SPIR me the details: building custom language support on OpenCL

Neil Henning - Technology Lead

Who am I?

- Five years in the industry
- Spent all of that using SPUs, GPUs, vector units & DSPs
- Last three years implementing compute for customers (OpenCL, RenderScript & other proprietary compute systems)
- Passionate about making compute easy for developers

Who are Codeplay?

- Heterogeneous Systems Experts
- Founded by Andrew Richards (pictured right!)
- 35 engineers based out of Edinburgh, Scotland
- Compilers, debuggers, test suites, & much more
- Mostly work with LLVM, Clang, LLDB & LLD
- Both off-the-shelf products & contractual work

Agenda

- What is SPIR?
- Why should you use SPIR?
- What tools are available to use SPIR?
- How do you make your language support SPIR?
- What kind of tools can you make using SPIR?
- What is the future of SPIR?

What is SPIR?

- Standard Portable Intermediate Representation
- Khronos standardization of LLVM IR format (LLVM 3.2)
- Vendor agnostic, platform agnostic, device agnostic
- Two variants, spir-unknown-unknown & spir64-unknown-unknown
- Allows for
  - Offline compilation
  - Online re-optimization
  - Custom language support on OpenCL

```
// OpenCL C Kernel Language
void kernel foo(
      global int * a,
      global int * b)
{
      *a = *b;
}

// SPIR human readable representation
define spir_kernel void @foo(
      i32 addrspace(1)* %a,
      i32 addrspace(1)* %b)
{
      %1 = load i32 addrspace(1)* %b,
            align 4
      store i32 %1, i32 addrspace(1)* %a,
            align 4
      ret void
}
```

What is SPIR?

- SPIR 1.2 is the current standardized version!
- SPIR 1.2 to match OpenCL 1.2
- SPIR inherits the restrictions of OpenCL C 1.2 Language
  - No recursion
  - Distinct address spaces for data
  - Can call any OpenCL 1.2 builtin function
  - SPIR is an inherently parallel IR

Why should you use SPIR?

- SPIR allows you to ship only a binary
- Means you *don't* have to ship source!
- Which means people can't just make off with your work
  - The caveat being that SPIR like any IR could still be stolen though!
  - SPIR does mean though that its harder to reproduce the original intention of the code
- Money, time, effort, energy & team morale preserved

Why should you use SPIR?

- SPIR can speed up OpenCL load times!
  - Compiler frontend (normally Clang) doesn't need to be involved
  - Code is just 'finalized' for the OpenCL target
  - Makes user experience better!
- Allows custom language support!
- Allows us to create tools!

What tools are available to use SPIR?

- Two implementors of the SPIR specification in the wild
- Intel OpenCL SDK targeting CPU, Xeon Phi & Intel Integrated GPU
- AMD's APP SDK targeting CPU & GPU
- Both OpenCL implementations can consume SPIR
- Khronos SPIR Generator built on Clang + LLVM 3.2
    - We can use this generator to turn OpenCL C kernels into SPIR

What tools are available to use SPIR?

- To verify our SPIR kernels are valid, we can use Khronos SPIR Verifier
- For profiling, we can follow Anteru's Guide to AMD's GPUPerfAPI
- Or Intel's (slightly more costly) VTune tool
- Can also use Oclgrind to interpret our kernels (useful to compare against real devices in case you suspect foul play)

How do you make your language support SPIR?

- Having custom languages that can target SPIR is awesome
- Many use cases for why allowing custom languages matters
- To enable your language
  - You need to produce SPIR compliant IR
  - Build your tool on LLVM 3.2, mimicking the SPIR producer provided by Khronos
  - Use the SPIR verifier to prove your code is ok
  - Run it through Oclgrind on host to check the logic is sound
  - Only then point it at an in-the-wild implementation

How do you make your language support SPIR?

- But we need to be aware of the SPIR 1.2 standard's restrictions
  - Need to be able to represent address spaces (or at least default all pointers to the global address space)
  - Need to be able to segregate work into parallel chunks
  - No support for recursion, so need to ban that in the languages that are targeting SPIR
  - Could also expose OpenCL builtins to the language
- And decide whether to expose OpenCL or just hide it underneath your own language constructs

How do you make your language support SPIR?

- SPIR enabled Codeplay to lead development of the new Khronos SYCL standard
- Any language that can use LLVM 3.2 as a backend could be made to support SPIR
- At the very least Clang 3.2 allows us to target ObjC, C, & C++ onto a SPIR implementation
- Also possible to 'backport' future LLVM IR, to LLVM 3.2 IR, but it is not a trivial task

What kind of tools can you make using SPIR?

- Online and offline re-optimizers!
- Can modify SPIR libraries!
- Can modify *other people's* SPIR libraries!
- Maybe change tan -> native_tan because you don't care about precision?
- Can be done offline (buy a SPIR library, specialize it for your application, ship!)
- Or online (application developer links in a SPIR library, can intercept and re-imagine!)

```
// Before
define spir_func <4 x float>
@myAwesomeFunction(
      <4 x float> %in) {
      %out = call <4 x float> @_Z3tanDv4_f(
            <4 x float> %in)
      ret <4 x float> %out
}

// After
define spir_func <4 x float>
@myAwesomeFunction(
      <4 x float> %in) {
      %out = call <4 x float>
            @_Z10native_tanDv4_f(
                  <4 x float> %in)
      ret <4 x float> %out
}
```

What kind of tools can you make using SPIR?

- Static analysis tools!
- Can take other people's libraries and prove features of them
  - Are they safe?
  - Do they use global data correctly?
  - Do they call some functions I don't want them too?
- Imagine a library we use has complicated integer arithmetic offsetting into a global array
- We could use static analysis on the SPIR library to warn us that some logic is not to our liking!

```
int myAwesomeFunction(global int * a)
{
      int offset = get_global_id(0);

      // now really mess up offset, in some
      // hilariously bad way

      // ...

      // offset into global array a
      // could be out of bounds!
      return a[offset];
}
```

What kind of tools can you make using SPIR?

- Debuggers and profilers!
- Not every platform that *will* support SPIR has the tool support of Intel or AMD
- So can we build our own?
- What if we buy in a SPIR library - and want to debug that?
  - We can use SPIR to create our own debuggers (kinda)
  - At the very least we can inject printf calls into library code!
- We could also split up a kernel into multiple sub-kernels for profiling!

```
// Before
void kernel myAwesomeKernel(
     global int * a, global int * b) {
     // ... do a, b & c ...
}

//After
void kernel myAwesomeKernelA(
     global int * a, global int * b,
     global struct AToBState * out) {
}
void kernel myAwesomeKernelB(
     global int * a, global int * b,
     constant struct AToBState * in,
     global struct BToCState * out) {
}
void kernel myAwesomeKernelC(
     global int * a, global int * b,
     constant struct BToCState * in) {
}
```

What is the future of SPIR?

- SPIR will track future version of the OpenCL specification
- OpenCL 2.0 -> SPIR 2.0
- OpenCL 2.1 -> SPIR 2.1
- This means any feature added to OpenCL, should be available in SPIR!
- Means all of the great innovation going into the OpenCL specification becomes available to our custom languages too!

Wrap up

- OpenCL SPIR specification is useful, understandable & powerful
- Enables non-vendors to interact with awesome hardware in a new and excited way!
- We can create languages and tools and throw them at the hottest hardware around (providing more vendors come forward with SPIR support!)
- Please provide feedback (Khronos forums for SPIR is a good place to holler!)
- I look forward to seeing all the awesome applications and libraries that *you* all will come up with!

Questions?

Neil Henning

Email - neil@codeplay.com

Twitter - @sheredom