TornadoVM: Hardware Acceleration for Java in Practice

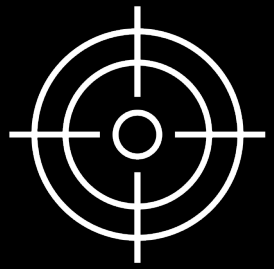
Thanos Stratikopoulos
athanasios.stratikopoulos@manchester.ac.uk

5 February 2021

Agenda

- Problem & Objective
- TornadoVM
- Factors that Impact Performance
- How to program using TornadoVM?
- Employment in existing Java code bases & IDEs
- Use cases
- Summary

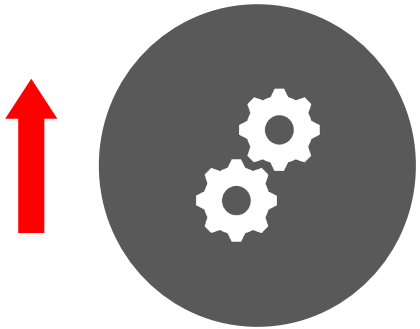
Problem



Key factors for hw acceleration in Software Industry

PERFORMANCE

DEVELOPMENT COST



OPERATIONAL COST

Hire experienced engineers to optimize code for dedicated hardware solutions

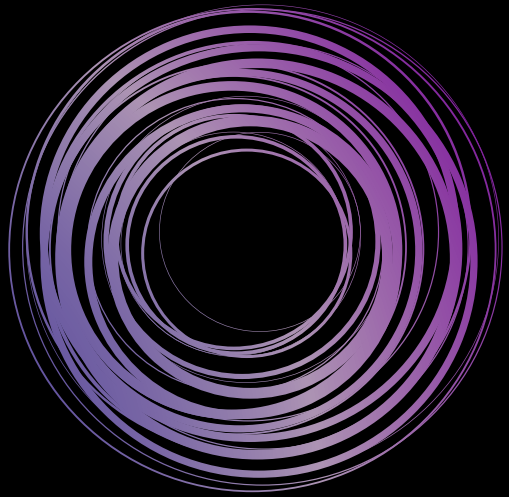
Objective

PERFORMANCE

DEVELOPMENT COST

OPERATIONAL COST

Increasing performance, while reducing costs!



TornadoVM

https://www.tornadovm.org/

Plugin to various JDKs

OpenJDK 8, 11, 17

GraalVM JDK8, JDK11, JDK 17

Amazon Corretto JDK 8, JDK 11,
Red Hat Mandrel JDK 11,
Microsoft JDK 11, JDK 17

Target Architectures

OpenCL
(Multi-core CPUs, GPUs, FPGAs)

PTX
(Nvidia GPUs)

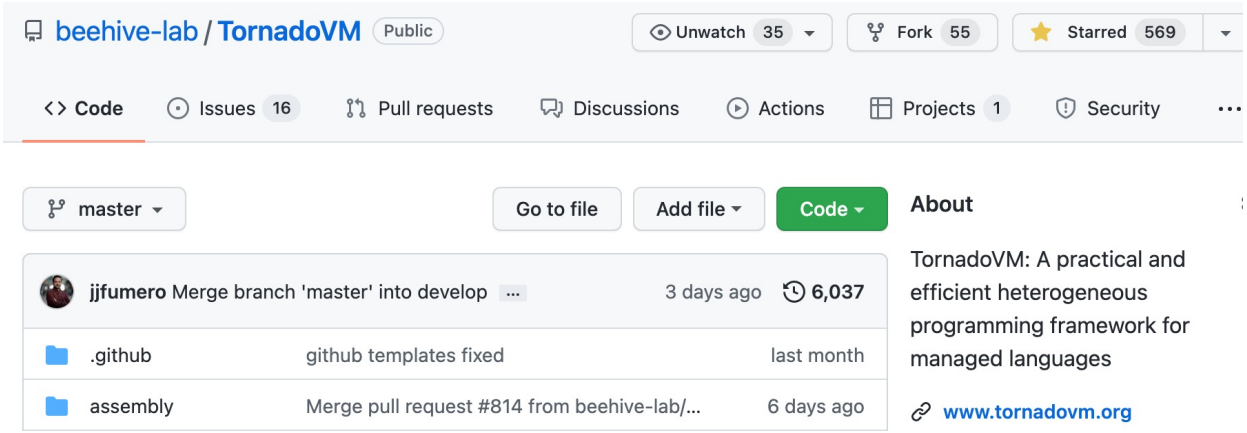
SPIR-V
(Intel Integrated GPUs, etc.)

Programming Languages

Java

Truffle Languages
(Python, R, Ruby, JavaScript, etc.)

TornadoVM available on GitHub & DockerHub

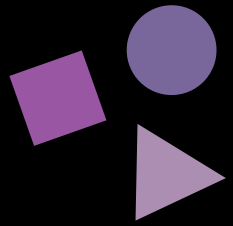


https://github.com/beehive-lab/TornadoVM

```
$ 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 Features



Easy to Install

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

```
thanos@thanoss-mbp tornadoVM % ./scripts/tornadoVMInstaller.sh
TornadoVM installer for Linux and OSx
Usage:
       --jdk8           : Install TornadoVM with OpenJDK 8
       --jdk11          : Install TornadoVM with OpenJDK 11
       --jdk17          : Install TornadoVM with OpenJDK 17
       --graal-jdk-11   : Install TornadoVM with GraalVM and JDK 11 (GraalVM 21.3.0)
       --graal-jdk-17   : Install TornadoVM with GraalVM and JDK 17 (GraalVM 21.3.0)
       --corretto-11    : Install TornadoVM with Corretto JDK 11
       --corretto-17    : Install TornadoVM with Corretto JDK 17
       --mandrel-11     : Install TornadoVM with Mandrel 21.3.0 (JDK 11)
       --mandrel-17     : Install TornadoVM with Mandrel 21.3.0 (JDK 17)
       --windows-jdk-11 : Install TornadoVM with Windows JDK 11
       --windows-jdk-17 : Install TornadoVM with Windows JDK 17
       --opencl         : Install TornadoVM and build the OpenCL backend
       --ptx            : Install TornadoVM and build the PTX backend
       --spirv          : Install TornadoVM and build the SPIR-V backend
       --help           : Print this help
```

Linux OS

mac OS
(experimental)

Windows OS
(experimental)

Transparent Acceleration

Multi-core CPUs (x86)

GPUs (Intel, Nvidia, AMD, Arm)

FPGAs (Intel, Xilinx)

Hardware-agnostic API	+	Enhanced Compiler Phases	=	Transparent Acceleration

Transparent Acceleration - Example

```
thanos-MacBook:tornadoVM thanos_str$ tornado --devices

Number of Tornado drivers: 1
Total number of OpenCL devices  : 2
Tornado device=0:0
        Apple -- Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz
                Global Memory Size: 16.0 GB
                Local Memory Size: 32.0 KB
                Workgroup Dimensions: 3
                Max WorkGroup Configuration: [1024, 1, 1]
                Device OpenCL C version: OpenCL C 1.2

Tornado device=0:1
        Apple -- Intel(R) Iris(TM) Graphics 6100
                Global Memory Size: 1.5 GB
                Local Memory Size: 64.0 KB
                Workgroup Dimensions: 3
                Max WorkGroup Configuration: [256, 256, 256]
                Device OpenCL C version: OpenCL C 1.2
```

```
$ tornado MatrixMultiplication2D
```

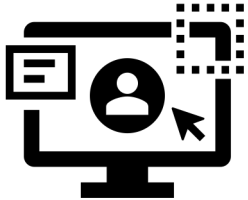
Same command to run on CPU/GPU/FPGA

Performance Impact Factors

Factors that can impact performance

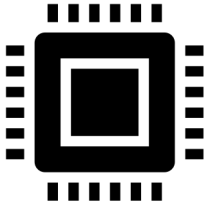
Algorithm

- Code pattern (parallelism)
- Data sizes
- Data types



Software Configuration

- Heap size, GC
- Warm-up



Hardware Configuration

- PCIe (for data transfers)
- Kernel (data locality, deployed threads)

How to determine if an application is suitable for acceleration?

Answer: Profiling 

Example (DFT)

Algorithm

- DFT (Inherent Parallelism)
- Data sizes (1 MB)

OpenJDK 17

- Heap size: 2 GB
- Warm-up: 10,000 executions

Nvidia 1050 GPU

- PCIe: Gen 3 (16x lanes)
- Kernel: 65,536 threads

async-profiler

4297x
(on 1050 GPU)

- [0] 99.99% 40,776 self: 0.00% 0 uk/ac/manchester/tornado/examples/dynamic/DFTJava.main
  - [1] 99.99% 40,776 self: 14.07% 5,738 uk/ac/manchester/tornado/examples/dynamic/DFTJava.computeDft
    + [2] 49.82% 20,316 self: 3.27% 1,334 uk/ac/manchester/tornado/api/collections/math/TornadoMath.floatCos
    + [2] 36.10% 14,722 self: 3.34% 1,364 uk/ac/manchester/tornado/api/collections/math/TornadoMath.floatSin
+ [0] 0.01% 3 self: 0.00% 0 start_thread
+ [0] 0.00% 1 self: 0.00% 0 java/lang/invoke/DirectMethodHandle$Holder.invokeStatic

TornadoVM Profiler

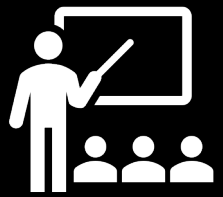
More info: https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/9_PROFILER.md


```
{
    "s0": {
        "TOTAL_KERNEL_TIME": "24437632",
        "TOTAL_DISPATCH_KERNEL_TIME": "5024",
        "TOTAL_DISPATCH_DATA_TRANSFERS_TIME": "60640",
        "COPY_IN_TIME": "44160",
        "TOTAL_TASK_SCHEDULE_TIME": "24886252",
        "COPY_OUT_TIME": "40992",
        "TOTAL_COPY_OUT_SIZE_BYTES": "524336",
        "TOTAL_COPY_IN_SIZE_BYTES": "1048672",
        "s0.t0": {
            "METHOD": "DFTDynamic.computeDft",
            "DEVICE_ID": "0:0",
            "DEVICE": "Quadro GP100",
            "TASK_KERNEL_TIME": "24437632"
        }
    }
}
```

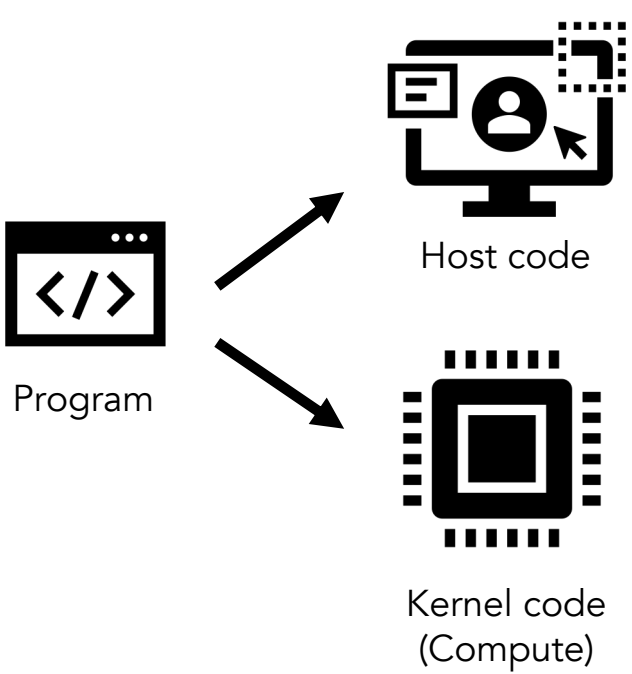

Breakdown	Percentage (%)
Kernel	98.20
Dispatching	0.26
CopyIn	0.18
CopyOut	0.16
Rest	1.20

```
$ tornado –enableProfiler console
–m tornado.examples/uk.ac.manchester.tornado.examples.dynamic.DFTDynamic 65536
gpu 10
```

Heterogeneous Programming Models



OpenCL/CUDA Programming Models



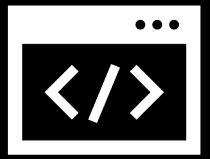
Execution Model

Move Data from Java to Device

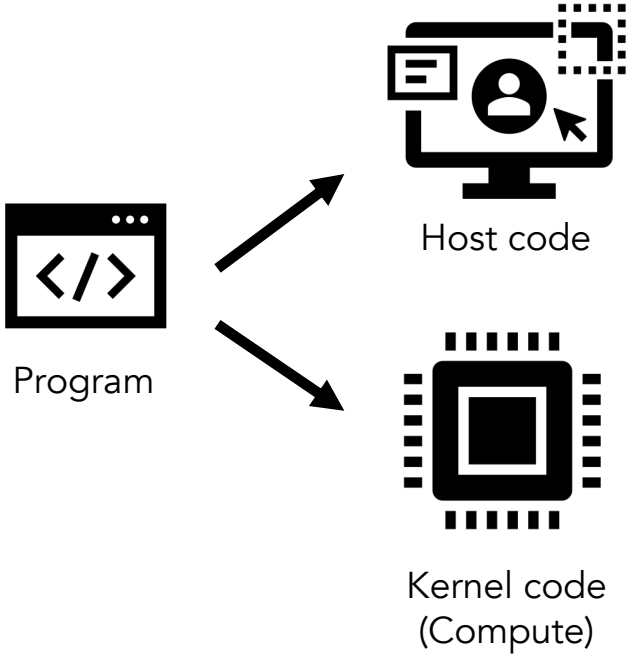
Parallel Compute

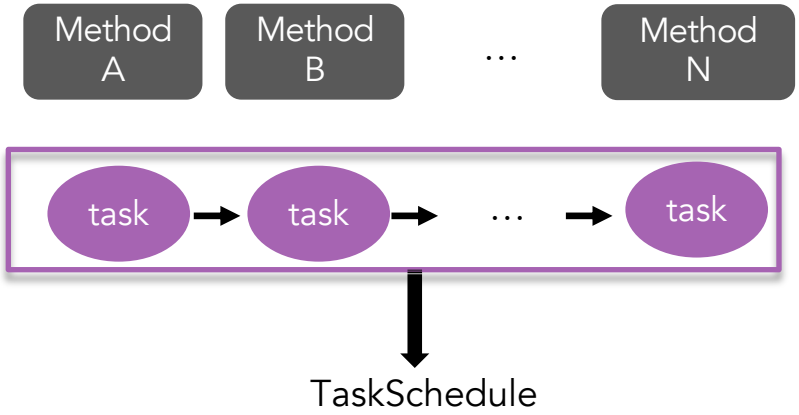
Move Data back to Java

How to program with TornadoVM?



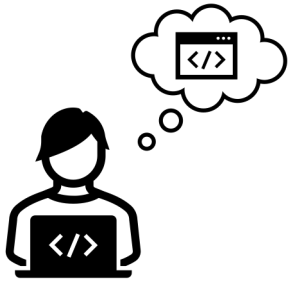
TornadoVM TaskSchedule





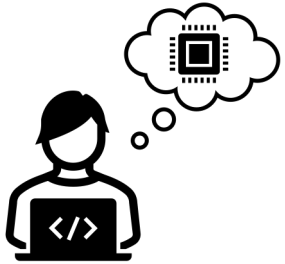
Custom pipelines

Two APIs



A) Loop Parallel API for Java programmers

→ Annotate methods with *@Parallel*



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

→ Customize methods with KernelContext

How to start? 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;
        }
    }
}
```

Compute Part

```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 Part

Java Host Part


```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 GPU
    TaskSchedule s0 = new TaskSchedule( name: "s0") //
        .streamIn(matrixA, matrixB) //
        .task( id: "t0", MatrixMultiplication1D::matrixMultiplication, matrixA, matrixB, matrixC, size) //
        .streamOut(matrixC);
    s0.execute();
}
```

```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);
}
```

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;
        }
    }
}
```

Kernel API for hardware programmers (1)

```
    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;
}
```

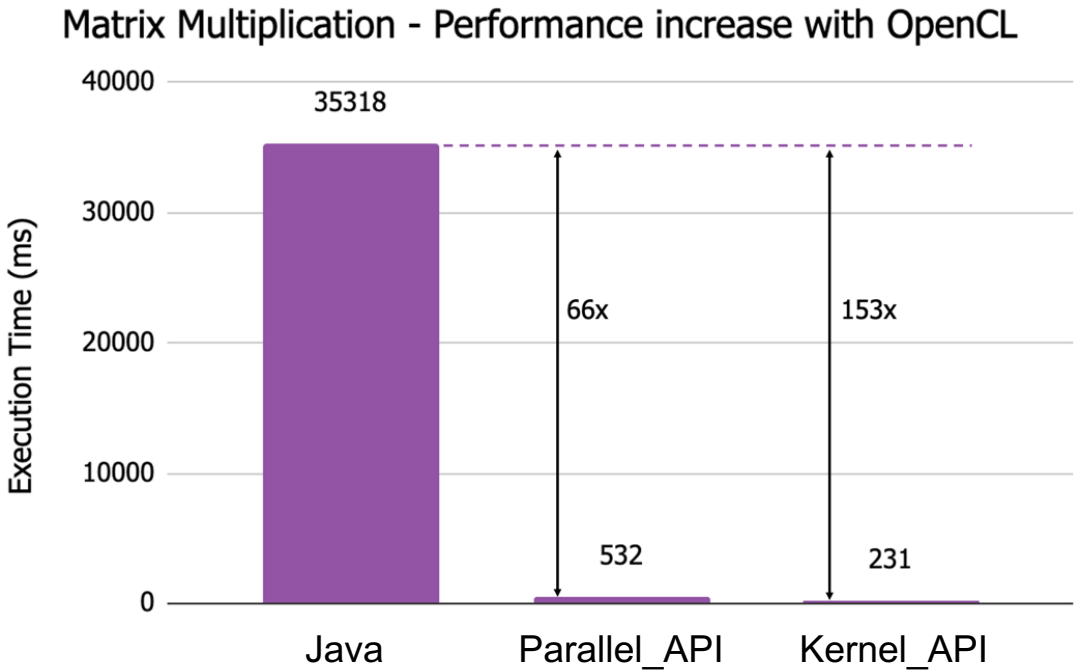
Kernel API for hardware programmers (2)

```
private static final int TS = 32;

public static void matrixMultiplication(KernelContext context, final float[] a, final float[] b,
                                        final float[] c, final int size) {

    int row = context.localIdx;
    int col = context.localIdy;
    int globalRow = TS * context.groupIdx + row;
    int globalCol = TS * context.groupIdy + col;

    float[] aSub = context.allocateFloatLocalArray( size: TS * TS);
    float[] bSub = context.allocateFloatLocalArray( size: TS * TS);

    float sum = 0;

    // Loop over all tiles
    int numTiles = size / TS;
    for (int t = 0; t < numTiles; t++) {

        // Load one tile of A and B into local memory
        int tiledRow = TS * t + row;
        int tiledCol = TS * t + col;
        aSub[col * TS + row] = a[tiledCol * size + globalRow];
        bSub[col * TS + row] = b[globalCol * size + tiledRow];

        // Synchronise to make sure the tile is loaded
        context.localBarrier();

        // Perform the computation for a single tile
        for (int k = 0; k < TS; k++) {
            sum += aSub[k * TS + row] * bSub[col * TS + k];
        }
        // Synchronise before loading the next tile
        context.localBarrier();
    }

    // Store the final result in C
    c[(globalCol * size) + globalRow] = sum;
}
```

OpenCL Kernel available:
https://cnugteren.github.io/tutorial/pages/page4.html

```
1. // Tiled and coalesced version
2. __kernel void myGEMM2(const int M, const int N, const int K,
3.                       const __global float* A,
4.                       const __global float* B,
5.                       __global float* C) {
6.
7.     // Thread identifiers
8.     const int row = get_local_id(0); // Local row ID (max: TS)
9.     const int col = get_local_id(1); // Local col ID (max: TS)
10.    const int globalRow = TS*get_group_id(0) + row; // Row ID of C (0..M)
11.    const int globalCol = TS*get_group_id(1) + col; // Col ID of C (0..N)
12.
13.    // Local memory to fit a tile of TS*TS elements of A and B
14.    __local float Asub[TS][TS];
15.    __local float Bsub[TS][TS];
16.
17.    // Initialise the accumulation register
18.    float acc = 0.0f;
19.
20.    // Loop over all tiles
21.    const int numTiles = K/TS;
22.    for (int t=0; t<numTiles; t++) {
23.
24.        // Load one tile of A and B into local memory
25.        const int tiledRow = TS*t + row;
26.        const int tiledCol = TS*t + col;
27.        Asub[col][row] = A[tiledCol*M + globalRow];
28.        Bsub[col][row] = B[globalCol*K + tiledRow];
29.
30.        // Synchronise to make sure the tile is loaded
31.        barrier(CLK_LOCAL_MEM_FENCE);
32.
33.        // Perform the computation for a single tile
34.        for (int k=0; k<TS; k++) {
35.            acc += Asub[k][row] * Bsub[col][k];
36.        }
37.
38.        // Synchronise before loading the next tile
39.        barrier(CLK_LOCAL_MEM_FENCE);
40.    }
41.
42.    // Store the final result in C
43.    C[globalCol*M + globalRow] = acc;
44. }
```

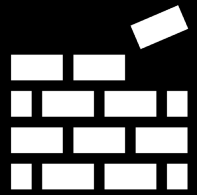
What about performance?



Nvidia 1050 GPU
Data Size: 2048x2048

From 35 seconds to 231 ms!

How TornadoVM can be used in Projects?



How to use TornadoVM?

In external projects

TornadoVM Sandboxing

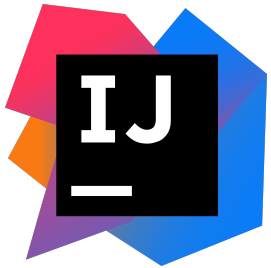
TornadoVM in External Projects

```
<repositories>
    <repository>
        <id>universityOfManchester-graal</id>
        <url>https://raw.githubusercontent.com/beehive-lab/tornado/maven-tornadovm</url>
    </repository>
</repositories>

<dependencies>
<dependency>
    <groupId>tornado</groupId>
    <artifactId>tornado-api</artifactId>
    <version>0.12</version>
</dependency>
<dependency>
    <groupId>tornado</groupId>
    <artifactId>tornado-matrices</artifactId>
    <version>0.12</version>
</dependency>
</dependencies>
```

Set this to your pom.xml file

TornadoVM Sandboxing

Integration with IDEs



https://github.com/beehive-lab/TornadoVM/blob/master/assembly/src/docs/3_INTELLIJ.md

$ python scripts/eclipseSetup.py

How to contribute?

- Look at GitHub issues tagged with good first issue
- Improve documentation
- New use-cases
- Bug fixes
- To contribute to the TornadoVM core, share your proposal as a Google Doc.

https://github.com/beehive-lab/TornadoVM/blob/master/CONTRIBUTING.md

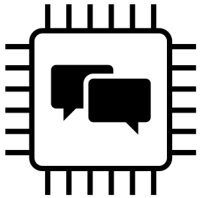
Use Cases

Existing Areas

Computer Vision

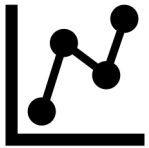
Face Detection

NLP



Machine Learning

Data Analytics & IoT



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

Our Team

Academic Staff

Christos Kotselidis

Research Staff

Juan Fumero

Thanos Stratikopoulos

Florin Blanaru

PhD Students

Maria Xekalaki

Alumni

James Clarkson

Foivos Zakkak

Benjamin Bell

Amad Aslam

Michail Papadimitriou

Gyorgy Rethy

Mihai-Christian Olteanu

Ian Vaughan

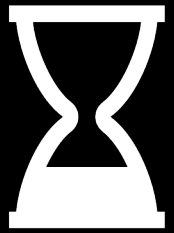
Ales Kubicek

Can we help you?

We are looking for collaborations (industrial & academics)!

https://www.tornadovm.org/contact-us

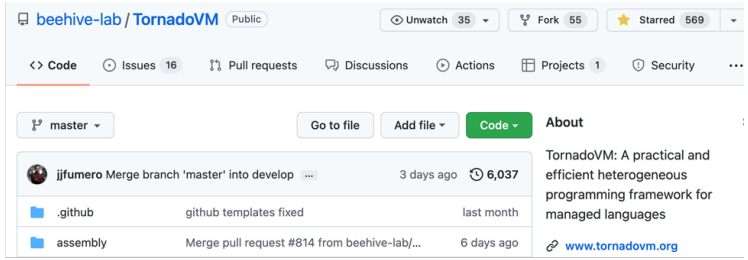
Wrap Up



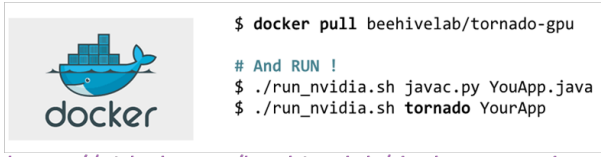
Summary

PERFORMANCE | DEVELOPMENT COST | OPERATIONAL COST

Increasing performance, while reducing costs!



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



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

A) Loop Parallel API for Java programmers

→ Annotate methods with *@Parallel*

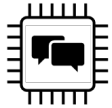
B) Kernel API for programmers with OpenCL/CUDA expertise, or not (e.g. porting existing kernels)

→ Customize methods with KernelContext

Computer Vision

Face Detection

NLP



Machine Learning

Data Analytics & IoT

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

This work is partially supported by the EU Horizon 2020 ELEGANT 957286 and Intel Corporation.

athanasios.stratikopoulos@manchester.ac.uk