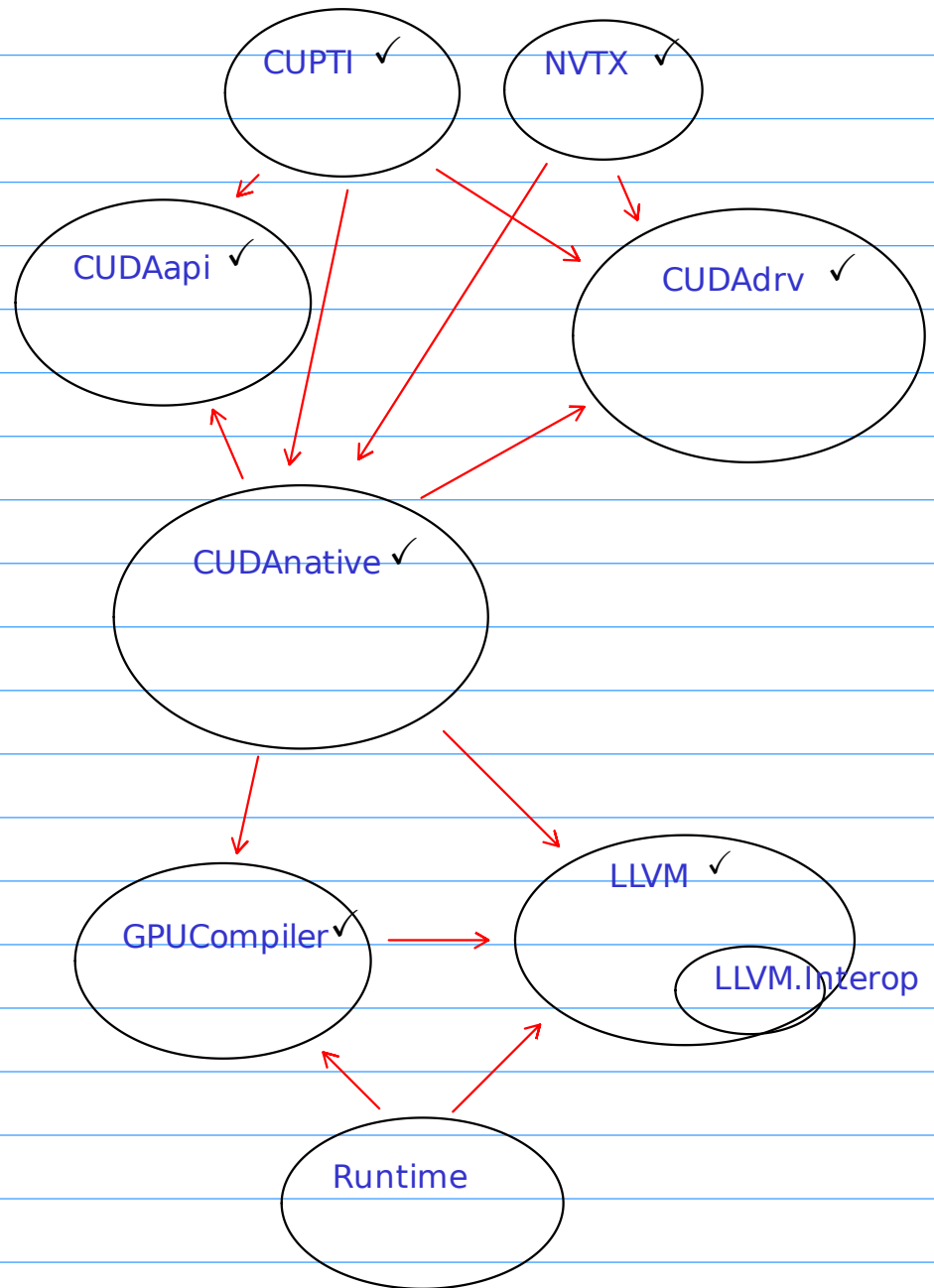


CUDAnative.jl

CUDAnative.jl

CUPTI: CUDA Profiling Tools Interface
NVTX: NVIDIA Tools Extension SDK

GPUのperf counter群を扱うhigh-level API?
annotationして超高級printf debugみたいな?



module 相関図

Why LLVM.jl

The high level Julia IR is accessible with the `code_lowered` and `code_typed` reflection functions, and can be modified with generated functions.

Low-level LLVM IR can be inspected by invoking `code_llvm` and injected via the `llvmcall` metaprogramming interface. Machine code is accessible through `code_native` and can be inserted indirectly as inline assembly in LLVM IR.

LLVM IR への interface は string based

例えばpointerを指定してloadする関数をstring-basedで書いたらその関数はstring操作で溢れて読めたもんじゃないし各pointer typeを個別に扱ってoptimizeするようにはめに陥るだろう

```
julia> using LLVM
julia> mod = LLVM.Module("llvmcall")
; ModuleID = 'llvmcall'
source_filename = "llvmcall"
julia> eltyp = LLVM.convert(LLVMType,Int64)
i64
julia> pram_typs = [LLVM.PointerType(eltyp)]
1-element Array{LLVM.PointerType,1}:
i64*
julia> ft = LLVM.FunctionType(eltyp, pram_typs)
i64 (i64*)
```

```
julia> LLVM.Builder() do builder
    bb = LLVM.BasicBlock(f, "entry")
    LLVM.position!(builder, bb)
    ptr = LLVM.parameters(f)[1]
    val = LLVM.load!(builder, ptr)
    LLVM.ret!(builder, val)
end
ret i64 %1
```

```
julia> f
```

```
define i64 @load(i64*) {
entry:
    %1 = load i64, i64* %0
    ret i64 %1
}
```

LLVM.jl

lib/

@apicall(:LLVMなんたらかんたら , ...) C function?

Clang.jlで生成?

wrap/wrap.jl

llvm-configを走らせてheaderを生成したりする?

src

almost except LLVM.jl are LLVM API wrappers

- LLVM.jl
- analysis.jl
- base.jl
- bitcode.jl
- buffer.jl
- core
- core.jl
- datalayout.jl
- debuginfo.jl
- deprecated.jl
- execution.jl
- init.jl
- interop
- interop.jl
- ir.jl
- irbuilder.jl
- linker.jl
- moduleprovider.jl
- pass.jl
- passmanager.jl
- passregistry.jl
- support.jl
- target.jl
- targetmachine.jl
- transform.jl
- types.jl
- util

base: Define @apicall

```
libllvm_paths = filter(Libdl.dllist()) do lib
    occursin("LLVM", basename(lib))
end
```

filter ideom

new{T,S} return something{T,S} type object

GPUCompiler.jl

@kwdef typedef
automatically defines a keyword-based constructor

ptx.jl hide_trap!
pass removes calls to `trap` and replaces them with inline assembly

CUDAnative.jl

CUDAnative.jl のすること:
CUDAのconfigurationをチェック
device/*.jl を include
CUDACompilerTarget, CUDACompilerJob を定義
*.jl を include
initialization
__init__(), __configure__(), __runtime_init__()

device/tools.jl Create MDNodes
device/pointer.jl Pointers with address space information, AS module, DevicePtr...
Generate LLVM IR for memory operation
device/array.jl Contiguous on-device arrays
device/cuda.jl Include device/cuda/*.jl
device/llvm.jl wrappers for LLVM-specific functionality: trap(), assume()

?bitcast

init.jl CUDA Initialization and Context Management
context, context!, device!, device_reset!
CUDAnative.attaskswitch, CUDAnative.atcontextswitch
compatibility.jl Julia, CUDA and LLVM version compatibility
bindeps.jl CUDA discovery and dependency

cuoti/CUPTI.jl Wrappers for CUPTI and NVTX
nvtx/NVTX.jl

execution.jl @cuda
exceptions.jl device-side exceptions
reflection.jl @device_code_*
array.jl CUDA host array type for testing purposes

deprecated.jl

device/cuda/*.jl

memory_shared.jl

@cuStaticSharedMem(T::Type, dims) -> CuDeviceArray{T,AS.Shared}

@cuDynamicSharedMem(T::Type, dims, offset::Integer=0) -> CuDeviceArray{T,AS.S

These 2 call _shmem @generated function which creates llvm function call so to returns a pointer to shared memory.

indexing.jl

threadIdx, blockDim, blockIdx, gridDim, warpsize

llvm.nvvm.read.ptx.sreg.tid.x などと呼ぶ generated function で

llvm.nvvm.read.ptx.sreg.warpsize を ccall

ATM, values are based on V100

synchronization.jl

sync_threads, sync_warp,

sync_threads_count, sync_threads_and, sync_threads_or

threadfence, threadfence_block, threadfence_system

llvm.nvvm.* function を ccall

warp_vote.jl

vote_all, vote_any, vote_ballot

@asmcallでPTXのvote insnを呼び出し

warp_shuffle.jl

llvm.nvvm.shfl.sync.\$mode.\$typ intrinsicをccallする関数群を生成

output.j

@cuprintf

Print a formatted string in device context on the host standard output.

@cuprint, @cuprintln

Print a textual representation of args to standard output from the GPU.

\$(map(esc, args)...)

asserstion.jl

@cuassert

@assert cond [text]

[continued]

memory_dynamic.jl
malloc

atomics.jl
##LLVM
@generated function llvm_atomic_op(::Val{binop}, ptr::DevicePtr{T,A}, val::T)
 where {binop, T, A}
 arithmetic operations on integers using LLVM instructions
@generated function llvm_atomic_cas(ptr::DevicePtr{T,A}, cmp::T, val::T)
 where {T, A}
##NVVM
@llvm.nvvm.atomic.load.add.*
@llvm.nvvm.atomic.load.inc.*
@llvm.nvvm.atomic.load.dec.*
##Julia
atomic_*(!())
Higher level interface
@atomic a[l] = op(a[l], val)
@atomic a[l] ...= val

misc.jl
clock, nanosleep
llvm.nvvm.read.ptx.sreg.clock64 and nanosleep.u32 insn

wmma.jl
Warp Matrix Multiply Accumulate ???

" The libdevice library is a collection of NVVM bitcode functions that implement common functions for NVIDIA GPU devices, including math primitives and bit-manipulation functions. These functions are optimized for particular GPU architectures, and are intended to be linked with an NVVM IR module during compilation to PTX."

math.jl

" The libcudadevrt library is a collection of PTX bitcode functions that implement parts of the CUDA API for execution on the device, such as device synchronization primitives, dynamic kernel APIs, etc."

libcudadevrt_common.jl, libvudadevrt.jl : cuda_device_runtime_api

cooperative_groups.jl
this_grid, sync_grid

dynamic_parallelism.jl
launch, synchronize

CUDAdrv.jl

CUDAdrv.jl のすること

pointer.jl が essential functionality

CuPtr{T}, PtrOrCuPtr{T}

各種 wrapper を include する

Do init that can't be done at module load time

deferred initialization

initialization

__configure__() which is essentially ccall of cuInit

low-level wrappers

libcuda_common.jl

error.jl CuError, CUDAdrv.initializer(f::Function)

libcuda.jl wrapper gened from cuda.h

libcuda_aliases.jl Remove _v2 postfix

high-level wrappers

version.jl version() and release() of CUDA

devices.jl CuDevice, name, totalmem, attribute,...

context.jl CuContext,...

context/primary.jl CuPrimaryContext

stream.jl CuStream,...

memory.jl Mem, Mem.DeviceBuffer, Mem.Device,
Mem.alloc, Mem.[un]register, Mem.free, copy...

module.jl include module/jit.jl encode/decode JIT options

CuModule, CuModuleFile

include module/function.jl CuFunction

include module/global.jl CuGlobal, get, set

include module/linker.jl CuLink,...

events.jl CuEvent, record, synchronize, elapsed

execution.jl CuDim, cudacall

cudacall calls launch() with convert_argument

profile.jl profiling with Nsight/NVTX or with external profiler

occupancy.jl active_blocks, occupancy, launch_configuration

deprecated.jl

cuXXXはCUDA_driver, cudaXXXはCUDA_runtime

CUDAapi.jl

CUDAのinstall状況を知る+お役立ちmodule

CUDAapi.jl includes

util.jl	@enum_without_prefix
discovery.jl	find_cuda_library, find_cuda_binary, find_toolkit, parse_toolkit_version, find_libdevice, find_libcudadevrt
availability.jl	has_cuda, has_cuda_gpu, usable_cuda_gpus call find_toolkit and ccall cuDeviceGetCount
complex.jl	Define cuda complex types
library_types.jl	Other cuda data types
call.jl	@runtime_ccall, decode_ccall_function, @checked ccall extention wrappers