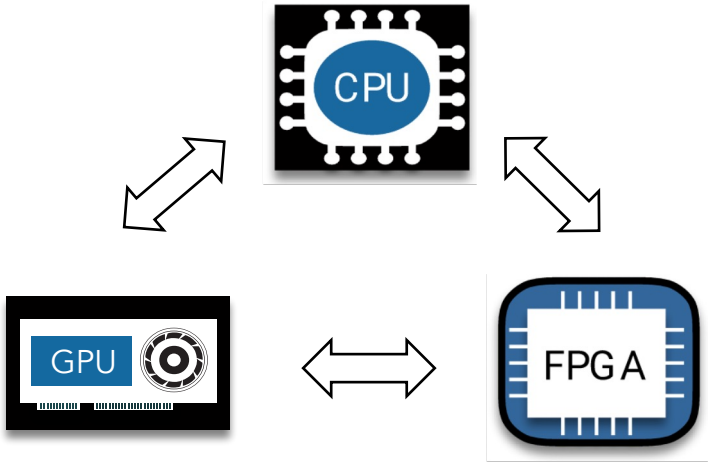


Abstraction

Portability (across different CPUs)

? Portability on heterogeneous hardware

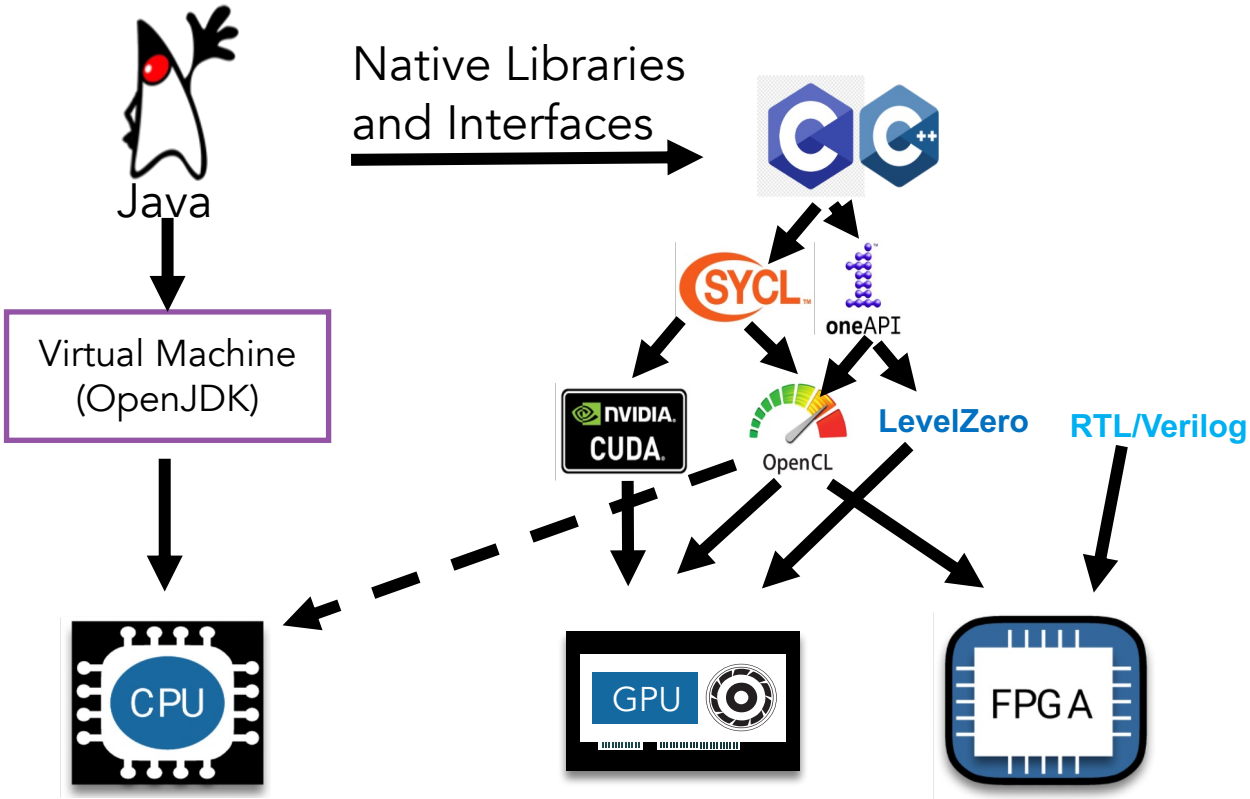
Hardware is evolving



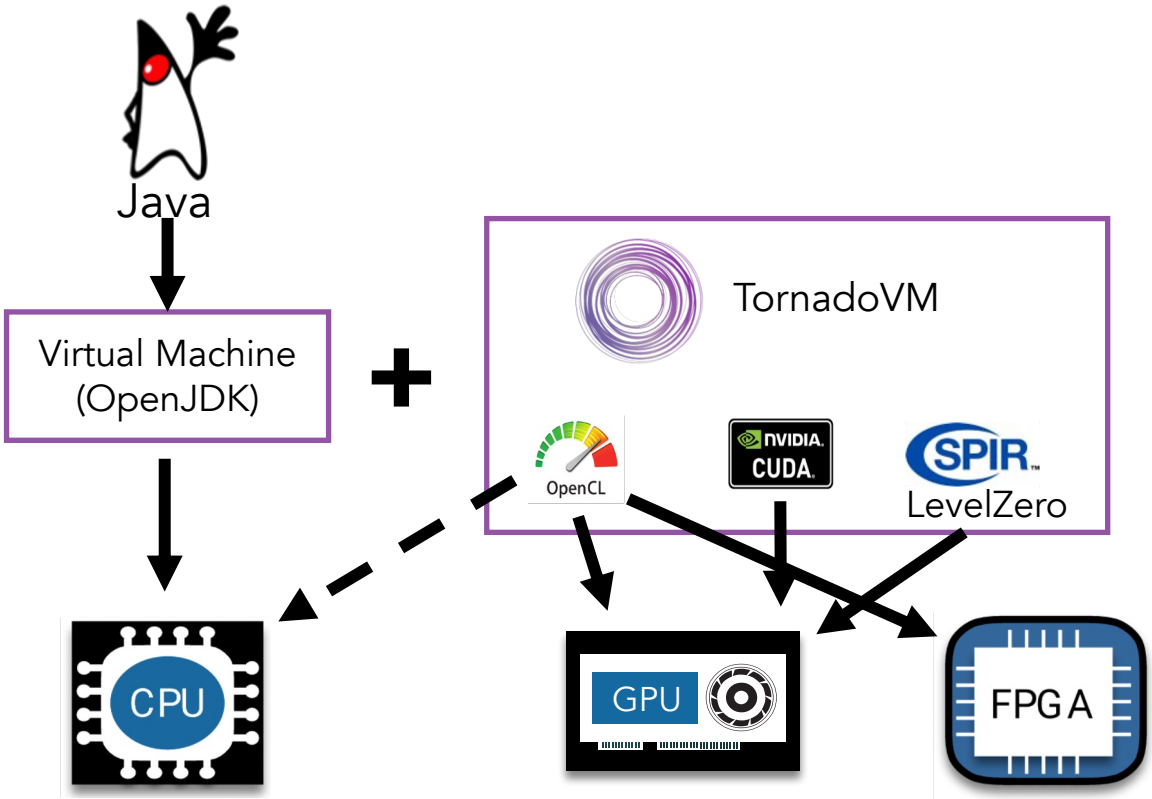
Pros: ☻
- Higher Performance
- Better Resource Utilization - Cost
- Energy Efficiency

Cons:☹
- Challenges in programming

Programming Heterogeneous Hardware



Programming Heterogeneous Hardware



https://tornadovm.org

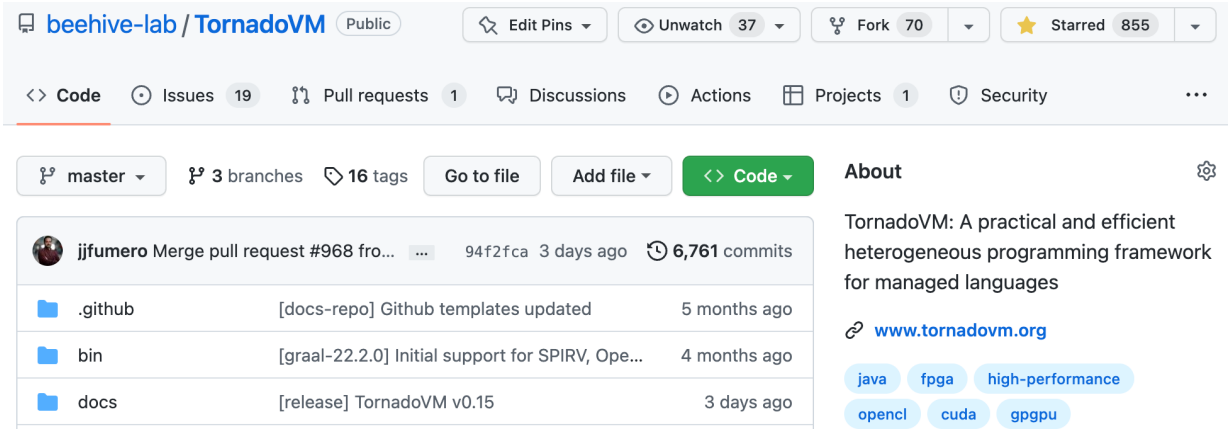
@tornadovm

A JVM plugin that accelerates Java methods on heterogeneous hardware!

Features:

- Lightweight API
- Platform agnostic
- Automatic code specialization

TornadoVM in GitHub & DockerHub!



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

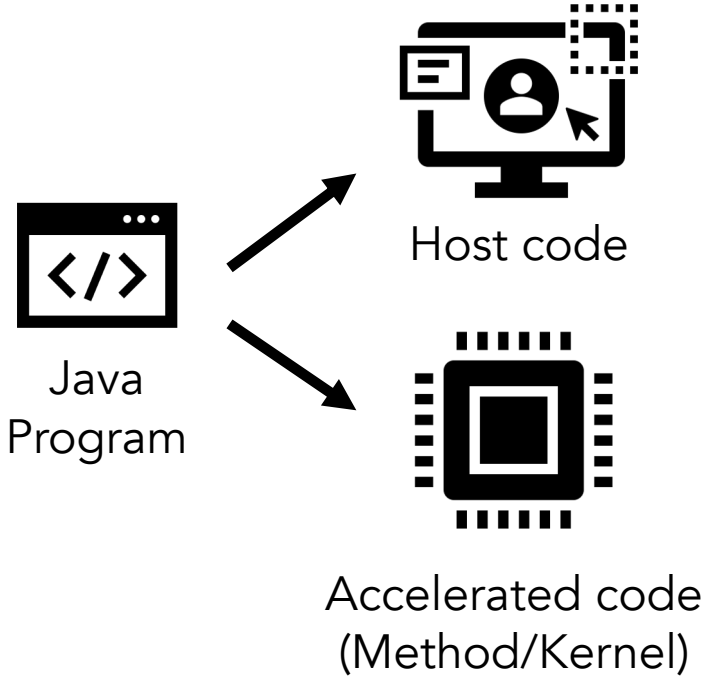

```
$ docker pull beehivelab/tornado-gpu

# And RUN !
$ ./run_nvidia.sh javac.py YouApp.java
$ ./run_nvidia.sh tornado YourApp
```

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

TornadoVM API

TornadoVM Programming Model



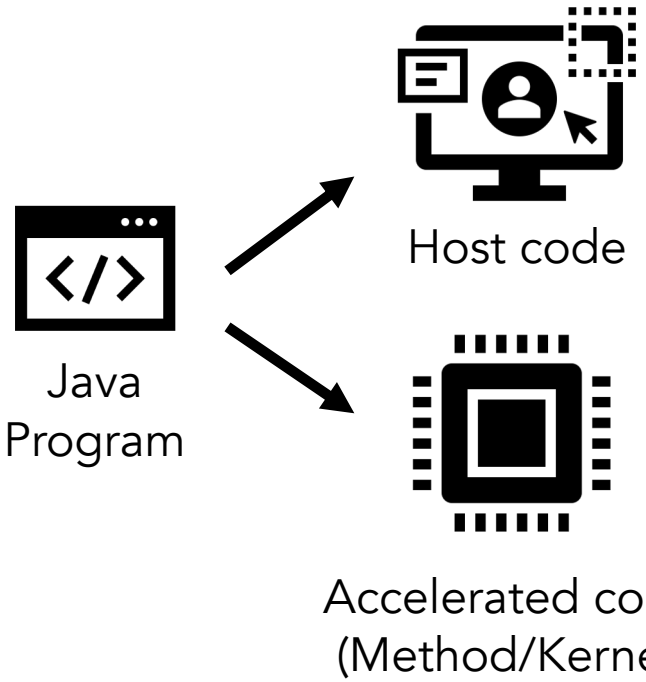
Execution Model

Trasfer Data to Device

Processing

Transfer Data to Host

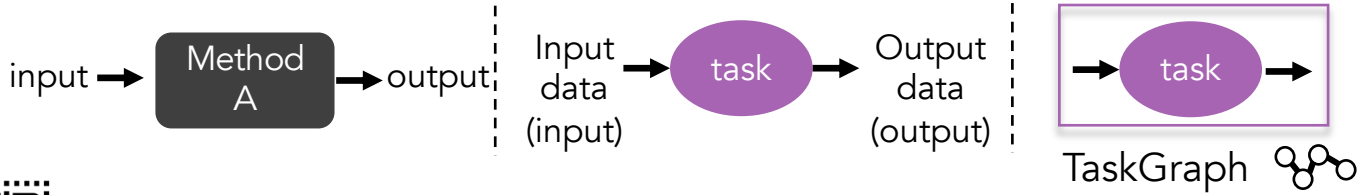
TornadoVM API

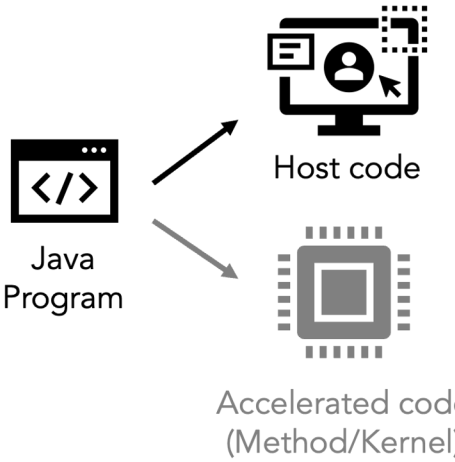


`TaskGraph`,
`TornadoExecutionPlan`

`@Parallel`,
`@Reduce` ,
`KernelContext`

TornadoVM TaskGraph (What to run?)



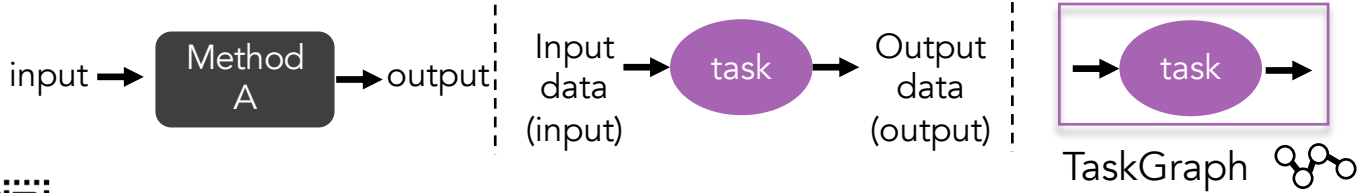


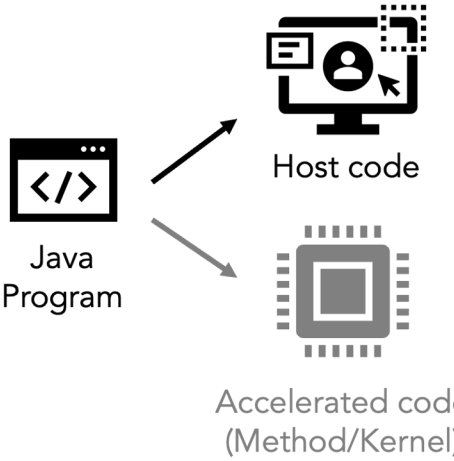
How often to transfer the data?
(first execution, always, last execution)

```
TaskGraph taskGraph = new TaskGraph("tg")
        .transferToDevice(DataTransferMode.FIRST_EXECUTION, input)
        .task("methodA", Class::MethodA, input, output, ...)
        .transferToHost(DataTransferMode.EVERY_EXECUTION, output);
```

The TaskGraph at this stage is **mutable**!

TornadoVM Immutable TaskGraph





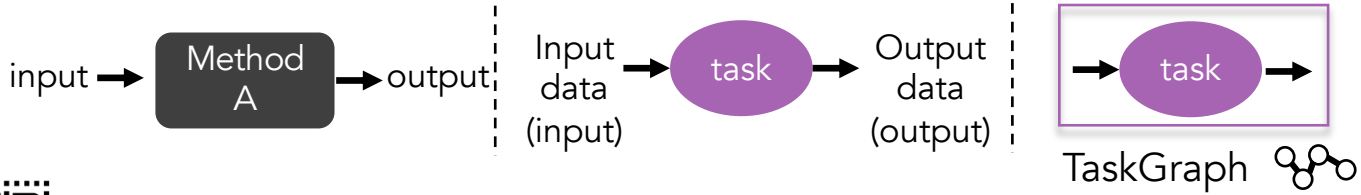
How to preserve the shape of a TaskGraph?


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

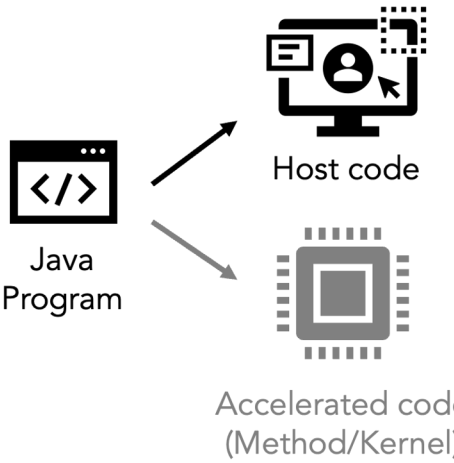
Mutable TaskGraph
- Can be modified

Immutable TaskGraph
- Cannot be modified

TornadoVM Execution Plan (How to run?)



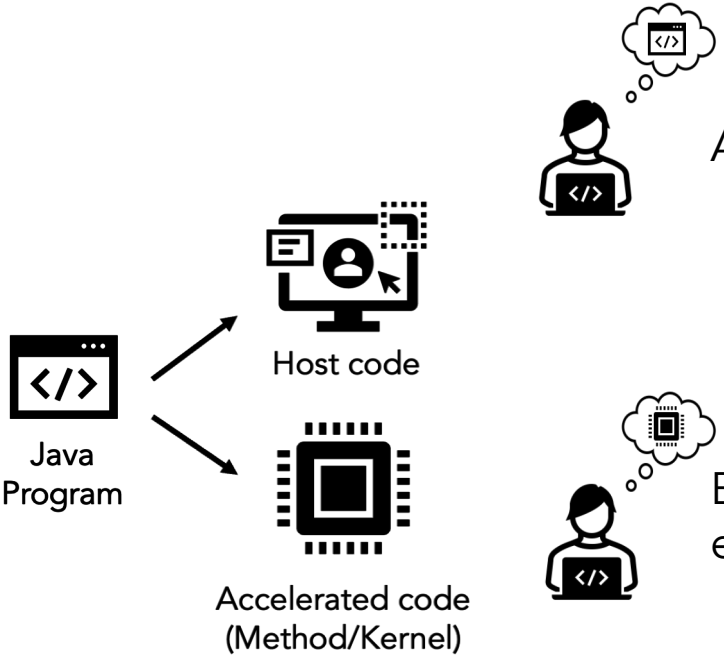
How to execute an **immutable** TaskGraph?



TornadoExecutionPlan

```
TornadoExecutionPlan executionPlan =
                    new TornadoExecutionPlan(immutableTaskGraph).build();
executionPlan.execute();
executionPlan.withDynamicReconfiguration(Policy.PERFORMANCE,
DRMode.PARALLEL);
```

TornadoVM - Expressing Parallelism



A) Loop Parallel API for Java programmers

Annotate for loops using *@Parallel*

B) Kernel API for programmers with OpenCL/CUDA expertise, or not (e.g. porting existing kernels)

Customize methods using the KernelContext object

For more info:
https://tornadovm.readthedocs.io/en/latest/programming.html

Example of Matrix Multiplication in Java

```java
private static void matrixMultiplication(final float[] A, final float[] B, final float[] C, final int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            float sum = 0.0f;
            for (int k = 0; k < size; k++) {
                sum += A[(i * size) + k] * B[(k * size) + j];
            }
            C[(i * size) + j] = sum;
        }
    }
}
```

Accelerated Code

```java
public static void main(String[] args){
    int size = 512;

    float[] matrixA = new float[size * size];
    float[] matrixB = new float[size * size];
    float[] matrixC = new float[size * size];
    float[] resultSeq = new float[size * size];

    Random r = new Random();
    IntStream.range(0, size * size).parallel().forEach(idx -> {
        matrixA[idx] = r.nextFloat();
        matrixB[idx] = r.nextFloat();
    });

    // Run matrixMultiplication on Java sequentially
    matrixMultiplication(matrixA, matrixB, resultSeq, size);
}
```

Host Code

Utilize the Loop Parallel API

```
private static void matrixMultiplication(final float[] A, final float[] B, final float[] C, final int size) {
    for (int i = 0; i < size; i++) {
        for (int j = 0; j < size; j++) {
            float sum = 0.0f;
            for (int k = 0; k < size; k++) {
                sum += A[(i * size) + k] * B[(k * size) + j];
            }
            C[(i * size) + j] = sum;
        }
    }
}
```

```
private static void matrixMultiplication(final float[] A, final float[] B, final float[] 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[(i * size) + k] * B[(k * size) + j];
            }
            C[(i * size) + j] = sum;
        }
    }
}
```

Utilize the Kernel API

```
private static void matrixMultiplication(final float[] A, final float[] B, final float[] 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[(i * size) + k] * B[(k * size) + j];
            }
            C[(i * size) + j] = sum;
        }
    }
}
```

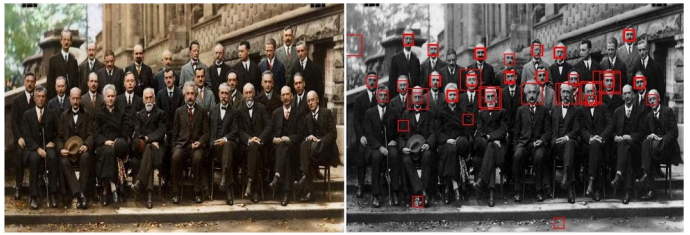
```
private static void matrixMultiplication(KernelContext context, final float[] A, final float[] B, final float[] C, final int size) {
    int globalRow = context.globalIdx;
    int globalCol = context.globalIdy;
    float sum = 0;

    for (int k = 0; k < size; k++) {
        sum += A[(k * size) + globalRow] * B[(globalCol * size) + k];
    }
    C[(globalCol * size) + globalRow] = sum;
}
```

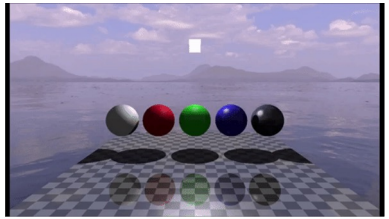
For more info, watch the FOSDEM'22 presentation:
https://www.youtube.com/watch?v=RD9W68rYS7I

Use Cases

COMPUTER VISION



FACE DETECTION



RAY TRACING

MACHINE LEARNING

https://www.tornadovm.org/use-cases

Acceleration as a Service…

Horizon 2020 ELEGANT

A software stack that unifies development for Big Data and IoT software ecosystems!

https://www.elegant-h2020.eu/

Acceleration as a Service for Java

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

```
    "deviceInfo": {
      "deviceName": "Nvidia GPU",
      "doubleFPSupport": true,
      "maxWorkItemSizes": [16, 1, 1],
      "deviceAddressBits": 64,
      "deviceType": "CL_DEVICE_TYPE_GPU",
      "deviceExtensions": "cl_khr_int64_base_atomics",
      "availableProcessors": 2048
    }
```

Source code, hw characteristics

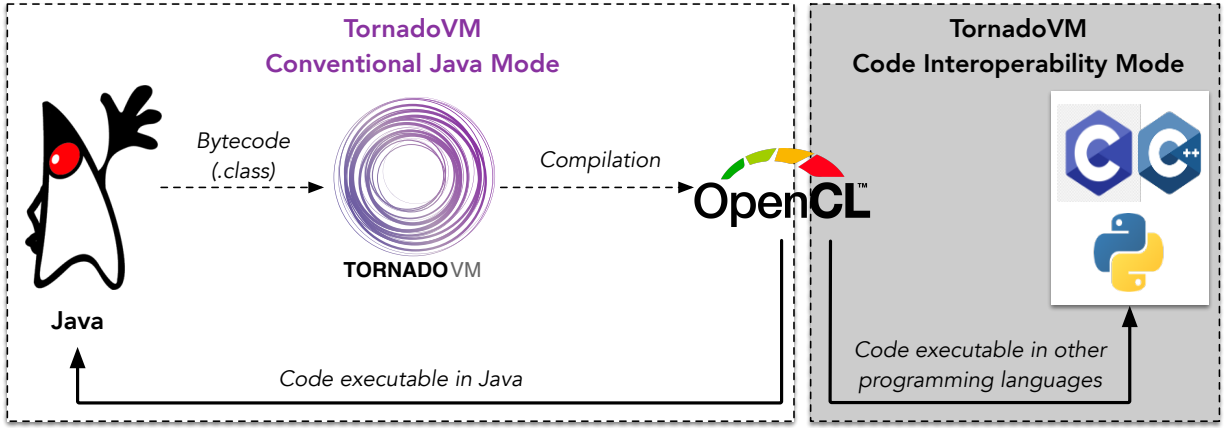
REST API

OpenCL kernel

```
__kernel void vectorAdd(__global uchar *a,
                        __global uchar *b,
                        __global uchar *c,
                        __private int size)
{
    // ...
    i_3  =  (ulong) size;
    for(;i_5 < i_3;) // Replacement of constant value
    {
        // BLOCK 2
        l_6  =  (long) i_5;
        l_7  =  l_6 << 2; ; // Header offset (24L) is skipped
        ul_8  =  ul_0 + l_7;
        i_9  =  *((__global int *) ul_8);
        ul_10  =  ul_1 + l_7;
        i_11  =  *((__global int *) ul_10);
        ul_12  =  ul_2 + l_7;
        i_13  =  i_9 + i_11;
        *((__global int *) ul_12)  =  i_13;
        i_14  =  get_global_size(0);
        i_15  =  i_14 + i_5;
        i_5  =  i_15;
    }  // B2
    // BLOCK 3
    return;
}  //  kernel
```

TornadoVM CIM Mode

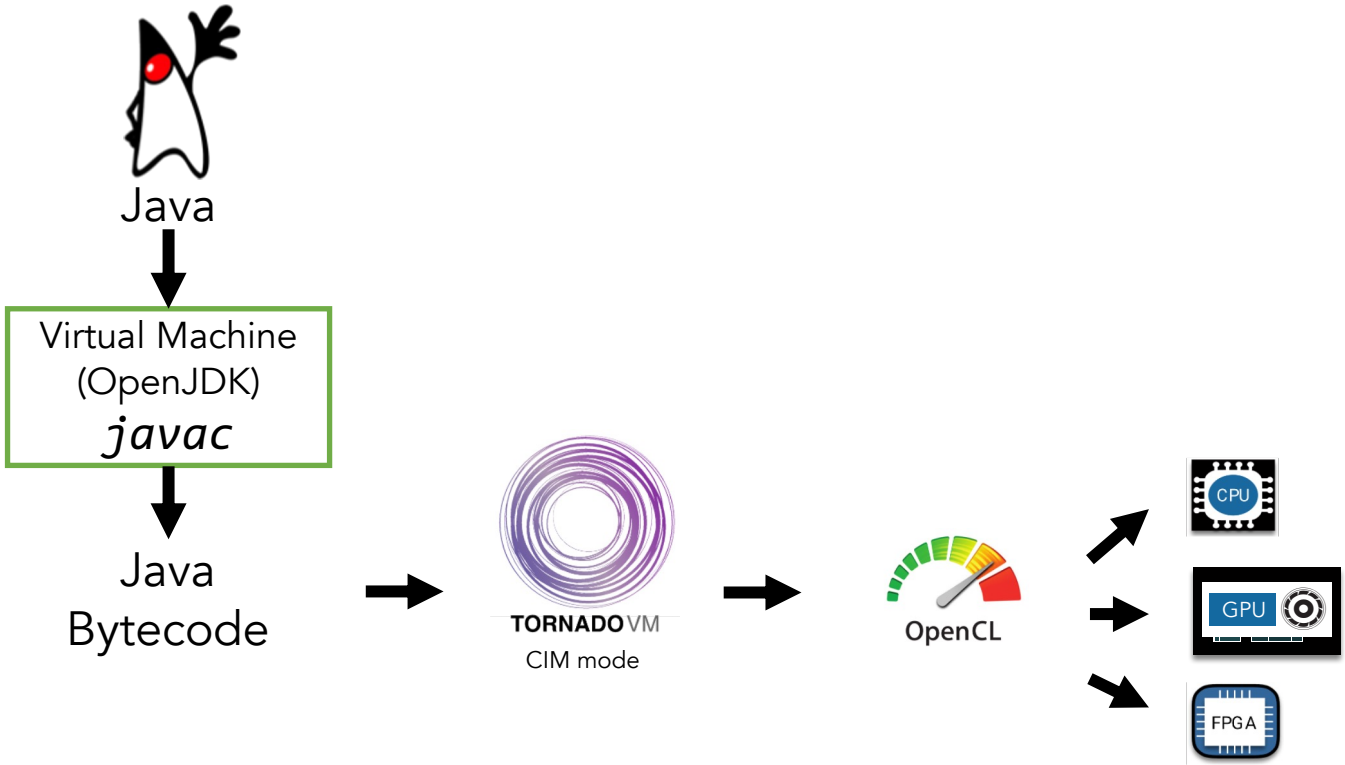
(Experimental – not upstream)



Java as prototyping language for parallel programming

Cross-language portability of the generated kernels

Write Once, Run Anywhere



Wrap Up

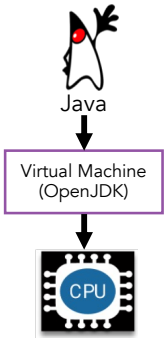
Can we help you?

 Glad to seek for synergies (acceleration of software).

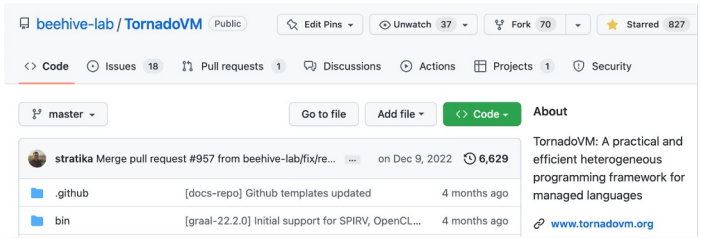
 https://tornadovm.org/contact/

Summary

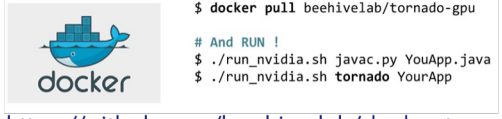
Write Once, Run Anywhere



✓Abstraction
✓Portability (across different CPUs)
? Portability on heterogeneous hardware



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

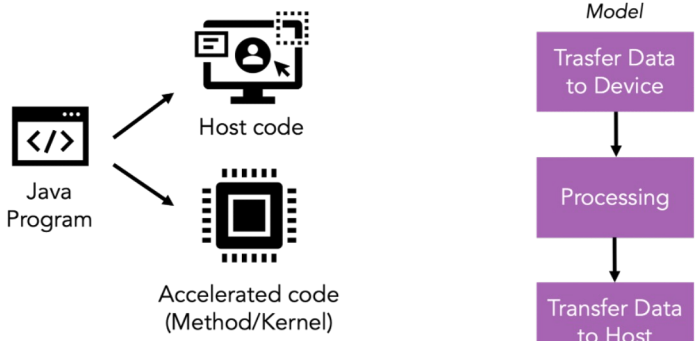


https://github.com/beehive-lab/docker-tornado

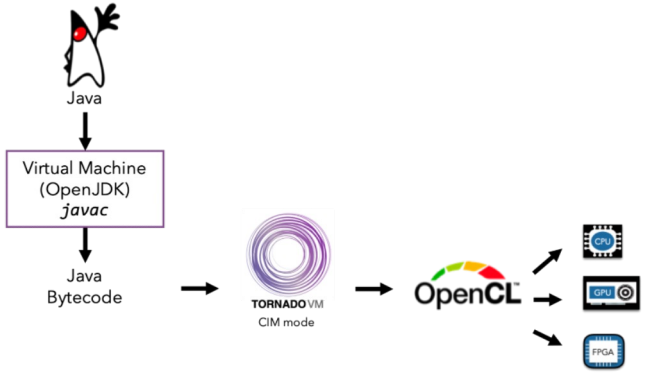
TORNADOVM

https://tornadovm.org
@tornadovm

TornadoVM Programming Model



Write Once, Run Anywhere



This work is partially supported by research grants from the
EU Horizon 2020 and EU Horizon Europe research and innovation programme,
UKRI, and Intel Corporation.

ELEGANT

encrypt

TANGO

AERO

INCODE

European Commission

UK Research and Innovation