Heterogeneous Computing with D

Using the PTX and SPIR-V targets with a system programming language

Kai Nacke

4 February 2018

LLVM dev room @ FOSDEM‘18

GPU Architecture

- Uses Single Instruction Multiple Thread (SIMT) model
  - Contains a huge number of cores
  - Blocks of threads executing the same code (“kernel”)
  - Hierarchy of Grid / Block / Warp

- Different memory types
  - Global and local memory
  - Shared memory
  - Constant memory
  - Memory may be cached
  - Register file uses memory, too

- Possible lot of restrictions, e.g. no double data type or no recursion

Development model

- There are open and proprietary APIs
  - OpenCL
  - CUDA
- General development flow
  - Kernels are developed in a high-level language
  - Compiled into an intermediate representation
  - GPU driver compiles to machine instructions during load time
- Host application loads and launches kernels

Applications of GPU computing

- Well suited for many scientific applications

- Examples
  - Machine learning
  - Monte-Carlo simulations
  - Pattern matching
  - Ray tracing
  - …

- Games

Challenges for developers

- Application developer
  - SIMT requires different thinking
  - Different memory spaces increase complexity

- Compiler developer
  - Application source compiles to different targets
  - Address spaces must be mapped to language constructs
  - Separation of compiler and library

LLVM support

- LLVM generates code for different GPU's
  - nvptx, nvptx64 (http://llvm.org/docs/NVPTXUsage.html)
  - r600, amdgcn (http://llvm.org/docs/AMDGPUUsage.html)

- OpenCL uses SPIR-V
  - SPIR is derived from LLVM IR
  - No SPIR-V backend in official LLVM
  - Requires custom LLVM (https://github.com/thewilsonator/llvm/releases)

A closer look at the PTX backend

- Global variables and pointer types need an address space

```
%res = alloca float addrspace(1)*
```

- Functions have a different calling convention

```
define ptx_kernel void @mykernel()
declare ptx_device i32 @llvm.nvvm.read.ptx.sreg.warpsize()
```

- Kernel functions need special metadata

```
!nvvm.annotations = !{!0}
!0 = !{void ()* @mykernel, !"kernel", i32 1}
```

D source level

```
@compute(CompileFor.deviceOnly)
module dcompute.tests.kernels;
pragma(LDC_no_moduleinfo);

import ldc.dcompute;
import dcompute.std.index;

@kernel void saxpy(GlobalPointer!(float) res,
                   GlobalPointer!(float) x,
                   GlobalPointer!(float) y,
                   float alpha, size_t N)
{
    auto i = GlobalIndex.x;
    if (i >= N) return;
    res[i] = alpha*x[i] + y[i];
}
```

Implementation details

- Uses attributes and templates
  - @compute(), @kernel, struct Pointer(AddrSpace as, T)
  - Requires **`–betterC`** switch because of target limitations (no garbage collection, no exceptions, no module init, …)
- Compiles to different targets in one invocation
  - Requires tweaks because of global variables (DataLayout, Target, …)

```
ldc2 -mdcompute-targets=cuda-500 -oq -betterC dcompute\tests\kernels.d
```

- New ABI implementation as usual
- A lot more stuff in runtime library

Conclusion

- Adding GPU targets was pretty easy
  - Global variables in compiler were main obstacle
- Still work in progress, especially the library
  - Library should enable uniform handling of PTX and OpenCL
- What we really miss is a SPIR-V backend in official LLVM

Questions?