

### LLVM Interpreter a key component in validation of OpenCL<sup>™</sup> compilers

<u>Oleg Maslov</u>, Alexey Bader, Yuri Veselov Intel April 30, 2013, The 3<sup>rd</sup> LLVM European meeting

OpenCL and the OpenCL logo are trademarks of Apple Inc. used by permission by Khronos

# Agenda

- OpenCL<sup>™</sup> standard overview
- OpenCL compiler standalone validation tool
- LLVM Interpreter
- Contribution to the LLVM community

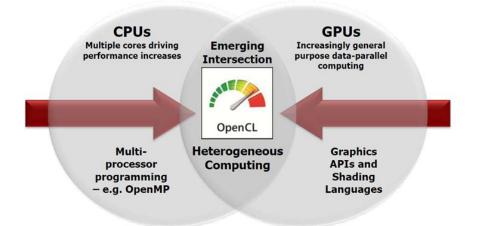






# OpenCL<sup>™</sup> Standard

### **OpenCL<sup>™</sup> - Open Standard for Parallel Computing**



- Open standard driven by Khronos\*
- Royalty-free
- First spec in 2009
- Cross-platform

Diagram based on PDF OpenCL overview available at http://www.khronos.org/opencl/





### **OpenCL<sup>™</sup> Standard Overview**

- Portable C code for all architectures
- Derived from ISO C99
  - Few restrictions, e.g. recursion, function pointers
  - Short vector types e.g., float4, short2, int16
  - Built-in functions: math (e.g., sin), geometric, common (e.g., min, clamp)
- Configurable N-dimensional computation domain
- Barriers and memory fences within workgroups
- Extensive list of optimized built-in functions

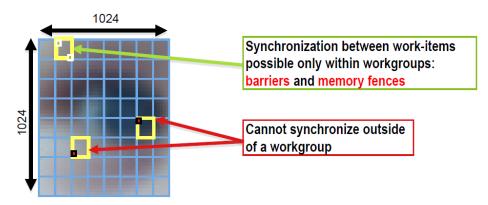


Diagram based on the PDF OpenCL<sup>™</sup> overview available at http://www.khronos.org/opencl/





# Intel<sup>®</sup> SDK for OpenCL<sup>™</sup> Applications 2013

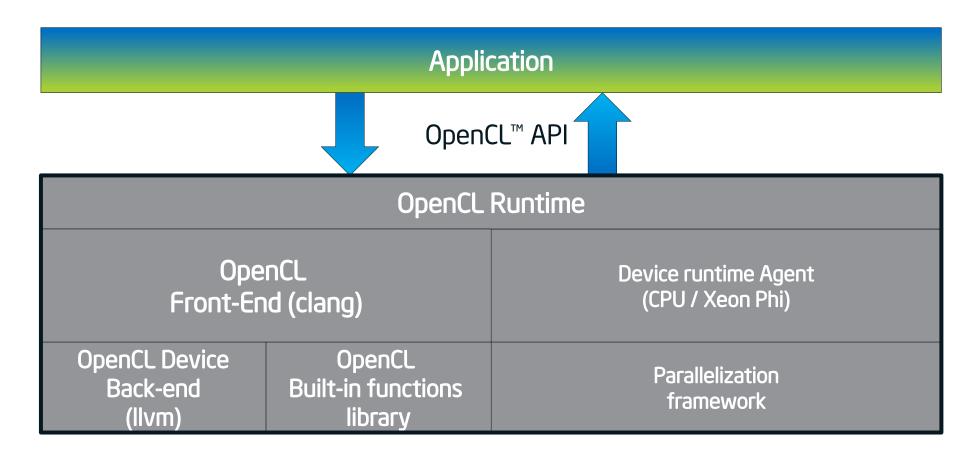
A Comprehensive Software Development Environment for OpenCL<sup>™</sup> Applications

	Target Processors	Target Operating System	OpenCL Spec Version	Target SDK	Developer Benefits
Visual Computing Domain	Cintel Inside Cintel Inside CORE 15 Inside CORE 17 COKE 15	Windows 7	OpenCL 1.2	(Version 2013)	Develop and deploy visual computing applications for 3 <sup>rd</sup> and 4 <sup>th</sup> Generation Intel <sup>®</sup> Core <sup>™</sup> Processors Get ready for next generations
Data Center Domain	veon' Vinside Xeon' Xeon' Xeon' Xeon' Phi	(Red Hat*, SUSE*)	OpenCL 1.2	(Version XE 2013)	Preserve your investment when developing high performance compute applications



### **OpenCL** compiler validation Standalone Tool

### Intel OpenCL<sup>™</sup> implementation architecture Running on Intel® CPU and Intel Xeon Phi<sup>™</sup> Coprocessor



#### OpenCL is multicomponent system challenging to validate





### **Conformance Tests** OpenCL<sup>™</sup> Implementation validation

- Provided and maintained by Khronos\*
- A compliance test for OpenCL<sup>™</sup> specification
- System level tests
  - Tests OpenCL implementation as whole entity
  - All components should be functional

#### Validation on component level is not trivial







# **OpenCL<sup>™</sup> Standard**

### **Supporting Intel® Many Integrated Core Architecture**

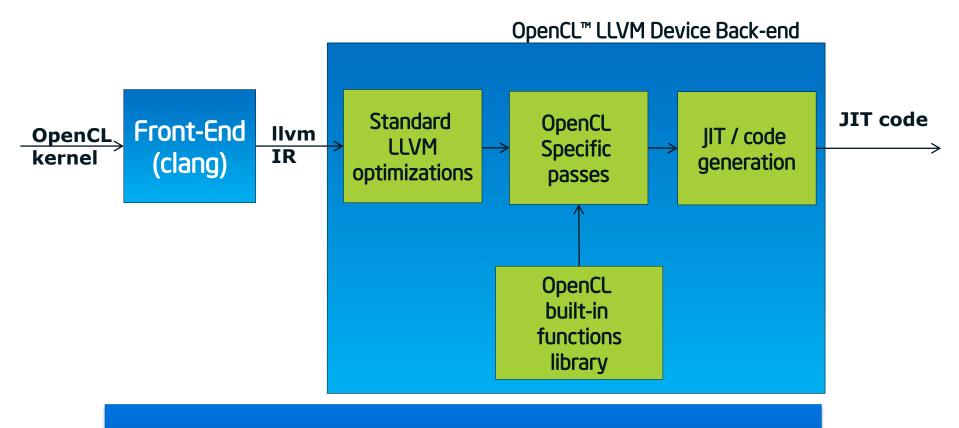
- Simultaneous development of multiple components
  - Runtime
  - Device agent
  - Device Back-end
  - OpenCL<sup>™</sup> built-in functions library
  - ...
- Conformance tests
  - Not functional until all components are ready
- Early testing is highly needed
  - Boosts development speed

#### How to validate device backend?





### OpenCL<sup>™</sup> Compiler for Intel® CPU and Intel Xeon Phi<sup>™</sup> Coprocessors Zoom in



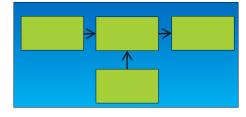
#### How to validate device backend?







#### LLVM Device Backend



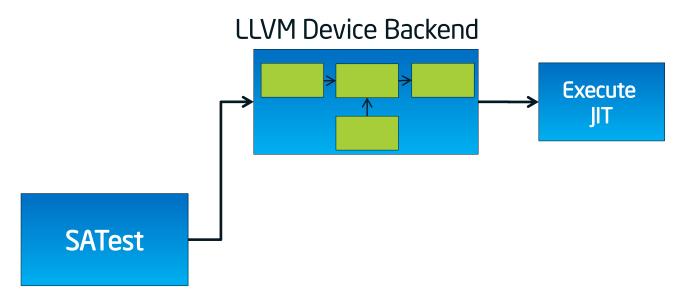
#### IDEA!!! Isolate testing device backend from other components

13 4/30/2013





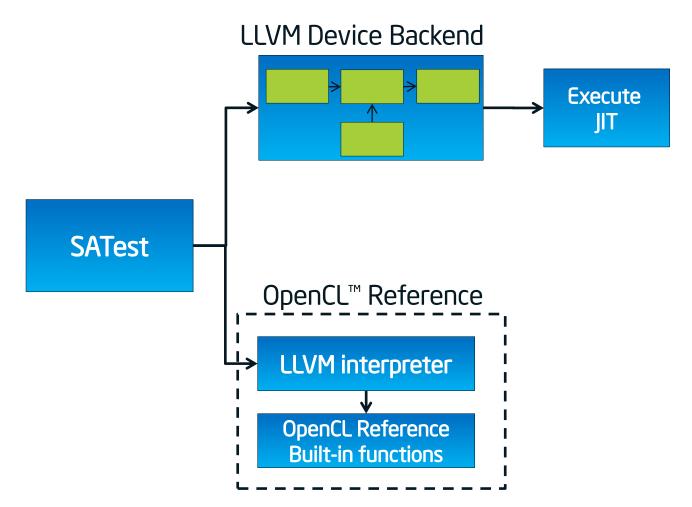




#### Implement standalone tool to work with device back-end







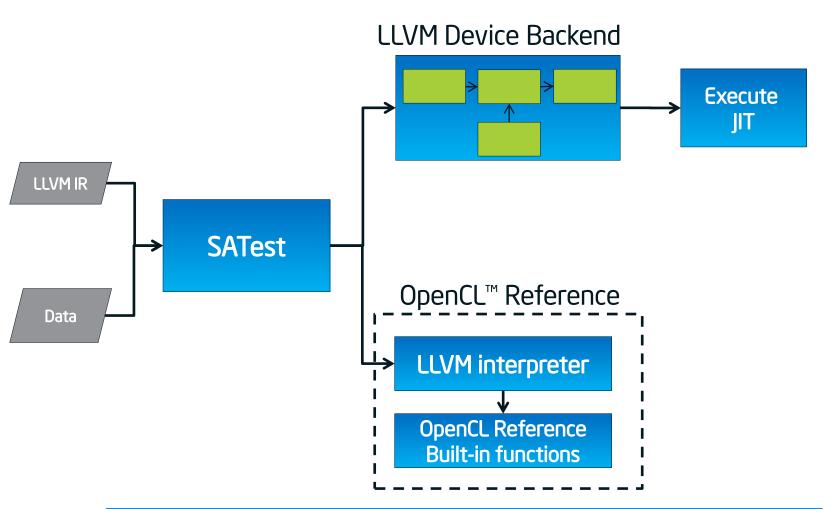
Implement OpenCL Reference to obtain reference results Share the same build/execute interface with back-end

15 4/30/2013

Copyright© 2013, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.





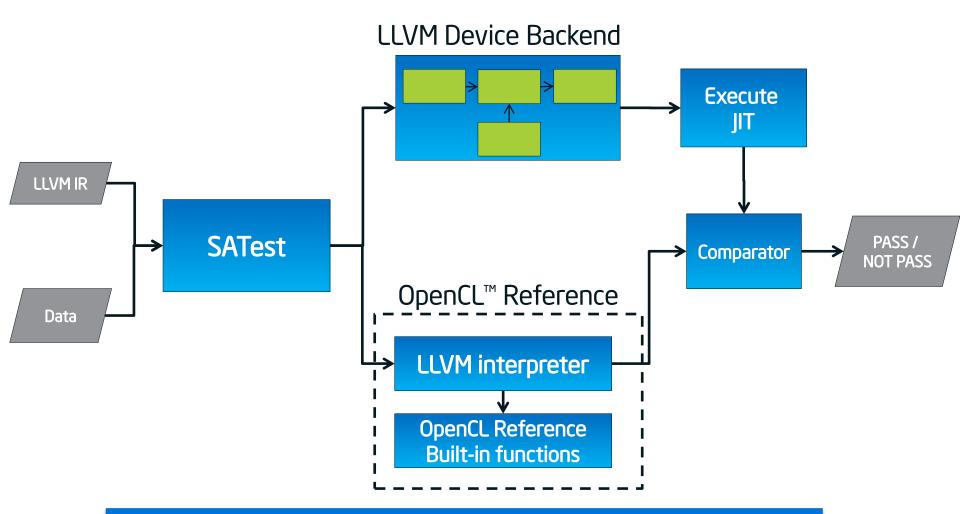


#### Import LLVM kernel compiled from OpenCL Import data recorded from running OpenCL kernel





Copyright© 2013, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.



#### Compare Reference and Back-end outputs Report PASS/NOT PASS



Copyright© 2013, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners.





### **Standalone compiler validation tool** Features

#### Several modes of operation

- Functional Validation
  - Check device back-end produces OpenCL<sup>™</sup> 1.2 conformant results
  - Real life kernels and data
- Performance
  - Compile time
  - Kernel execution time
  - Single thread execution
  - Isolate and Detect issues coming from backend
- Build
  - Dump LLVM IR and JIT code on specified pass
  - Directly track/debug optimizations

### Tool for backend developers



# LLVM Interpreter

# **LLVM Interpreter**

- MUST have component for OpenCL<sup>™</sup> Reference
- Executes LLVM bytecode directly
- Produces bitwise accurate results
- Slow
- No optimizations
- LLVM trunk version lacks vector and aggregate types support. Need to implement them

Produces reference results of running OpenCL kernel





# **Missing pieces in LLVM interpreter**

### **OpenCL**<sup>™</sup>

### LLVM IR

define void @cl\_exp(<4 x float> addrspace(1)\* %a, <4 x float> kernel void cl\_exp(global const float4 \*a, addrspace(1)\* %result) nounwind { global float4 \*result) { %1 = call i64 @get\_global\_id(i32 0) nounwind readnone %sext = shl i64 %1, 32 %2 = ashr exact i64 % sext, 32int id = get global id(0); %3 = getelementptr inbounds <4 x float> addrspace(1)\* %a, result[id] = exp(a[id]);i64 %2  $\%4 = \text{load} < 4 \times \text{float} > \text{addrspace}(1)^* \%3$ , align 16  $\%5 = call <4 \times float > @ Z3expDv4 f(<4 \times float > \%4)$ nounwind readnone %sext1 = shl i64 %1, 32 %6 = ashr exact i64 %sext1, 32 %7 = getelementptr inbounds <4 x float> addrspace(1)\* %result, i64 %6 store <4 x float> %5, <4 x float> addrspace(1)\* %7, align 16 ret void } Not supported in LLVM interperter



### **LLVM Interpreter: Current State** Essential Methods of Interpreter

class Interpreter : public ExecutionEngine, public
InstVisitor<Interpreter> {

// The runtime stack of executing code.

// current function record.

std::vector<ExecutionContext> ECStack;

#### All interpreter logic encapsulated into 'Interpreter' class

...

•••







### **LLVM Interpreter: Current State** Essential Methods of Interpreter

class Interpreter : public ExecutionEngine, public
InstVisitor<Interpreter> {

 $///\ {\rm run}$  - Start execution with the specified function and arguments.

/// Opcode Implementations(e.g. void visitLoadInst(LoadInst &I))
void visit\*();

#### visit\*() methods execute instructions

...

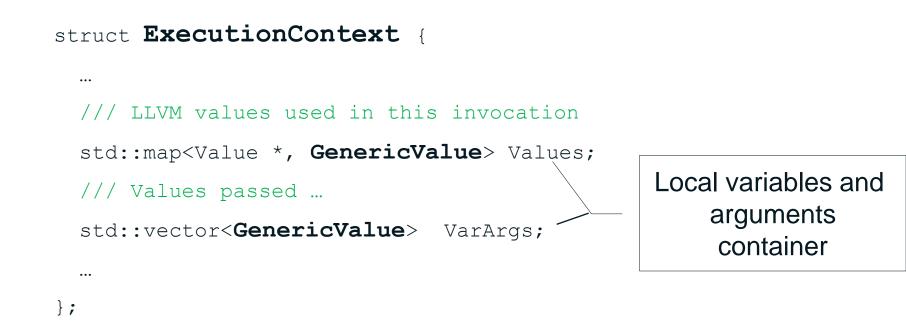
...







### **LLVM Interpreter: Current State** Execution Context



#### Stores current basic block, function and next instruction







# **GenericValue Structure**

#### struct GenericValue {

```
union {
    double DoubleVal;
    float FloatVal;
    PointerTy PointerVal;
    struct { unsigned int first; unsigned int second; } UIntPairVal;
    unsigned char Untyped[8];
  };
  APInt IntVal; // also used for long doubles
    ...
};
```

#### Universal container for variable values





# **GenericValue Structure**

#### struct GenericValue {

};

. . .

Adding new single field covers vector and aggregate types







# **Example of New visit\* Method**

### Typical visit\* method

```
void visit*(Instruction& I)
{
    ExecutionContext &SF = ECStack.back();
    const Type *Ty = I.getType();
    GenericValue Src1 = getOperandValue(I.getOperand(0), SF);
    GenericValue Src2 = getOperandValue(I.getOperand(1), SF);
    GenericValue R; // Result
    switch (Ty->getTypeID())
    {
        case: Type::IntegerTyID: // R.IntVal = Src1.IntVal OP Src2.IntVal;
        case: Type::FloatTyID: // R.FloatVal = Src1.FloatVal OP Src2.FloatVal;
        ...
}
```



# **Example of New visit\* Method**

### Typical visit\* method

```
void visit*(Instruction& I)
  ExecutionContext &SF = ECStack.back();
  const Type *Ty = I.getType();
  GenericValue Src1 = getOperandValue(I.getOperand(0), SF);
  GenericValue Src2 = getOperandValue(I.getOperand(1), SF);
 GenericValue R; // Result
  switch (Ty->getTypeID())
    case: Type::IntegerTyID: // R.IntVal = Src1.IntVal OP Src2.IntVal;
    case: Type::FloatTyID: // R.FloatVal = Src1.FloatVal OP Src2.FloatVal;
    case: Type::VectorTyID:
      // set vector size
       R.AggregateVal.resize(Src1.AggregateVal.size());
       if (cast<VectorType>(Ty)->getElementType()->isFloatTy())
       // process R.FloatVecVal
       if (cast<VectorType>(Ty)->getElementType()->isIntegerTy())
       // process R.IntVecVal
```

#### Processing vectors





### **Interpreter** Operations implemented

Group	Number of operations
Vector Memory Access and Addressing	4
Vector binary	12
In-vector	3
Vector comparison	3
Vector bitwise binary	6
Vector conversion	10
Aggregate	2



### **OpenCL<sup>™</sup> Reference Tool** Built-in Functions Implemented

- High-precision reference implementation
- Full support of OpenCL<sup>™</sup> 1.2 standard
- Over 3000 functions
  - Math
  - Geometric
  - Common
  - Image
  - Conversions between types (~2000)
  - etc













### OpenCL<sup>™</sup> Back-end Validation for Intel® Xeon Phi<sup>™</sup> Coprocessor Summary

- Enabled validation at early stage
  - At 1<sup>st</sup> stages on functional simulator
  - Other components of OpenCL<sup>™</sup> infrastructure not yet ready
  - Before hardware was ready
- Saved time on isolated backend development

OpenCL Runtime					
Ope Front-En		Device runtime Agent (CPU / Xeon Phi)			
OpenCL Device Back-end (Ilvm)	OpenCL Built-in functions library	Parallelization framework			
		Optimization (intel)			

Copyright© 2013, Intel Corporation. All rights reserved. \*Other brands and names are the property of their respective owners. Notice 🖵

# Contribution to the LLVM Community

### **Interpreter** Summary of our Modifications

- Added Vector and aggregate types to Execution Context
- Added Vector operations
  - Memory access and addressing
  - Binary
  - In-vector
  - Comparison
  - Constant
- Added Aggregate types operations
  - 'extractvalue'
  - 'insertvalue'



# **Contribution to the LLVM Community**

- Uploading interpreter changes to LLVM repository
- Incremental commits adding new features to interpreter
- Vectors and aggregate types will be supported in interpreter



Optimization

# **Wish list for the Interpreter**

- Add standard C LLVM intrinsics support
- Enhance processing of constant expressions

4/30/2013







- Multiple components are developed simultaneously
- Existing tests work only after all components are ready
- Lack of Early validation
- We provide early validation infrastructure using LLVM interpreter







# Thank you

## **Legal Disclaimer & Optimization Notice**

INFORMATION IN THIS DOCUMENT IS PROVIDED "AS IS". NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

Copyright ©, Intel Corporation. All rights reserved. Intel, the Intel logo, Xeon, Xeon Phi, Core, VTune, and Cilk are trademarks of Intel Corporation in the U.S. and other countries.

#### **Optimization Notice**

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804



