Towards Practical Heterogeneous Virtual Machines

James Clarkson, Juan Fumero, Michalis Papadimitriou, Maria Xekalaki, Christos Kotselidis

The University of Manchester

MoreVMs 2018, Nice (France), 9th April 2018
Outline

- Context
- Motivation
- Challenges
- Proposal
Context of this project

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


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. Motivation
Current Heterogeneous Computing Landscape

- FPGA
- GPU
- NDP
- x86 Arch
Current Heterogeneous Computing Landscape

- **VHDL**
- **OpenCL**
- **DSL?**

- **FPGA**
- **GPU**
- **NDP**
- **x86 Arch**
Current Heterogeneous Computing Landscape
Current Virtual Machines
Proposal: Heterogeneous Virtual Machines

Java  JS  Python  Ruby  R  ...

Heterogeneous Virtual Machine

FPGA  GPU  NDP  x86 Arch

Hardware
Why heterogeneous programming is challenging?
Let’s see an example in OpenCL with the NVIDIA Kepler GPU architecture.
NVIDIA Kepler Architecture

Figure from NVIDIA Docs.
Example: Reduction in Java

```java
1  public static void reduce(float[] input, float[] result) {
2      result[0] = 0;
3      for (int i = 0; i < input.length; i++) {
4          result[0] += input[i];
5          }
6  }
```
Example of a GPU Program: Reduction

```c
// Reduce version taken from: dournac.org
__kernel void reduce(__global float* input,
    __global float* partialSums,
    __local float* localSums) {

    int idx = get_global_id(0);
    uint localIdx = get_local_id(0);
    uint group_size = get_local_size(0);
    localSums[localIdx] = input[idx];
    for (uint stride = group_size / 2; stride > 0; stride /=2) {
        barrier(CLK_LOCAL_MEM_FENCE);
        if (localIdx < stride) {
            localSums[localIdx] += localSums[localIdx + stride];
        }
    }

    if (localIdx == 0) {
        partialSums[get_group_id(0)] = localSums[0];
    }
}
```

Execution on NVIDIA GTX 1050 is 4.5x faster than C-OpenMP with 8 threads
Heterogeneous Programming Challenges

Developers using managed programming languages must learn new programming models and hardware details to efficiently exploit the heterogeneous hardware.

We propose Heterogeneous VMs to abstract away all this complexity

However, this is not a trivial task...
2. Challenges
1. Programmability of heterogeneous devices from High-Level Languages

- As easy as mainstream programming, but efficiently and transparently exploit the heterogeneous architecture
- Respect the high-level semantics
- Constraints:
  - Dynamic memory allocation on devices is not allowed
  - Hardware exceptions are not supported (division by zero!?)
2. Transparency

- Use the accelerator, if possible, transparently to the final user
- Transparent memory management
  - Data Buffer
  - Data transfer and allocation between host and devices
  - Heterogeneous GC?
3. Adaptability

- Selecting the best combination of heterogeneous devices (e.g. GPUs vs FPGAS) is challenging
  - Global and local work dimensions on GPUs using OpenCL
  - Parameters for tuning FPGA code: loop unroll factor, number of CUs, degree of vectorization, etc.
4. Device Portability

- Re-compilation for the target device reacting to runtime changes
  - E.g. Fault-tolerance in Big Data deployments

- Use of de-optimisation and re-optimisation

- Dynamic device selection:
  - GPU1 -> GPU2
  - GPU -> FPGA
  - CPU -> GPU
5. Performance Portability

- Recompilation is not enough to maintain performance from one device to another
- The runtime and the VM should explore different parameters to achieve the expected performance
3. Our Proposal: Addressing the challenges in the JVM
We propose

- **A VM in a VM** to run heterogeneous Java programs
- **Four main components:**
  - API
  - Runtime
  - Virtual Machine
  - Device drivers
- **This is work in progress**
Proposed API for Heterogeneous Programming

```java
public class ComputeReduction {
    public void reduce(float[] input, @Reduce float[] output) {
        float result[0] = 0.0f;
        for (@Parallel int i = 0; i < input.length; i++) {
            result[0] += input[i];
        }
    }

    public static void main(String[] args) {
        ...
        TaskSchedule s0 = new TaskSchedule();
        s0.task(ComputeReduction::reduce, in, out)
            .execute();
    }
}
```
Proposed API for Heterogeneous Programming

```java
public class Compute {
    public static void t1(in, out) {
        for (int i ...) {
        }
    }

    public static void t2(in2, out2) {
        for (int i ...) {
        }
    }
}

public static void main(String[] args) {
    // Data initialization
    ...
    TaskSchedule s0 = new TaskSchedule();
    s0.task(Compute::t1, in, inter)
        .task(Compute::t2, inter, out);
    .execute();
}
```
Proposed Heterogeneous Runtime which:

1. Obtains tasks of a task schedule
2. Builds a Data Flow Graph
3. Optimizes data dependencies
4. Generates new bytecode for heterogeneous execution
Proposed Heterogeneous Virtual Machine

1. Executes the generated bytecode
2. JIT compiles and executes the input tasks to OpenCL via Graal extensions for heterogeneous compilation
OpenCL Device Driver: Just In Time Compiler

OpenCL JIT Compiler and Runtime

```c
__kernel void add(...) {
    // BLOCK 0
    u1_0 = (ulong)_frame[6];
    u1_1 = (ulong)_frame[7];
    u1_2 = (ulong)_frame[8];
    i_3 = get_global_id(0);
    // BLOCK 1 MERGES [0 2]
    i_4 = i_3;
    for(;i_4 < 32;) {
        // BLOCK 2
        i_5 = (long)i_4;
        i_6 = (long)i_4 + 2;
        i_7 = (long)i_8 + 2;
        u1_8 = u1_0 + i_7;
        s_9 = (*((__global int *)u1_8);
        u1_10 = u1_1 + i_7;
        s_11 = (*((__global int *)u1_10);
        u1_12 = u1_2 + i_7;
        i_13 = i_9 + i_11;
        *((__global int *)u1_12) = i_13;
        i_14 = get_global_size(0);
        i_15 = i_14 + i_4;
        i_4 = i_15;
    }
    // BLOCK 3
    return;
}
```
Conclusions

- Why heterogeneous computing is hard
- Main challenges of enabling heterogeneous VMs
- Proposal for addressing with those challenges (work in progress)

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/L000725/1, and the EU Horizon 2020 E2Data 780245.

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