AMD OpenCL Compiler

Utilizing LLVM to create a

cross-platform heterogeneous compiler

tool-chain

Micah Villmow

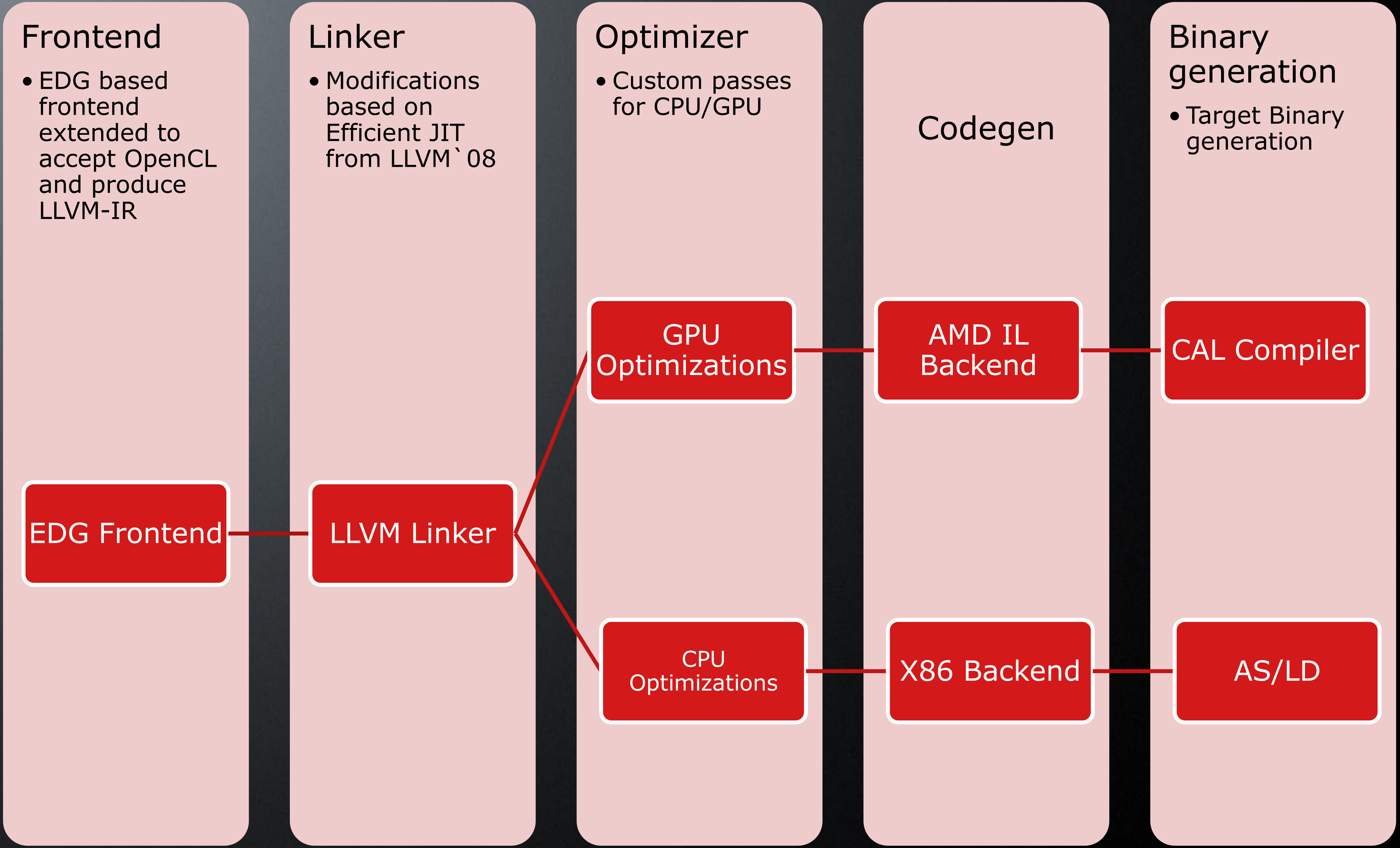
AMD

Summary

- Compiler Tool Chain
- Target Backends
- CPU Compiler Path
- GPU Compiler Path
- Ideas

Compiler Toolchain

AMD Accelerated Parallel Processing TECHNOLOGY


Frontend
• EDG based frontend extended to accept OpenCL and produce LLVM-IR
Linker
• Modifications based on Efficient JIT from LLVM`08
Optimizer
• Custom passes for CPU/GPU
Codegen
Binary generation
• Target Binary generation
EDG Frontend
LLVM Linker
GPU Optimizations
CPU Optimizations
AMD IL Backend
X86 Backend
CAL Compiler
AS/LD

AMD The future is fusion

Summary

- Compiler Tool Chain
- **Target Backends**
- CPU Compiler Path
- GPU Compiler Path
- Ideas

Target Backends

- The CPU backend targets SSE2+ devices
  - LLVM x86 backend
  - Minor changes/bug fixes to work with OpenCL
  - More information in Twin Peaks, Gummaraju, et al. PACT 2010
- The GPU backend targets AMD IL
  - IL spec in stream SDK
  - Unique constraints that don't exist most other languages

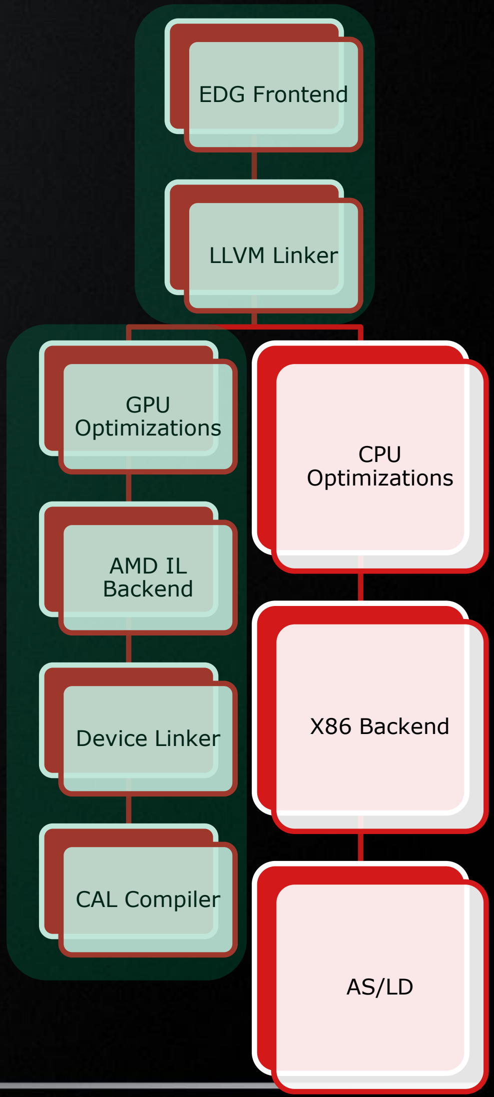
Summary

- Compiler Tool Chain
- Target Backends
- **CPU Compiler Path**
- GPU Compiler Path
- Ideas

AMD Accelerated Parallel Processing TECHNOLOGY

CPU Compiler Path

- Optimizer has mostly standard passes
- Backend expands mem* ops
- Custom win32 ulong->float conversion
- Custom data layout alignments
- Disable MMX generation
- MC Binary Emitter



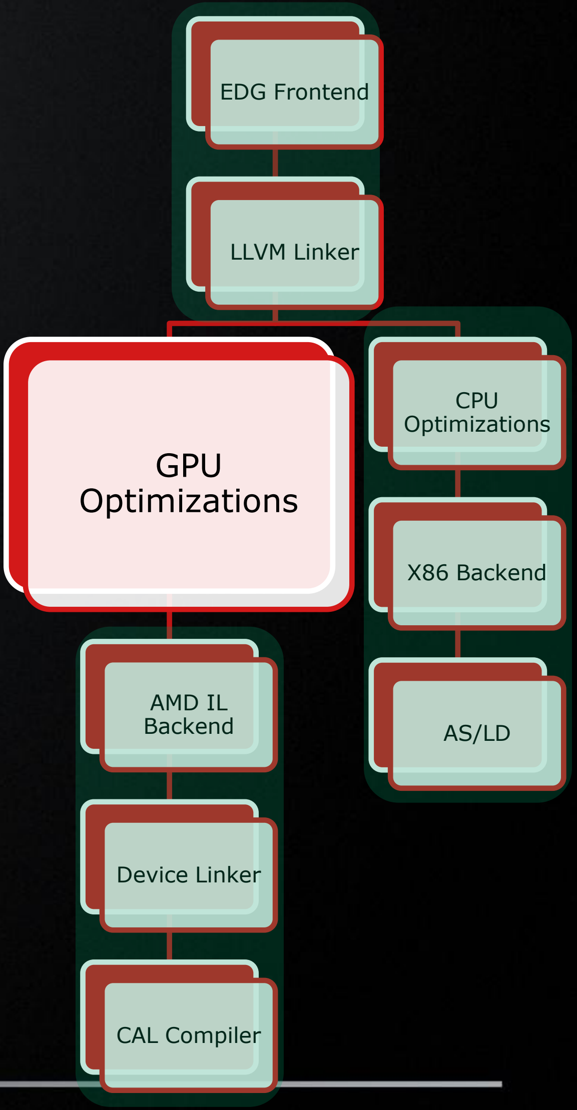
AMD The future is fusion

Summary

- Compiler Tool Chain
- Target Backends
- CPU Compiler Path
- **GPU Compiler Path**
- Ideas

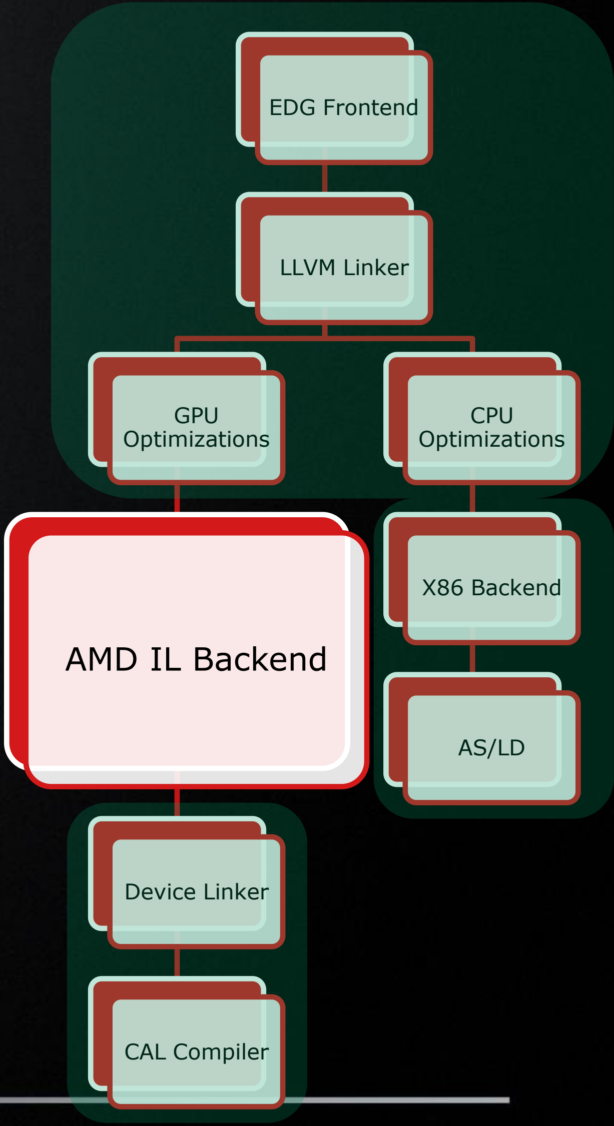
GPU Optimizations

- Modified FC optimizations
  - Loop Unswitch
  - Simplify CFG
  - Jump Threading
  - Loop Simplify
  - Loop Unrolling Pragma
- SCEV Binomial Coefficient
  - Limits bit width to 64bits
- Inline everything



AMD IL Backend

- Production backend targets AMD IL
- Backend is mainly self contained
- Required to support multiple LLVM versions simultaneously from the same codebase
- Required to work across OS X, Linux and Windows
- Supports all RV7XX and later AMD GPUs



AMD IL Language

- Forward compatible device agnostic pseudo-ISA
- Supports R300 and later chips
- Designed around graphics, not compute
- Virtual GPR set containing 2^16 4x32 registers
- Declare before use semantics
- Resources are globally scoped
- Compute heavy instruction set
- Functions, macros, flow control

AMD IL Constraints

- No unstructured control flow
- No calling conventions
- No extended or truncating memory ops
- Multiple register families(i.e. r#, l#, cb#, x#, mem, …)
- 32bit uniform memory region
  - 8/16/32 bit path different from 32/64/128 bit path
  - Different resource IDs for different paths
- Literals are 4 x 32 bits and declare before use
  - dcl_literal l3, 0x12348FFF, 12345678, 1.0, 1.5

AMD IL Constraints

- 5 main non-uniform memory regions
  - Device, Constant, Private, GDS and LDS memory
  - OpenCL Address spaces map directly to them
  - 137 unique sub-regions in device memory
    - 128 read-only image resources
    - 9 generic 32bit DWORD read/write resources
    - 1 32bit byte addressable read/write resource
  - 16 unique sub-regions in constant memory
  - Small device specific size limitations on non-device memory
  - Resource IDs must be known at compile time

AMDIL Examples

- Add a register and a literal
  - iadd r0, r0, l4 ← 4 x i32 add
- Add a constant and private memory
  - iadd r2, cb3[l10.z], x3[r0.x]
- Read a 32 bit value
  - uav_raw_load_id(3) r1.x, r2.x
- Read a 128 bit value
  - uav_raw_load_id(11) r1, r2.x
- Read a 8 bit value
  - uav_arena_load_id(8)_byte r1, r2.x

GPU Modifications

- Many optimizations disabled in optimization phase
- GPU custom passes to handle the following:
  - Printf – No native library support
  - Images – No way to represent exactly in LLVM
  - GPU specific peephole optimizations
- Hook added to SelectionDAG to disable conditional short circuiting on flow control (Bug7696)
- Some codegen fixes for vector types
- Overload CommonCodeGenPasses to disable MachineCSE

GPU Backend Workarounds

- MI:AsmPrinterFlags
  - Stores resource ID’s for I/O instructions
  - Stores load/store specific flags
    - Load cacheable
    - 8/16 bit store
    - Unknown Resource ID
- Tablegen - GCCBuiltins
  - Maps function calls to native instructions
  - Hack to allow function overloading to intrinsic mapping(i.e. sqrt_[f32|v2f32|v4f32|f64|v2f64] -> amdil.sqrt intrinsic)

Summary

- Compiler Tool Chain
- Target Backends
- CPU Compiler Path
- GPU Compiler Path
- **Ideas**

LLVM Improvements

- Tablegen
  - Vector immediates
  - Many to Many patterns
  - Function to instruction matching
  - Allow register Operand to match equivalent immediate
  - Method to access custom fields in Instruction
- LLVM
  - Allow expansion of all extload nodes
  - Vector channel encoded in Machine Operand
  - Vector machine friendly optimizations

Questions?

The future is fusion