Memory Access Management with LLVM and OpenCL

Mohammed Yassine Jazouani
jazouani@gmail.com
Université Grenoble Alpes, Grenoble
Supervised by:
Ylies Falcone, Brice Videau
and Jean-François Mehaut

LLVM - June, 16th 2015

Summary

1. Introduction
2. Low Level Virtual Machine: Compiler Infrastructure
3. Open Computing Language: Open Standard For Parallel Programming
4. Memory Management Done By LLVM On OpenCL
5. Conclusion

Summary

1 Introduction

2 Low Level Virtual Machine: Compiler Infrastructure

3 Open Computing Language: Open Standard For Parallel Programming

4 Memory Management Done By LLVM On OpenCL

5 Conclusion

Context

- Computing devices opt for parallel programming
- Shared memory management problem
- Tool capable to give enough information concerning the memory to the developer to avoid these problems
- Existing tools:
  - Implemented for LLVM: AddressSanitizer, MemorySanitizer, ThreadSanitizer
  - Used on simulated environments: Symbolic OpenCL, OCLgrind

Problem

- Compile time: syntax, typechecking and compiler errors
- Runtime: only division by zero, deferencing a null pointer or running out of memory errors

Main idea:

- Go from an OpenCL program
- Construct LLVM tools and passes to get as much memory information as possible during compile time
- Provide them to the developer

Summary

1 Introduction

2 Low Level Virtual Machine: Compiler Infrastructure

3 Open Computing Language: Open Standard For Parallel Programming

4 Memory Management Done By LLVM On OpenCL

5 Conclusion

Definition and purpose

- Framework with many tools, implemented in C++
- Open Source and providing low level code representation in Single Static Assignment
- Compilation with LLVM: clang -c -emit-llvm const.c -o const.bc

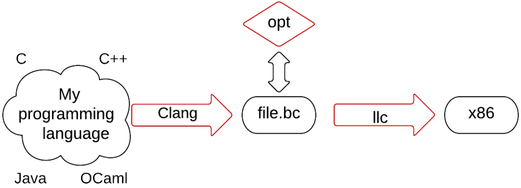

Figure: Global view of LLVM use

LLVM Pass

- Collection of libraries ready-to-use allowing code analyses and optimizations. Each analyse or optimization is called a pass.
- Example of pass: Memory allocation, common subexpression elimination ...

Remark

A pass can be run on a complete code or only on a portion. This leads to the distinction of various types of passes. In our case, we will focus on ModulePass (ran on the whole code) and FunctionPass (ran in functions)

Summary

1. Introduction
2. Low Level Virtual Machine: Compiler Infrastructure
3. Open Computing Language: Open Standard For Parallel Programming
4. Memory Management Done By LLVM On OpenCL
5. Conclusion

Definition

- Framework dedicated to writting C programs executed across multiple heterogeneous platforms
- OpenCL programs are compiled at runtime
- Platforms are composed of many compute devices
- Compute device execute programs: kernels
- One kernel can be executed in different compute devices
- One kernel can have multiple memory areas: buffers
- 1 compute device = many processing elements

Hierarchy

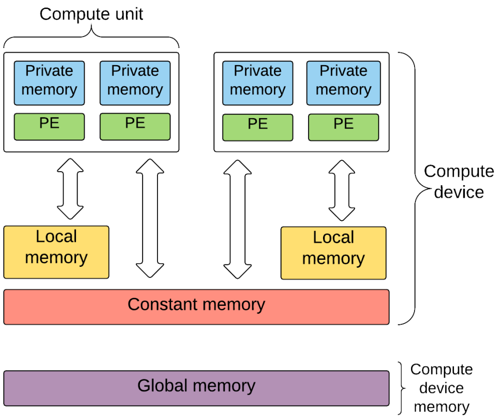

Figure: Memory hierarchy in OpenCL

From Runtime to Compile Time

- Information about kernel buffers needed (size, access right...)
- Known by OpenCL during runtime: communication
- Link between runtime and compile time

Summary

1 Introduction

2 Low Level Virtual Machine: Compiler Infrastructure

3 Open Computing Language: Open Standard For Parallel Programming

4 Memory Management Done By LLVM On OpenCL

5 Conclusion

Organization

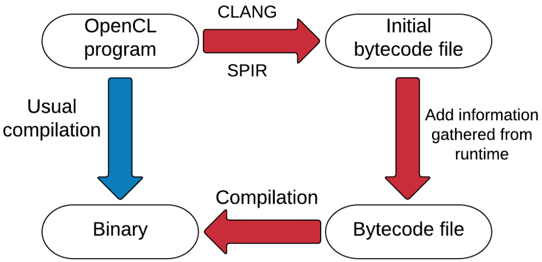

Figure: Steps of our work

Information of kernels needed. Two options:

- Force the user to insert information
- LLVM pass changing the function signatures at compile time

Adding arguments

Function signature before LLVM pass :

```
define cc76 void @addition(<2 x float> %alpha,
float addrspace(1)* nocapture %x,
float addrspace(1)* nocapture %y) nounwind
{/*Body of the function*/}
```

Function signature after LLVM pass :

```
define cc76 void @addition(<2 x float> %alpha,
float addrspace(1)* nocapture %x,
float addrspace(1)* nocapture %y,
int %size_x, int %size_y) nounwind
{/*Body of the function*/}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)
    *, float addrspace(1)*)* @addition, metadata !1,
    metadata !2, metadata !3, metadata !4, metadata !5}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)*, float addrspace(1)*)* @addition, metadata !1, metadata !2, metadata !3, metadata !4, metadata !5}

!1 = metadata !{metadata !"kernel_arg_addr_space", i32 0, i32 1, i32 1}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)*, float addrspace(1)*)* @addition, metadata !1, metadata !2, metadata !3, metadata !4, metadata !5}

!1 = metadata !{metadata !"kernel_arg_addr_space", i32 0, i32 1, i32 1}

!2 = metadata !{metadata !"kernel_arg_access_qual", metadata !"none", metadata !"none", metadata !"none"}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)
    *, float addrspace(1)*)* @addition, metadata !1,
    metadata !2, metadata !3, metadata !4, metadata !5}

!1 = metadata !{metadata !"kernel_arg_addr_space", i32
    0, i32 1, i32 1}

!2 = metadata !{metadata !"kernel_arg_access_qual",
    metadata !"none", metadata !"none", metadata !"none
    "}

!3 = metadata !{metadata !"kernel_arg_type", metadata
    !"float2", metadata !"float*", metadata !"float*"}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)
    *, float addrspace(1)*)* @addition, metadata !1,
    metadata !2, metadata !3, metadata !4, metadata !5}

!1 = metadata !{metadata !"kernel_arg_addr_space", i32
     0, i32 1, i32 1}

!2 = metadata !{metadata !"kernel_arg_access_qual",
    metadata !"none", metadata !"none", metadata !"none
    "}

!3 = metadata !{metadata !"kernel_arg_type", metadata
    !"float2", metadata !"float*", metadata !"float*"}

!4 = metadata !{metadata !"kernel_arg_type_qual",
    metadata !"", metadata !"const", metadata !""}
```

Complementary Information: Metadata

```
!opencl.kernels = !{!0} /*Metadata before LLVM pass*/
!0 = metadata !{void (<2 x float>, float addrspace(1)*, float addrspace(1)*)* @addition, metadata !1, metadata !2, metadata !3, metadata !4, metadata !5}

!1 = metadata !{metadata !"kernel_arg_addr_space", i32 0, i32 1, i32 1}

!2 = metadata !{metadata !"kernel_arg_access_qual", metadata !"none", metadata !"none", metadata !"none"}

!3 = metadata !{metadata !"kernel_arg_type", metadata !"float2", metadata !"float*", metadata !"float*"}

!4 = metadata !{metadata !"kernel_arg_type_qual", metadata !"", metadata !"const", metadata !""}

!5 = metadata !{metadata !"kernel_arg_base_type", metadata !"float2", metadata !"float*", metadata !"float*"}
```

Complementary Information: Metadata

Metada before LLVM pass:

```
!3 = metadata !{metadata !"kernel_arg_type",
     metadata !"float2", metadata !"float*",
   metadata !"float*"}
```

Metadata after LLVM pass:

```
!3 = metadata !{metadata !"kernel_arg_type",
   metadata !"float2", metadata !"float*",
   metadata !"float*", metadata !"int",
   metadata !"int"}
```

Aliasing

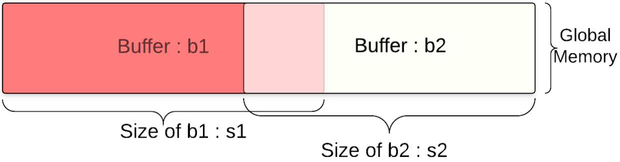

Figure: Example of aliasing

Out Of Bound

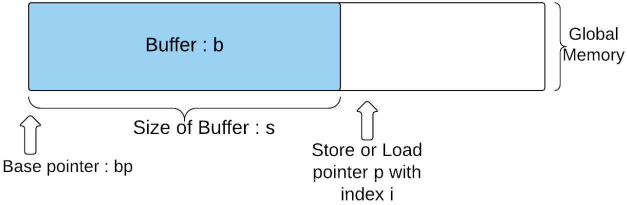

Figure: Example of out-of-bound memory access

Summary

1 Introduction

2 Low Level Virtual Machine: Compiler Infrastructure

3 Open Computing Language: Open Standard For Parallel Programming

4 Memory Management Done By LLVM On OpenCL

5 Conclusion

Conclusion

Future work:

- Function call using buffers as arguments inside kernel functions
- Access flag testing
- Concurrent memory access

Application:

- Created to overcome memory management problem
- Next big step: Detecting Direct Memory Access races

End

Thank you for your attention !