Denis Steckelmacher
Free Software and Research

Using Clang to compile OpenCL kernels

Published on lun 02 mai 2011 in Clover, (0 Comments)

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
target triple = "x86_64-unknown-linux-gnu"

define void @test(float addrspace(1)* nocapture %in, float addrs
{
    entry:
        %call = tail call i32 @get_global_id(i32 0) nounwind
        %idxprom = sext i32 %call to i64
```

Categories

- Clover *(22)*
- Nepomuk *(38)*
- Personal *(2)*
- VHDL *(3)*

See also

- My GitHub page
- GSoC 2011 repos
- GSoC 2013 nepomuk-core
- GSoC 2013 nepomuk-widgets
- GSoC 2014 kdev-qmljs

```
        %arrayidx = getelementptr inbounds float addrspace(1)* %
        %tmp2 = load float addrspace(1)* %arrayidx, align 4, !tb
        %mul = fmul float %tmp2, 0x400921FA00000000
        %add = fadd float %mul, %tmp2
        %arrayidx11 = getelementptr inbounds float addrspace(1)*
        store float %add, float addrspace(1)* %arrayidx11, align
        ret void
}

declare i32 @get_global_id(i32)

!opencl.kernels = !{!0}
!0 = metadata !{void (float addrspace(1)*, float addrspace(1)*)*
!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.

« Launch of my project   Repository available »

Comments for this thread are now closed. ×

0 Comments **Steckdenis' developer blog** 1 **Login**

Recommend 1 Share Sort by Best

This discussion has been closed.

Subscribe Add Disqus to your siteAdd DisqusAdd Privacy