OpenMP Accelerator Offloading using OpenCL with SPIR-V

Daniel Schürmann | AES | Lightning Talk
OpenMP Accelerator Offloading

#pragma omp target
• map code section to a device

#pragma omp teams
• create a league of independent thread teams

#pragma omp distribute
• distribute loop over the thread teams

```c
#pragma omp target teams distribute parallel for
for (int i = 0; i < n; ++i) {
    A[i] = B[i] + C[i]
}
```
Problem

• GPUs offer high performance and efficiency, but are difficult to program

• Working implementations exist for NVPTX/Cuda and Intel Xeon Phi.

• Specification is available since 2 years but there is no sight of support for OpenCL devices.
Idea & Motivation

Enable all OpenCL 2.1 devices to be targeted by OpenMP accelerator offloading:

- simplify the parallel programming of heterogeneous systems
- easily convert existing scalar code for GPUs
- potential target for libraries/programming languages to provide single-source GPGPU capabilities
Implementation: Clang Driver

- Selects Toolchain for each target
- spirv-link from Khronos Group spirv-tools

*) Uses LLVM-backend from Nicholas Wilson
Implementation: \#pragma omp parallel

\#pragma omp target
{
<some code goes here>

\#pragma omp parallel
{

...

}
<some code goes here>
}
Implementation: `#pragma omp parallel`

```c
#pragma omp target
IF (thread_idx == 0)
<some code goes here>

#pragma omp parallel
(copy shared variables to local memory)
ENDIF
   Call inlined parallel function
IF (thread_idx == 0)
(copy shared variables back)
<some code goes here>
```
Implementation: Run-Time Library

Write a run-time plugin for OpenMP to offload device code to the GPU
Benchmark: LULESH

• Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics Benchmark

• Models hydrodynamics as representing a typical scientific application

• Studies behavior of fluid flow when subject to forces

• Implementations: CUDA, OpenCL, OpenMP, MPI, serial for various targets
Benchmark: LULESH

Challenge 1: missing #pragma omp declare target

• Declares a function definition to be available on the device
• Solution: Inline everything! 😊

Challenge 2: Math Functions

• LULESH uses functions included by <math.h>
• Solution: Use Itanium name mangling to use OpenCL library functions
Benchmark: LULESH

System:
CPU: AMD Ryzen 1700
GPU: AMD Radeon RX 560
Benchmark: Mandelbrot

- Offloaded version benchmarked on AMD Radeon RX 560
- Scalar version benchmarked on AMD Ryzen 1700
- Resolution: 3840x2160

```c
#pragma omp target teams map(from:output[0:width*height]) thread_limit(width*heigth)
#pragma omp distribute parallel for collapse(2)
for (int j = 0; j < height; j++) {
    for (int i = 0; i < width; ++i) {
        float x = x0 + i * dx;
        float y = y0 + j * dy;
        ...
    }
}
```
Benchmark: Mandelbrot

Speed-up compared to serial implementation

- Serial
- OpenMP GPU
- OpenCL GPU
- OpenCL CPU

Comparison with and without compilation overhead:
- w/ compilation overhead
- w/o compilation overhead
Conclusion & Future Work

• We could demonstrate the functionality & efficiency of the approach
• SPIR-V linker available
• OpenCL C library functions available

• No optimizations are enabled yet
• Reduction clause not implemented
• OpenMP library functions not available
Source Code Available

- Clang:
  https://github.com/daniel-schuermann/clang

- LLVM:
  https://github.com/thewilsonator/llvm/tree/compute

- OpenMP:
  https://github.com/daniel-schuermann/openmp