Steckdenis' developer blog

Development news and thoughts

May 2, 2011

Using Clang to compile OpenCL kernels

By steckdenis
Hello,

After about a week, here are some news about my Summer work. I will describe here how I want to implement the compiler part of OpenCL.

Using Clang to compile kernels

I'll begin with a good news : an incredible Free C compiler exists : Clang. It is very easily used and embedded into applications, and runs very fast. It is also compatible with the OpenCL C subset I have to implement.

A few days ago, I tried to make Clang compile some simple kernels. I used for that the command line, not an application. My test kernel was the following :

```
/* Header to make Clang compatible with OpenCL */
#define __global __attribute__((address_space(1)))

int get_global_id(int index);

/* Test kernel */
__kernel void test(__global float *in, __global float *out) {
        int index = get_global_id(0);
        out[index] = 3.14159f * in[index] + in[index];
}
```

This code, placed in the file "test.cl", can be compiled with Clang using the following command line :

```
clang -S -emit-llvm -o test.ll -x cl test.cl
```

The interesting part is "-x cl", which tells the compiler to enable its OpenCL compatibility features. These features include more type checks (like a warning if I don't put a "f" after the float value, saying that double precision floating point needs an OpenCL extension) and a very nice feature : the recognition of "__kernel". All the kernels of a source file are placed in a special LLVM section. Here is the LLVM assembly output by Clang :

```
; ModuleID = 'test.cl'
target datalayout = "e-p:64:64:64-i1:8:8-i8:8:8-i16:16:16-i32:32:32-i64:64:64-f32:32:32-f64:64:64-v64:64:64-v128:128:128-a0:0:64-s0:64:64-f80:128:128-n8:16:32:64"
target triple = "x86_64-unknown-linux-gnu"

define void @test(float addrspace(1)* nocapture %in, float addrspace(1)* nocapture %out) nounwind {
entry:
  %call = tail call i32 @get_global_id(i32 0) nounwind
  %idxprom = sext i32 %call to i64
  %arrayidx = getelementptr inbounds float addrspace(1)* %in, i64 %idxprom
  %tmp2 = load float addrspace(1)* %arrayidx, align 4, !tbaa !1
  %mul = fmul float %tmp2, 0x400921FA00000000
  %add = fadd float %mul, %tmp2
  %arrayidx11 = getelementptr inbounds float addrspace(1)* %out, i64 %idxprom
  store float %add, float addrspace(1)* %arrayidx11, align 4, !tbaa !1
  ret void
}

declare i32 @get_global_id(i32)

!opencl.kernels = !{!0}

!0 = metadata !{void (float addrspace(1)*, float addrspace(1)*)* @test}
!1 = metadata !{metadata !"float", metadata !2}
!2 = metadata !{metadata !"omnipotent char", metadata !3}
!3 = metadata !{metadata !"Simple C/C++ TBAA", null}
```

We can see that the "__global" macro is expanded and that address-space information can be found in the LLVM output. The "opencl.kernels" metadata entry also contains the signature of my kernel.

Running the LLVM code

Compiling a code is good, running it is better. For my Google Summer of Code, I want to implement a software OpenCL library, with all the needed stuff to be able to easily add an hardware accelerated path.

To do that, I will use an abstraction layer : a class named for instance "KernelRunner" that will be subclassed by "CPUKernelRunner" and "GalliumKernelRunner". The CPUKernelRunner class will use the LLVM JIT to run the kernels. The Gallium one will transfort the LLVM bitcode generated by Clang to TGSI, or will pass it directly to the Gallium drivers if they support that when I will implement that.

This class will also translate some state information, for instance the bound arguments. I hope it will be fairly easy with the LLVM JIT, but it could be a bit more difficult for the accelerated path (transferring data to the GPU isn't an easy task).

The end word

To close this post, I'll say that I'm pretty confident that I will be able to do interesting things. I have plenty of Free Software project to rely upon, and many great people. I already asked the Clang mailing-list how to properly integrate Clang in a library and got a response. I have a question posted on the Mesa mailing list that currently has no response, but it doesn't stop me.

Thanks for reading.

Advertisements

This entry was posted on Monday, May 2nd, 2011 at 20:50 and posted in Research. You can follow any responses to this entry through the RSS 2.0 feed.

7 responses to "Using Clang to compile OpenCL kernels"

- *blob » OpenCL without hardware.*

  May 18th, 2011 at 22:35
  […] of Denis latest posts contained the hint that llvm's clang – from rawhide – already provides (some) […]

  Reply

- *pankaj86*

  May 19th, 2011 at 08:32
  Awesome, i've been waiting for an open source opencl implementation for quite some time.
  What part of opencl is currently not supported by the "-x cl" option of clang. It appears from your post that the syntax extensions are supported but a few (many) of the opencl functions are not supported

  Reply

- *Shwetashree*

  September 2nd, 2011 at 08:27
  Its really good! Is the output of clang in LLVM IR or in assembly as you have suggested above? If its in assembly how do i get output in LLVM IR?

  Reply

- *eya*

  September 28th, 2011 at 09:56
  Really interesting post. Can you generate hardware description language here? did u used transformation and optimization processes to generate LLVM assembly output?

  Reply

- *fabeirojorge*

  November 17th, 2011 at 14:14
  Hi! Very interesting post. Do you know how to tell Clang that is going to parse an OpenCL file ("-x cl") when you use it in a program instead of as command-line tool?

  Reply

- *steckdenis*

  November 19th, 2011 at 17:00
  Hello,

  You can see what I've done in http://cgit.freedesktop.org/~steckdenis/clover/tree/src/core/compiler.cpp . Basically, you only have to say to Clang that an input file is in OpenCL C, by using (for instance) :

  frontend_opts.Inputs.push_back(std::make_pair(clang::IK_OpenCL, "program.cl"));

  Reply

  - *fabeirojorge*

    November 20th, 2011 at 13:33
    Thank you so much for your help. I've tried to apply this changes to my code, but I'm not compiling it (i don't need at all the LLVM-IR representation of the code), just creating a Preprocessor (that doesn't admit a FrontendOptions instance as an attribute) in order to parse the code with clang::ParseAST method and do some source-to-source transformation using a class of my own that inherits from ASTConsumer, DeclVisitor and StmtVisitor.

    Given that, I've only tried "_langOpts.OpenCL = true;", but my visitor just crash when it arrives to the firsl line of the kernel function. (I've added to my kernel the two lines you propose in the post).

    Thank you so much again.

    Reply

Subscribe to RSS
Create a free website or blog at WordPress.com.