Marawacc: A Framework for Heterogeneous Computing in Java

Juan Fumero, Michel Steuwer, Christophe Dubach

The University of Edinburgh

UK Many-Core Developer Conference 2016
Motivation

Heterogeneous Hardware
Motivation

Heterogeneous Hardware

Programming Interface

- NVIDIA CUDA
- OpenCL
- SYCL
Motivation
Motivation
Marawacc: our approach

Three levels of abstraction
Example: Saxpy in Java

```java
float[] v1 = new float[size];
float[] v2 = new float[size];
float[] result = new float[size];

for (int i = 0; i < size; i++) {
    result[i] = alpha * v1[i] + v2[i];
}
```
Example: Saxpy in Java

```java
Float[] v1 = new Float[size];
Float[] v2 = new Float[size];

ArrayFunc<Tuple2<Float, Float>, Float> f;
f = new MapFunction<>((t -> alpha * t._1() + t._2()));

Float[] result = f.zip(v1, v2).apply();
```
Runtime Code Generation

Java User Application

Parallel Skeletons

Standard Java Byte-Codes

Graal VM

OpenCL Code Generation

Heterogeneous Runtime
## Runtime Code Generation

### Workflow

1. **Type inference**
2. **IR generation**
3. **Optimizations**
4. **Kernel generation**

Java source

```
Map.apply(f)
```

Java bytecode

```
... 10: aload_2
     11: iload_3
     12: aload_0
     13: getfield
     16: aaload
     18: invokeinterface#apply
     23: aastore
     24: iinc
     27: iload_3
     ...  
```

Graal VM

```
1. Type inference
```

```
void kernel (
    global float* input,
    global float* output) {
    ...
    ...
}
```

OpenCL Kernel

```
CGF + Dataflow (Graal IR)
```

11 / 23
MapFunction\(<\text{Integer}, \text{Double}\> (x \rightarrow x \times 2.0)\)

inline double lambda0 ( int p0 ) {
    double cast_1 = (double) p0;
    double result_2 = cast_1 * 2.0;
    return result_2;
}
Marawacc: Runtime Management

Java User Application

Parallel Skeletons

Standard Java Byte-Codes

Graal VM

OpenCL Code Generation

Heterogeneous Runtime

Marshalling
Copy to Device
Execution
Copy to Host
Un-Marshall
Where the time is spent?

Black-scholes benchmark.

\[ \text{Float[]} \rightarrow \text{Tuple2} < \text{Float}, \text{Float} > [] \]

- Un/marshal data takes up to 90% of the time
- Computation step should be dominant

This is not acceptable. Can we do better?
Custom Array Type: PArray

With this layout, un/marshal operations are not necessary
Float [] v1 = new Float [size];
Double [] v2 = new Double [size];

ArrayFunc<Tuple2<Float, Double>, Double> f;
f = new MapFunction<>()(t -> alpha * t._1() + t._2());

Float [] result = f.zip(v1, v2).apply();
Saxpy with our Custom PArrays

```java
Float[] v1 = new Float[size];
Double[] v2 = new Double[size];
PArray input = new PArray(v1, v2);

ArrayFunc<Tuple2<Float, Double>, Double> f;
f = new MapFunction<>((t -> alpha * t._1() + t._2()));
PArray<Double> output = f.apply(input);
```
Results
OpenCL GPU Execution
AMD R9 and NVIDIA GeForce GTX Titan

![Graph showing speedup vs. Java sequential for different benchmarks and marshalling methods.]
Comparison with OpenCL C++

AMD R9 and NVIDIA GeForce GTX Titan
Present

- We have presented Marawacc framework for programming GPUs from Java
- Custom array type to reduce overheads when transforming the data
- Runtime system to run heterogeneous applications within Java

Future

- Code generation for multiple devices
- Runtime scheduling (Where is the best place to run the code?)
Thanks so much for your attention

Juan Fumero
<juan.fumero@ed.ac.uk>
OpenCL code generated

```c
double lambda0(float p0) {
    double cast_1 = (double) p0;
    double result_2 = cast_1 * 2.0;
    return result_2;
}

kernel void lambdaComputationKernel(
    global float * p0,
    global int * p0_index_data,
    global double * p1,
    global int * p1_index_data) {
    int p0_dim_1 = 0; int p1_dim_1 = 0;
    int gs = get_global_size(0);
    int loop_1 = get_global_id(0);
    for (; ; loop_1 += gs) {
        int p0_len_dim_1 = p0_index_data[p0_dim_1];
        bool cond_2 = loop_1 < p0_len_dim_1;
        if (cond_2) {
            float auxVar0 = p0[loop_1];
            double res = lambda0(auxVar0);
            p1[p1_index_data[p1_dim_1 + 1] + loop_1] = res;
        } else {
            break;
        }
    }
}
```