Virtual Machine and Bytecode for Optimization on Heterogeneous Systems

Kerry A. Seitz, Jr. and Mark C. Lewis
Trinity University, San Antonio, TX
Acknowledgments

- Trinity University Department of Computer Science

- Trinity University Mach Fellowship Program
Overview

- Introduction
- Bytecode Design
- Implementation
- Conclusion
Heterogeneous Computing

- Use all hardware components available
  - CPU, GPU, FPGA, DSP

- Increase Performance

- Hardware independent
Heterogeneous Computing Challenges

- Different programming models
- Different combinations of devices
- Little development tool support
  - OpenCL, CUDA
- Developer training
Motivations

- Make heterogeneous computing easier
- Runtime code optimizations
- Platform independence
Annotation

- Provide optimization hints to VM
- Can be disregarded until optimization
- Backwards compatibility

- Two formats
  - Within instructions
  - Annotation file
Annotation Examples

- Immutability
- Escape thread/stack frame
- Generics/Type Erasure
- Specify Device
Popular Virtual Machines

- Java – stack-based
- Dalvik – register-based (16 registers)
- Both have issues
Virtual Machine Design

- Variables
  - $2^{16}$ per stack frame
  - Type associated with each
  - Annotations

- Class Files
  - Header modeled after Dalvik
  - Annotations
Instruction Format

<table>
<thead>
<tr>
<th>instruction</th>
<th>var1</th>
<th>var2</th>
<th>var3</th>
<th>var4</th>
<th>var5</th>
<th>var6</th>
<th>annotation</th>
</tr>
</thead>
</table>

- Instruction format
- Var 1
- Var 2
- Var 3
- Var 4
- Var 5
- Var 6
- Annotation
### Instruction Format

<table>
<thead>
<tr>
<th>instruction</th>
<th>var1</th>
<th>var2</th>
<th>var3</th>
<th>var4</th>
<th>var5</th>
<th>var6</th>
<th>annotation</th>
</tr>
</thead>
</table>

Instantiates variable `destVar` with the value from `sourceVar`

| init-var | destVar | type | sourceVar | | | | annotation |

Instantiates variable `destVar` with the value of `literal`

| init-var-lit | destVar | type | literal (64 bits) | | | | annotation |
# Instruction Format

<table>
<thead>
<tr>
<th>instruction</th>
<th>var1</th>
<th>var2</th>
<th>var3</th>
<th>var4</th>
<th>var5</th>
<th>var6</th>
<th>annotation</th>
</tr>
</thead>
</table>

Instantiates variable `destVar` with the value from `sourceVar`

```
init-var | destVar | type | sourceVar | | | | annotation |
```

Instantiates variable `destVar` with the value of `literal`

```
init-var-lit | destVar | type | literal (64 bits) | | | | annotation |
```

Binary operators on two variables

<table>
<thead>
<tr>
<th>op-type</th>
<th>destVar</th>
<th>sourceVar1</th>
<th>sourceVar2</th>
</tr>
</thead>
<tbody>
<tr>
<td>add</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sub</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mul</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>div</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>rem</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>and</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>or</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>xor</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>annotation</th>
</tr>
</thead>
</table>
## Instruction Format

<table>
<thead>
<tr>
<th>instruction</th>
<th>var1</th>
<th>var2</th>
<th>var3</th>
<th>var4</th>
<th>var5</th>
<th>var6</th>
<th>annotation</th>
</tr>
</thead>
</table>

Instantiates variable `destVar` with the value from `sourceVar`  

```
init-var  destVar  type  sourceVar
```

Instantiates variable `destVar` with the value of `literal`  

```
init-var-lit  destVar  type  literal (64 bits)
```

Binary operators on two variables

```
op-type  destVar  sourceVar1  sourceVar2
add      
sub      
mul      
div      
rem      
and      
or      
xor
```

```
annotation
```

Accesses information for the annotation type `annotation` at offset `literal` in the annotation file  

```
annot  literal (64 bits)
```

---

| Introduction | Bytecode Design | Implementation | Conclusion |
Programming Language

Scala
- High-level
- Object oriented/functional
- Compiles to JVM

OpenCL
- Lower-level
- Heterogeneous
- Platform independent (with correct drivers)
GPU Execution

- Uses OpenCL (through JavaCL)
- Compiles bytecode instructions into OpenCL kernel
- Parallelizes over an array of data
Summary

- Heterogeneous computing is hard, but increasingly essential

- Designed bytecode to aid in heterogeneous computing
  - Optimization hints in annotations
  - Platform independent

- Implementation in Scala with OpenCL
Future research

- Design more annotations
- Implement more annotations
- Develop lower level interpreter/JIT compiler
- Explore program analysis techniques
Questions?