Improving the OpenMP Offloading Driver: LTO, Libraries, and Toolchains

LLVM Performance Workshop @ CGO 2022
April 3rd 2022
Joseph Huber
Overview & Motivation
OpenMP Offloading Overview

- Allows users to offload execution of code to another device
- Requires the compiler driver to compile & link multiple programs
- The linked image also needs to be registered

```c
#include <complex>

using complex = std::complex<double>;

void zaxpy(complex *X, complex *Y, complex D, int N) {
    #pragma omp target teams distribute parallel for
    for (int i = 0; i < N; ++i)
        Y[i] = D * X[i] + Y[i];
}

int main() {
    const int N = 1024;
    complex X[N], Y[N], D;
    #pragma omp target data map(to:X[:N]) map(tofrom:Y[:N])
    zaxpy(X, Y, D, N);
}
```
Current OpenMP Offloading Driver Overview

Compilation

C/C++ and OpenMP → Clang

Dev IR → LLVM

Dev ASM → Dev Obj

Vendor tools

Host IR → LLVM

Host Obj → Bundler

Fat Binary

Linking

Fat Binary → Clang

Dev Obj → Dev Exe

Vendor tools

Wrapper

Host Image

Exe

libomptarget (host, plugins)

Different stages for AMDGPU and NVPTX!
Motivation

• Why isn't the current method good enough?
• Handle device code the same as host code and support **static linking**
• Unify the required stages across **all toolchains**
• Support **Link Time Optimization** on the device
• Enable offloading language **interoperability**
New Driver Implementation
New OpenMP Offloading Driver

- Embed the device objects directly in the host object
  - Data is stored in an excluded section
- Linking is done by a linker wrapper application
  - Scan each input for embedded device objects
  - Extract & link each image
  - Wrap the linked device image in a new host object file
  - Run the original linking job with the new wrapped image
New OpenMP Offloading Driver Overview

Compilation

Device Codegen

C/C++ and OpenMP

Clang

Dev IR

LLVM

Dev ASM

Vendor tools

Dev Obj

LLVM

Host IR

Dev Obj

Host

Obj

Linking

Device code embedded directly as a section in host object

Device linking complexity handled in a single stage

Host Obj

Linker Wrapper

Exe

libomptarget (host, plugins)
Embedding OpenMP Offloading Code

C/C++ and OpenMP

Host IR

Device Object

Active Toolchains

Host Sections

.llvm.offloading

omp_offloading_entries

Section Table

Section Contents

@.llvm.embedded.object = private constant [N x i8] c"...", section ".llvm.offloading"

Contains kernels and globals to register
OpenMP Offloading Linker Wrapper

Device Linking

Links every extracted object with a compatible triple & architecture

Vendor Linker

Wrappe

Exe
Benefits

• Offloading binaries behave like host binaries
  – Static libraries and relocatable linking works as expected
• Fewer stages required to create an offloading program
• Much simpler driver code
• Fully functional LTO on the device `-offload-lto`
  – Greatly improves performance on some applications
  – The OpenMPOpt pass greatly benefits from whole program visibility
    • (See Optimizing OpenMP GPU Execution in LLVM @ LLVM Dev2021)
• Will be the default method for OpenMP Offloading very soon!
Future Work & Interoperability
Extending the New Driver

• The new driver can be adapted for CUDA / HIP as well
  – Change code-generation to support the offloading sections
  – Implement a wrapper for CUDA / HIP code
• Allows for redistributable device code (RDC) support in Clang
• The linker wrapper will link all compatible object files
• Allows for OpenMP to call CUDA code and vice-versa
  – Needs additional code for full interoperability
Embedding CUDA & OpenMP Offloading Code

C/C++/CUDA and OpenMP

Active Toolchains

New CUDA CodeGen

Host IR

Device Object

Host Object

Section Table

Host Sections

@.llvm.embedded.object = private constant [N x i8] c"...", section ".llvm.offloading"

Section Contents

<triple, arch and kind>

<Bitcode or Cubin>

Contains kernels and globals to register for both
Offloading Linker Wrapper

* Device Images will be distinct

- Linker Input
- Extract
- Linker Input
- Device Linking
  - Vendor Linker
  - Cuda Obj
  - OMP Obj
  - Dev Obj
- Wrapped Linker job
- Host Image
- Host Image
- OpenMP Wrapper
  + -lcudart -l...
- Cuda Wrapper
  + -lcudart -l...
- Exe
Generic Offloading Libraries

- Create a static library with code for every offloading target
  - Allow more compatible architectures to be linked
- No longer need to specially compile & link device bitcode libraries

Section Contents

- `<nvptx64 sm_80>`
- `<nvptx64 sm_70>`
- `<nvptx64 any>`
- `<amdgcn gfx908>`
- `<...>`
Linker Wrapper in the Linker

- Currently we rely on Clang to call the linker wrapper with the appropriate arguments
- Prevents offloading code from being truly agnostic
- Embed the linker wrapper functionality inside a linker plugin or LLD
Application Experiences
MiniMDock

- Protein-ligand docking mini-application
- A call to an external function prevented a crucial optimization
- Device-side LTO allows us to see the whole program

Figure Generated by Mathialakan Thavappiragasam
Thermo4PFM

- Library to evaluate alloy compositions in Phase-Field models
- Application built with static libraries
- Large amount of files with device code

![Figure Generated by Jean-Luc Fattebert](image-url)
OpenMC

- Monte-Carlo particle transport application
- Needed a CMake Unity build to get reasonable performance
- Device-side LTO gives the same performance and compiles several times faster

The data for the figure was generated and generously provided by John Tramm.
MiniQMC

• Quantum Monte-Carlo mini-application
• Made heavy use of static libraries
  – Can now compile without a CMake workaround
• No performance difference with and without LTO
Conclusion & Closing Thoughts
Conclusion & Closing thoughts

• The new driver greatly improves the usability of OpenMP Offloading in LLVM
  – Allows interoperability
  – Multiple devices embedded in the same binary
• Device-side LTO gives real-world applications significant performance increases
• A unified offloading Toolchain is possible
Questions?
Current Drawbacks

• Currently still relies on Clang to call the linker wrapper
• Cannot embed device code without host LLVM IR
  – Need a new phase to perform an objcopy
• Cannot properly handle incremental compilation
  – e.g. clang foo.c -S