Exploiting High-Performance Heterogeneous Hardware for Java Programs using Graal

James Clarkson\(^\pm\), Juan Fumero\(^*\), Michalis Papadimitriou\(^*\), Foivos S. Zakkak\(^*\), Christos Kotselidis\(^*\) and Mikel Luján\(^*\)

\(^\pm\)Dyson, \(^*\)The University of Manchester

ManLang’18, Linz (Austria), 12th September 2018
Outline

- Background

- Tornado
  - Tornado-API
  - Tornado Runtime
  - Tornado JIT Compiler

- Performance Results

- Conclusions
Context of this project

Started as the PhD thesis of **James Clarkson**: *Compiler and Runtime Support for Heterogeneous Programming*

James Clarkson, Christos Kotselidis, Gavin Brown, and Mikel Luján.
*Boosting Java Performance using GPGPUs.*
*In Proceedings of the 30th International Conference on Architecture of Computing Systems*

Christos Kotselidis, James Clarkson, Andrey Rodchenko, Andy Nisbet, John Mawer, and Mikel Luján.
*Heterogeneous Managed Runtime Systems: A Computer Vision Case Study*
*ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE ’17)*

Partially funded by the EPSRC AnyScale grant EP/LO00725/1
Currently part of the EU H2020 E2Data Project

"End-to-end solution for heterogeneous Big Data deployments that fully exploits and advances the state-of-the-art in infrastructure"  https://e2data.eu/

European Union’s Horizon H2020 research and innovation programme under grant agreement No 780622
1. Background
Current Heterogeneous Computing Landscape
Current Heterogeneous Computing Landscape
Current Heterogeneous Computing Landscape
Current Virtual Machines

- C/C++
- Fortran
- ...

Virtual Machine

- Java
- JS
- Python
- Ruby

Hardware

- FPGA
- GPU
- NDP
- x86 Arch
Our Solution: VM + Heterogeneous Runtime
2. Tornado: A Practical Heterogeneous Programming Framework
Tornado

- A Java based Heterogeneous Programming Framework
- It exposes a task-based parallel programming API
- It contains an OpenCL JIT Compiler and a Runtime for running on heterogeneous devices
- Modular system currently using:
  - OpenJDK/Graal
  - OpenCL
- It currently runs on CPUs, GPUs and FPGAs*
Tornado Overview

Tornado API
- Task Schedule
- Task API

Tornado Runtime
- Optimiser
- DFG
- OCL-Execution Engine

Tornado JIT Compiler
- JIT
- Heap
Tornado API: @Parallel

"It's a developer provided annotation that instructs the JIT compiler that it is OK for each iteration to be executed independently."

It does not specify or imply:

- iterations should be executed in parallel;
- the parallelization scheme to be used
Task Schedules

"A task schedule describes how to co-ordinate the execution of tasks across heterogeneous hardware."

- Composability
- Sequential consistency
- Task-based parallelism
- Automatic and optimised data movement
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];
    }
}

// execute array addition on an accelerator
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];

new TaskSchedule("s0")
    .task("t0", ArrayAddInt::add, a, b, c)
    .streamOut(c)
    .execute();
Tornado API: enabling task-based parallelism

Auto-parallelization directive

```java
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];
    }
}
```

// execute array addition on an accelerator
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];

new TaskSchedule("s0")
    .task("t0", ArrayAddInt::add, a, b, c)
    .streamOut(c)
    .execute();
Tornado API: enabling task-based parallelism

Auto-parallelization directive

```java
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];
    }
}
```

// execute array addition on an accelerator
int[] a = new int[100];
int[] b = new int[100];
int[] c = new int[100];

new TaskSchedule("s0")
  .task("t0", ArrayAddInt::add, a, b, c)
  .streamOut(c)
  .execute();
```

Task-based execution model
Task Schedules: example

```java
public class Ex {

    public static void multiply
        (Double4[] a, Double4[] b, Double4[] c) {
        // code here
    }

    public static void add
        (Double4[] a, Double4[] b, Double4[] c) {
        // code here
    }
}
```
Task Schedules: example

```java
TaskSchedule schedule = new TaskSchedule("s0")
    .streamIn(a, b) // copy a & b to device
    .task("t0", Ex::multiply, a, b, c) // multiply task
    .task("t1", Ex::add, c, b, d) // add task
    .streamOut(d); // return d to host

while (true) {
    schedule.execute();
}
```
Task Schedules: example

```
while (true) {
    task(multiply(a, b, c));
    task(add(c, b, d));
}
```
Task Schedules: example

```java
while (true) {
    schedule.execute();
}
```

Diagram:
- a
- b
- Ex1.mult(a, b, c)
- Ex1.add(c, b, d)
- d
3. Tornado Runtime
Tornado: WorkFlow

Describes a data-flow graph

Each node is a task

Tornado Runtime

Tornado Compiler

Task Schedule

new TaskSchedule("s0")
  .add(Ex1::add, a, b, c)
  .streamOut(c)
  .execute();

Task

void add(int[] a, int[] b, int[] c) {
    for(int i=0; i<c.length; i++) {
        c[i] = a[i] + b[i];
    }
}

Tornado API

APIreachable

code reachability analysis

data dependency analysis

Graph Optimizer

- task placement
- data-flow optimization
- inserts low-level tasks

Sketcher

- Tornado API
- code reachability analysis
- data dependency analysis

HIR Cache

Code Generator

- compiles cached sketches
- parallelization
- device specific built-ins

Task Executor

- maps tasks onto driver API
- triggers JIT compilation
- triggers data-movements

OpenCL Runtime

clEnqueueWriteBuffer()
clEnqueueNDRangeKernel()
clEnqueueReadBuffer()

OpenCL C

__kernel void foo(…)
{
    …
}

Optimize Task Schedule

Code Cache

Execute Task Schedule

Puggable Driver

Driver API

Task Execution

Runtime Optimizations

Source Task Schedule

Serialized Task Schedule

Tornado Runtime

Tornado Compiler

new TaskSchedule("s0")
  .add(Ex1::add, a, b, c)
  .streamOut(c)
  .execute();

void add(int[] a, int[] b, int[] c) {
    for(int i=0; i<c.length; i++) {
        c[i] = a[i] + b[i];
    }
}
Data parallelism - Task specialisation

E.g., currently we have two parallel schemes: course-grain and fine-grain

```c
1 // Loop for GPUs
2 int idx = get_global_id(0);
3 int size = get_global_size(0);
4 for (int i = idx; i < c.length;
5     i += size) {
6     // computation
7     c[i] = a[i] + b[i];
8 }
```

```c
1 // Loop for CPUs
2 int id = get_global_id(0);
3 int size = get_global_size(0);
4 int block_size = (size +
5     inputSize - 1) / size;
6 int start = id * block_size;
7 int end = min(start + bs, c.length);
8 for (int i = start; i < end; i++) {
9     // computation
10    c[i] = a[i] + b[i];
11 }
```
Memory Management

- Each heterogeneous device has a managed heap
- Enables objects to persist on devices
- Currently we duplicate objects which reside in the JVM heap
- No object creation on devices
4. Tornado JIT Compiler
Tornado JIT Compiler

Input: Java Method

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

Output: OpenCL Kernel

```c
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;) {
        i_4 = (long) i_3;
        i_5 = i_4 << 2;
        i_6 = i_5 + 24;
        ul_7 = ul_0 + i_6;
        i_8 = *((global int *) ul_7);
        ul_9 = ul_1 + i_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;
    }
}
```
5. Case study
Case study

**Kinect Fusion**: it is a complex computer vision application that is able to re-construct a 3D model from RGB-D camera in real time.
Why KFusion?

- Not a normal Java application
- Complex multi-kernel pipeline
  - Sustained the execution of 540-1620 kernels per second.
  - SLA of 30 FPS
- Representative of cutting edge robotics/computer vision applications
- Want to deploy across many platform and accelerator combinations
What did we get with Tornado?

Running on NVIDIA Tesla, up to 150 fps
And compared to native code?

Tornado is 28% slower than the best OpenCL native code.
6. Announcement & Conclusions
Tornado is now Open Source!

$ git clone https://github.com/bee hive-lab/tornado
$ make
$ tornado YourApplication

- We also have a poster tomorrow, come along!
- If you are interested, we can also show you demos on GPUs and FPGAs!
Takeaway

- We have presented Tornado
- We have shown runtime code generation for OpenCL
- We have shown a case study for computer vision
- It is open-source, give a try!

We are looking forward for your feedback!
Thank you very much for your attention

This work is partially supported by the EPSRC grants PAMELA EP/K008730/1 and AnyScale Apps EP/LO00725/1, and the EU Horizon 2020 E2Data 780245.

Juan Fumero <juan.fumero@manchester.ac.uk>
Compilation times

<table>
<thead>
<tr>
<th>Time (seconds)</th>
<th>OpenCL</th>
<th>Graal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.05</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.20</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **AMD A10 7850K**
- **Intel i7 4850HQ**
- **Intel E5 2620**
- **AMD Radeon R7**
- **Intel Iris Pro 5200**
- **NVIDIA GT 750M**
- **NVIDIA Tesla K20m**
OpenCL Device Driver: Just In Time Compiler

OpenCL JIT Compiler and Runtime

```c
__kernel void add(...) {
    // BLOCK 0
    u3_0 = (ulong) _frame[0];
    u1_1 = (ulong) _frame[7];
    u1_2 = (ulong) _frame[8];
    i_3 = get_global_id(0);
    // BLOCK 1 MERGES [0 2]
    l_4 = i_3;
    for(l_4 < 32) {
        // BLOCK 2
        l_5 = (long) l_4;
        l_6 = l_5 << 4;
        l_7 = l_6 = 24;
        u1_8 = u3_0 + l_7;
        i_9 = *((__global int *) u1_8);
        u1_10 = u1_1 + l_7;
        i_11 = *((__global int *) u1_10);
        u1_12 = u1_2 + l_7;
        l_13 = i_9 + i_11;
        *((__global int *) u1_12) = l_13;
        l_14 = get_global_size(0);
        l_15 = l_14 + l_4;
        l_4 = l_15;
    }
    // BLOCK 3
    return;
}
```