HZDR

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Published under CC BY-SA 4.0 DOI:10.5281/zenodo.4021877

Adding CUDA® Support to Cling:JIT Compile to GPUs

S. Ehrig[1], A. Huebl[1,2], A. Naumann[3] and V. Vassilev[3]

[1] Helmholtz-Zentrum Dresden – Rossendorf

[2] Lawrence Berkeley National Laboratory

[3] CERN

2020 Virtual LLVM Developers' Meeting

October 6th-8th 2020

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

HELMHOLTZ ZENTRUM
DRESDEN ROSSENDORF

Introduction

CASUS
CENTER FOR ADVANCED
SYSTEMS UNDERSTANDING

DRESDEN concept

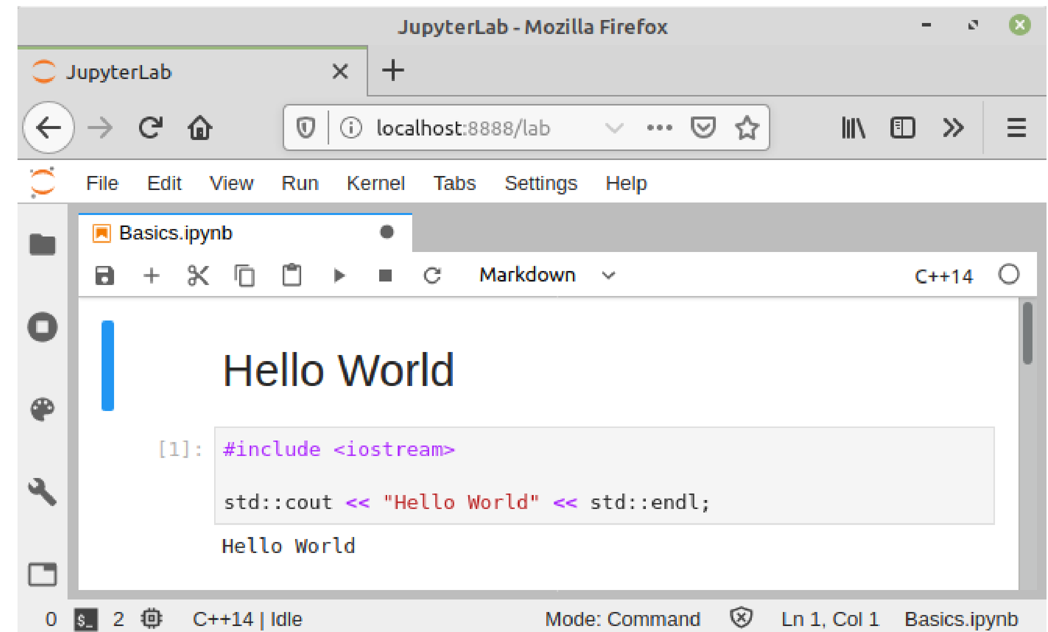
Using Cling

```
simeon@ELMN3:~$ cling

****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ #include <iostream>
[cling]$ std::cout << "Hello Cling" << std::endl;
Hello Cling
[cling]$
```

```cpp
#include "cling/Interpreter/Interpreter.h"

int main(int argc, char *argv){
      auto cling = cling::Interpreter(argc, argv);
      return 0;
}
```



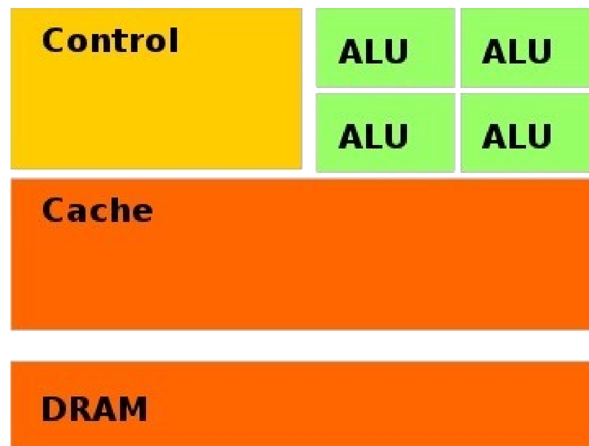
Properties

- Read-Eval-Print loop principle
- Does not interpret → the code is JIT compiled
- Fully compatible to existing libraries
  - Can include header files, load unmodified shared libraries and JIT compile C++ source code
- Modifications on syntax and semantic of C++
  - No main() function → everything in global space
  - Missing semicolon at the end of the statement will print the return value
  - Just allowed in the Cling terminal interface or Jupyter Notebook

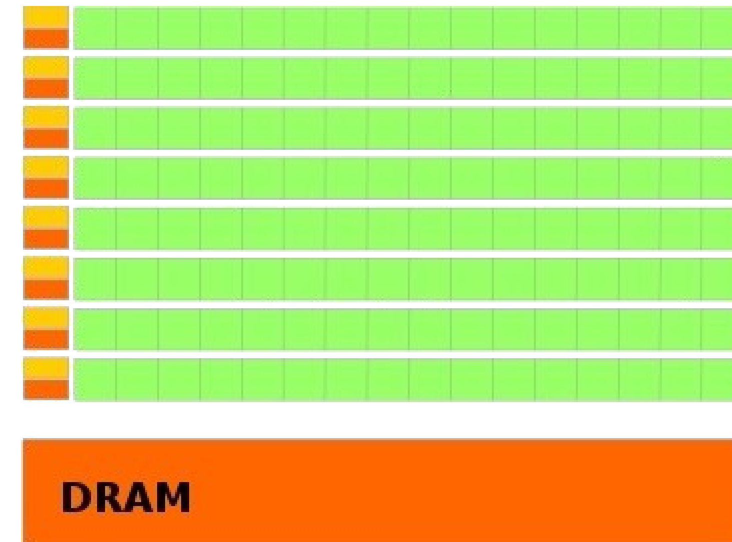
CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

CPU/GPU Model

CPU



GPU

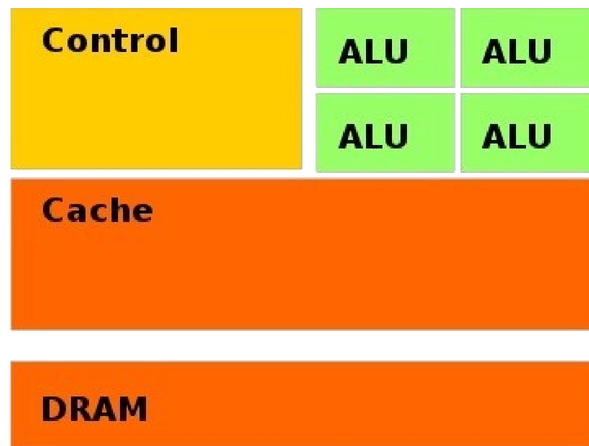


Sources: Nvidia. *CUDA Reference Guide*

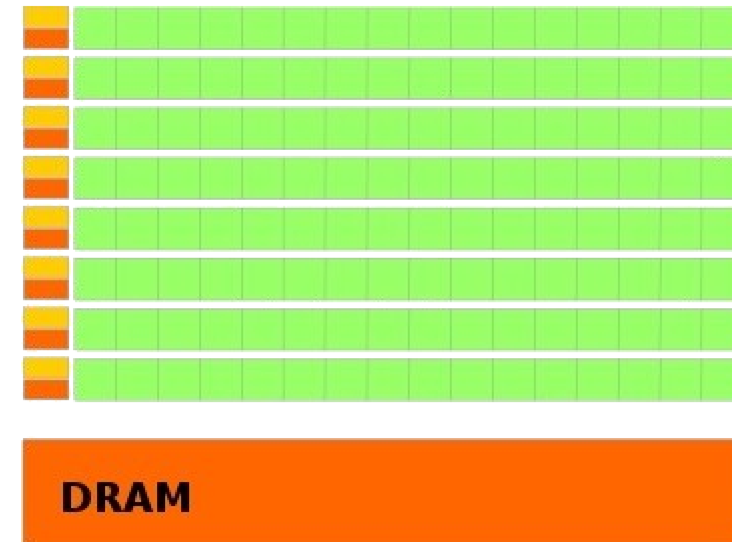
CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

CPU/GPU Model

CPU



GPU



- Why GPU: Better performance for certain algorithms
- Why CUDA: existing algorithms and widest distribution

Sources: Nvidia. *CUDA Reference Guide*

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

HZDR

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

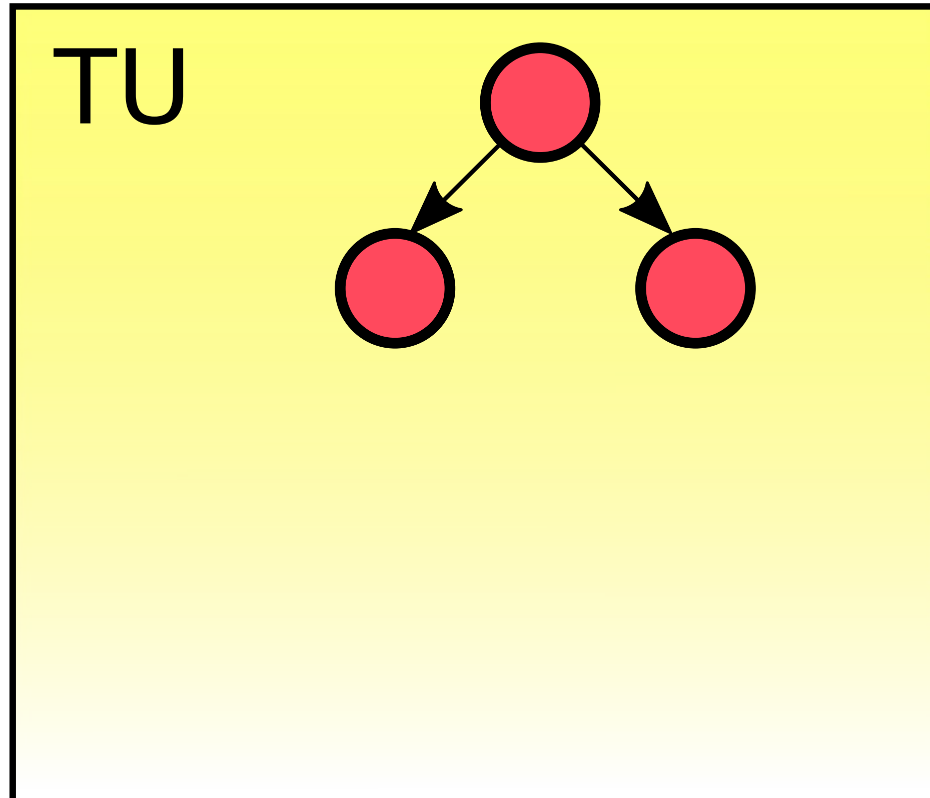
Basic concept

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

Extendable application flow

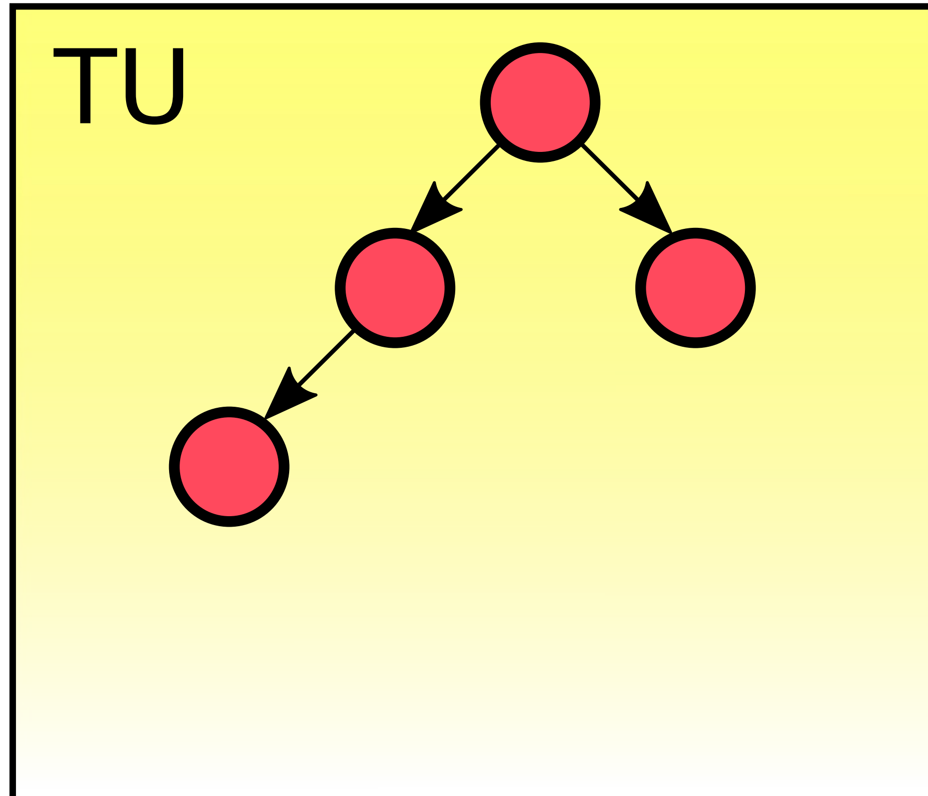
Extendable application flow



Transaction 1
(initial state)

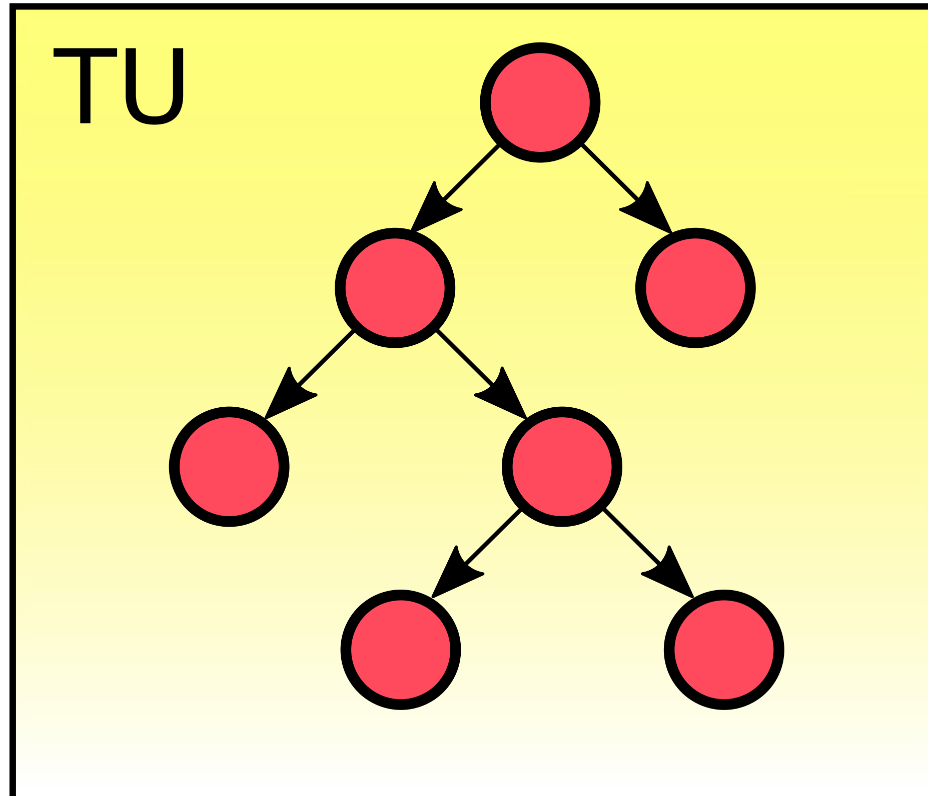
CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Extendable application flow



CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Extendable application flow



Transaction 1
(initial state)

Transaction 2
int i = 3;

Transaction 3
i = i + 3;

Creating a single transaction

Input

Metaparser

Parser

AST-Transformer

Code Generator

Executor

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction

Input

foo()

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ int foo() { return 3;}
[cling]$ foo()
```

Class references:
cling::UserInterface

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction

Input

Metaparser

```
void __cling_Un1Qu32(void* vpClingValue)
{
   foo();
}
```

Tasks of the Metaparser
- Transforms source code
- Detects meta commands
  - e.g.: .L libz.so
  - Linking the shared library z

Class references:
cling::Metaprocessor
cling::utils::getWrapPoint

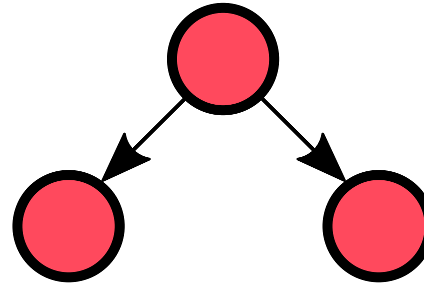
CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction

Input

Metaparser

Parser



Properties of the Parser

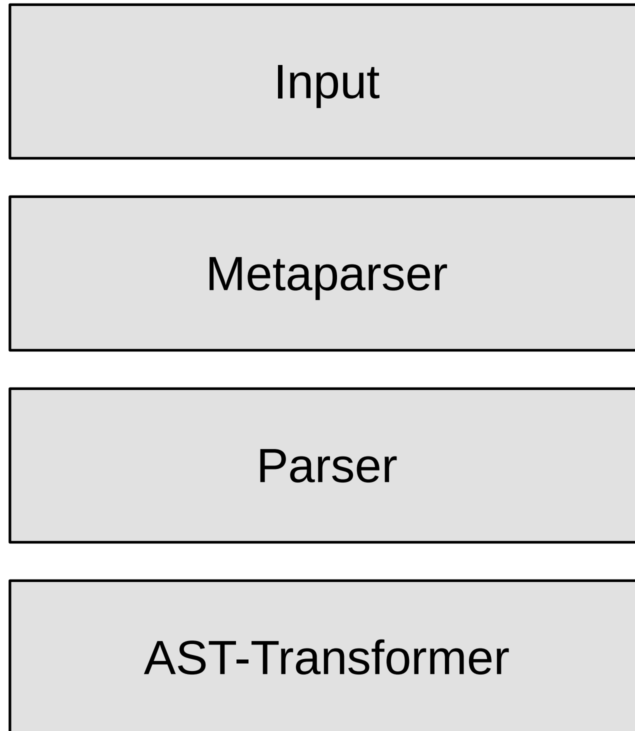
- Non-modified Clang parser
- Needs valid C++ code

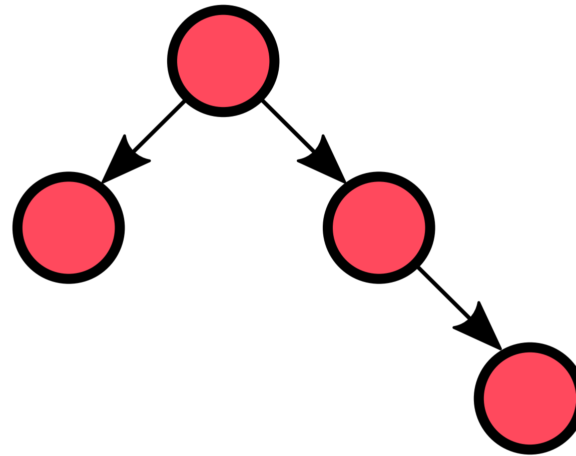
Class references:

cling::IncrementalParser
clang::Parser
clang::ASTConsumer

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction





Tasks of the AST-Transformer

- Enables functionality
  - e.g. CUDA device kernel inliner
- Adds error protection
  - e.g. nullptr access
- Adds cling specific features
  - Shadow namespaces for redefinition

Class references:
cling::ASTTransformer
llvm::legacy::PassManager

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction

Input

Metaparser

Parser

AST-Transformer

Code Generator

```
push    rbp
mov     rbp, rsp
sub     rsp, 8
mov     QWORD PTR [rbp-8], rdi
call    foo()
nop
leave
ret
```

Class references:
  cling::IncrementalJIT
  llvm::orc

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Creating a single transaction

Input

Metaparser

Parser

AST-Transformer

Code Generator

Executor

foo()
(int) 3

```
****************** CLING ******************
* Type C++ code and press enter to run it *
*             Type .q to exit             *
*******************************************
[cling]$ int foo() { return 3;}
[cling]$ foo()
(int) 3
[cling]$
```

Class references:
cling::IncrementalExecutor

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Challenges

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

Challenges

1) Is interactive CUDA C++ possible?

- The driver API allows it, but we want to use the runtime API
- Answered with many experiments with modified LLVM IR and prototypes

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Challenges

1) Is interactive CUDA C++ possible?
   - The driver API allows it, but we want to use the runtime API
   - Answered with many experiments with modified LLVM IR and prototypes
2) How does Cling understand CUDA C++?
   - CUDA C++ is not valid C/C++ → e.g. foo<<<1,1>>>();
   - Google‘s GPUCC project solved the problem for the compiler pipeline → only needed to be activated in Cling
   - Metaparser does not use the Clang parser

Sources: Google. gpucc: *An Open-Source GPGPU Compiler*

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Challenges

1) Is interactive CUDA C++ possible?
   - The driver API allows it, but we want to use the runtime API
   - Answered with many experiments with modified LLVM IR and prototypes
2) How does Cling understand CUDA C++?
   - CUDA C++ is not valid C/C++ → e.g. `foo<<<1,1>>>();`
   - Google's GPUCC project solved the problem for the compiler pipeline → only needed to be activated in Cling
   - Metaparser does not use the Clang parser
3) How to integrate the device pipeline?
   - Cling was not designed for a second compiler pipeline
   - Solved a lot of different implementation tasks

Sources: Google. gpucc: *An Open-Source GPGPU Compiler*

General Problems

- CUDA is proprietary
  - In general, the documentation is good …
  - … but some details are not documented → black box testing

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

General Problems

- CUDA is proprietary
  - In general, the documentation is good …
  - … but some details are not documented → black box testing
- Documentation
  - The whole software stack containing Cling, Clang and LLVM is really complex and I had to learn a lot
  - The LLVM documentation is really good
  - The Clang documentation was okay
  - The Cling documentation is rudimentary and there are no other similar projects

General Problems

- CUDA is proprietary
  - In general, the documentation is good …
  - … but some details are not documented → black box testing
- Documentation
  - The whole software stack containing Cling, Clang and LLVM is really complex and I had to learn a lot
  - The LLVM documentation is really good
  - The Clang documentation was okay
  - The Cling documentation is rudimentary and there are no other similar projects
- The CUDA Runtime API was not used interactively until now
  - No experience
  - Some workarounds necessary

HZDR

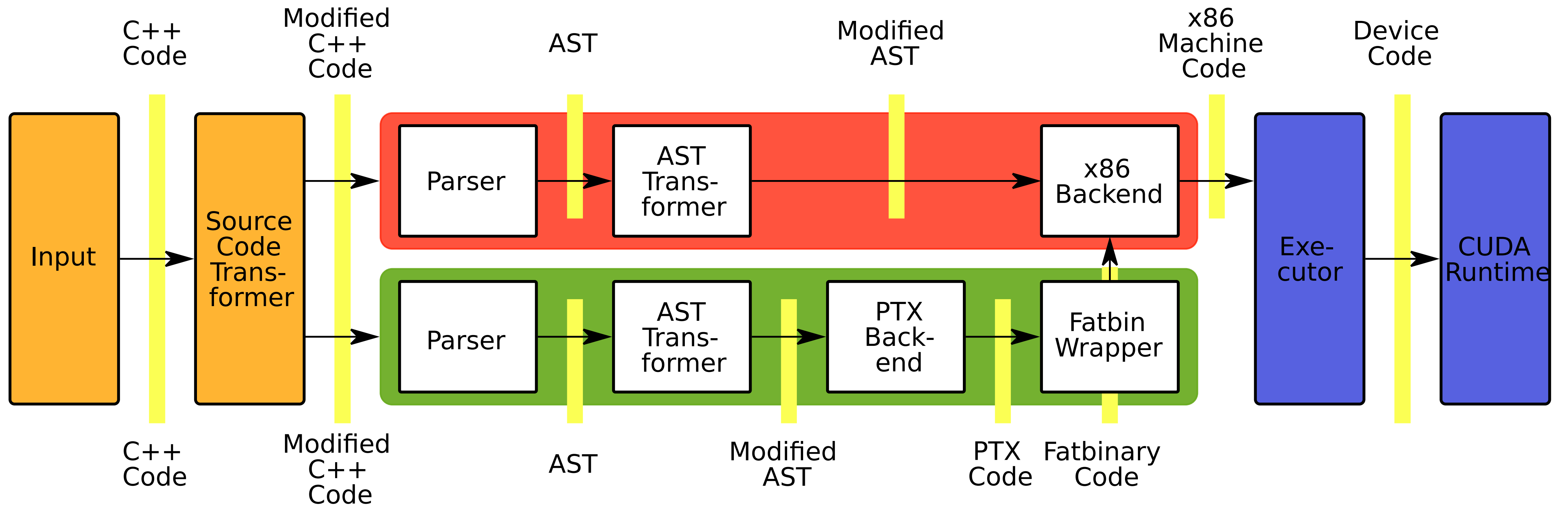
HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Implementation

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

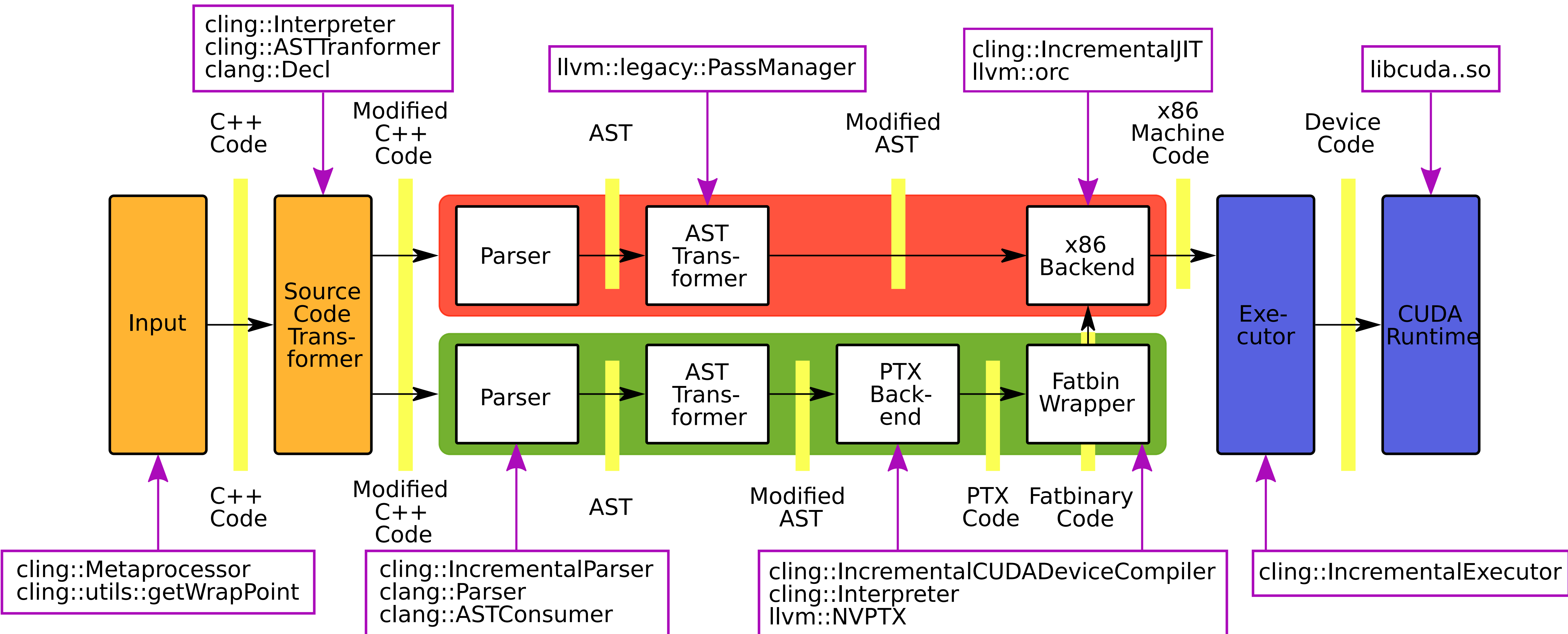
General Implementation



Versions:
- Cling 0.7
- Clang/LLVM 5.0

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

General Implementation



Detail Problem: Metaparser + CUDA

- Problem
  - The Metaparser is completely self-written and parses the “interactive” C++ semantic and the meta commands of Cling
  - The semantic of C++ is complex, the Cling extension makes it even more complex and the CUDA extension too
  - A lot of implementation work is necessary to cover all cases
- Solution
  - Still looking for an optimum solution
  - The most important cases are covered
  - Raw input mode as workaround
- Possible improvements
  - Modifying the Clang parser to handle the “interactive” C++ semantic of Cling

Function references:
cling::utils::getWrapPoint

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Detail Problem: Catching errors

- Problem
  - The interpreter runtime and the user code use the same process and memory space. If a segmentation fault occurs in the user code, the entire interpreter crashes.
- Solution
  - Catch the errors with code analysis before the code is executed.
  - Current solution is not generally applicable
    - e.g. Segmentation faults via indirect pointers

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Detail Problem: Updating the Clang/LLVM base

- Problems
  - Each new Clang/LLVM version supports new CUDA versions, C++ features and has a lot of bug fixes especially with respect to CUDA.
  - The C++ API is not stable and changes continuously. The JIT backend is also continuously developed further.
  - Cling requires a patched version of Clang/LLVM.
  - Updating the Clang specific patches causes a lot of work.
- Possible Solution
  - RFC for simple Clang REPL by Vassil Vassilev (August 2020)
  - Move as many REPL specific patches as possible upstream to Clang

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

What is still missing

- Some C++ and CUDA statements, although supported by Clang 5.0 on CUDA 8.0
  - e.g. CUDA `__constant__` memory
  - and CUDA global `__device__` memory
- Not all Cling features work with CUDA yet
  - e.g. redefinition of kernels via namespace shadowing
- Metaparser does not detect all valid CUDA C++ statements
- Error catching needs to be improved

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

HZDR

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

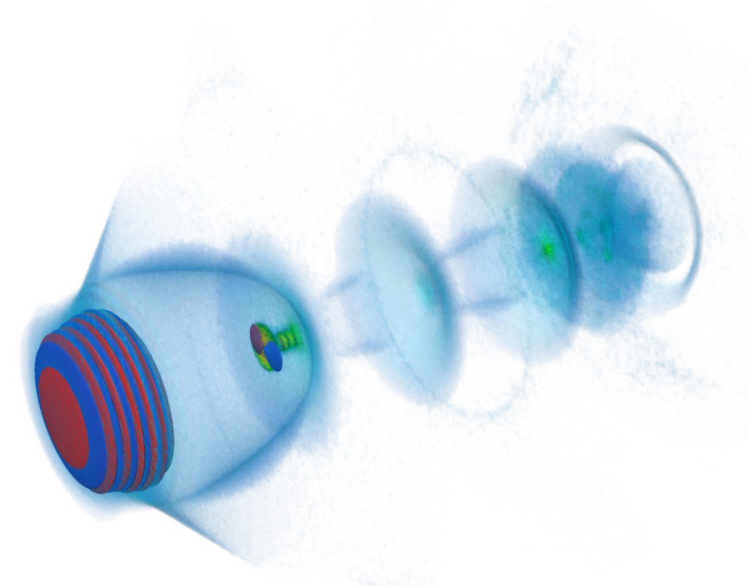
Application Areas

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

Application areas

- Cling was initially developed for **large data analysis** in HEP physics
- Big, **interactive simulation** with GPUs
- **Teaching** GPU programming
- Easing **development** and debugging



https://github.com/alpaka-group/alpaka

https://github.com/ComputationalRadiationPhysics/picongpu/

HZDR

HELMHOLTZ ZENTRUM DRESDEN ROSSENDORF

Summary

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING

DRESDEN concept

Summary

- First interactive C++ JIT compiler for the CUDA runtime API
- Added a dual compiler instance concept to Cling, which can be used for other GPU APIs (AMD, Intel)
- Most features already upstream in cling master
- Interactive CUDA C++ in Jupyter Notebook enables new areas of application
  - Data analysis in notebooks with GPUs
  - Big, interactive simulations with GPUs
  - Teaching GPU programming
  - Easing development and debugging

Versions:
Cling 0.7
Clang/LLVM 5.0

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR

Detail Problem: Clang CUDA expected a completed TU

- Problem
  - How does CUDA register kernels? No official documentation.
  - The Compiler generates the __cuda_module_ctor and __cuda_module_dtor functions which register and unregister the kernels and register the functions in the global constructor and destructor.
  - Cling creates the functions for each transaction. But Cling is lazy and only translates the first occurrence of __cuda_module_ctor into machine code and reuses it for each transaction. So you can only register one kernel in each cling instance.
- Solution
  - Make the function names __cuda_module_ctor and __cuda_module_dtor unique.

Class references:
UnqiueCUDACtorDtorName

Detail Problem: Embedding the Fatbin Generator

- Problem
  - The LLVM IR code of the device compiler pipeline is translated into Nvidia PTX code (a kind of assembler) and embedded in a fatbinary file (struct with meta data and ptx code).
  - Compared to the PTX code, the fatbin struct is not officially specified. Only Nvidia's external fatbin tool is available for embedding PTX code in the fatbin struct.
- Solution
  - Reimplementation of the fatbin tool based on a header file from the CUDA SDK in "llvm-project-cxxjit"
  - Thanks to Hal Finkel

Class references:
cling::IncrementalCUDADeviceCompiler

CASUS CENTER FOR ADVANCED SYSTEMS UNDERSTANDING DRESDEN concept HZDR