OpenMP as GPU Kernel Language

Progress Report

Johannes Doerfert <jdoerfert@llnl.gov>
Tom Scogland <scogland1@llnl.gov>

Prepared by LLNL under Contract DE-AC52-07NA27344.

Offload Style

```
int i = blockId.x;
if (i >= N) return;
int j = threadId.x;
if (j >= M) return;
body1(i, j);
__syncthreads();
body2(i, j);
}
```

Offload Style

```
int i = blockId.x;
if (i >= N) return;
int j = threadId.x;
if (j >= M) return;
body1(i, j);
__syncthreads();
body2(i, j);
}
```

```
#pragma omp target teams distribute
for (int i = ...) {
  #pragma omp parallel for
  for (int j = ...)
    body1(i, j);
  #pragma omp parallel for
  for (int j = ...)
    body2(i, j);
}
```

Offload Style

```
int i = blockId.x;
if (i >= N) return;
int j = threadId.x;
if (j >= M) return;
body1(i, j);
__syncthreads();
body2(i, j);
}
```

```
#pragma omp target loop
for (int i = ...) {
  #pragma omp loop
  for (int j = ...)
    body1(i, j);
  #pragma omp loop
  for (int j = ...)
    body2(i, j);
}
```

OpenMP Kernel Language

libompx — Host Wrappers

```
_LIBOMPX_NAMESPACE_BEGIN

extern "C" void *ompx_malloc(size_t size) {
  if (size <= 0)
    return nullptr;
  return omp_target_alloc(size, omp_get_default_device());
}

void *ompx::malloc(size_t size) { return ompx_malloc(size); }

_LIBOMPX_NAMESPACE_END
```

libompx — Device Wrappers

```
/// AMDGCN Implementation
///
///{
#pragma omp begin declare variant match(device = {arch(amdgcn)})

uint32_t ompx_get_thread_num(int Dim = 0) {
  switch(Dim) {
    case 0:
  return __builtin_amdgcn_workitem_id_x();
    case 1:
  return __builtin_amdgcn_workitem_id_y();
    case 2:
  return __builtin_amdgcn_workitem_id_z();
    default:break;
  }
  __builtin_unreachable();
}

...

#pragma omp end declare variant
```

[1/2] CUDA Syntax

```
kern<<<nblocks, nthreads, shmem>>>(a1, a2)
```

```
#pragma omp target teams num_teams(nblocks) thread_limit(nthreads) \
                         ompx_cgroup_dyn_mem(shmem) ompx_kernel
kern(a1, a2)
```

[2/2] CUDA Syntax

```
__device__ void foo();
```

```
void foo();
#pragma omp declare target device_type(nohost) to(foo)
```

[2/2] CUDA Syntax

```
__device__ void foo();
```

```
[[ompx::declare_target(device_type(nohost))]] void foo();
```

CUDA via LLVM/OpenMP

LLVM/OpenMP as Target Independent Runtime Layer

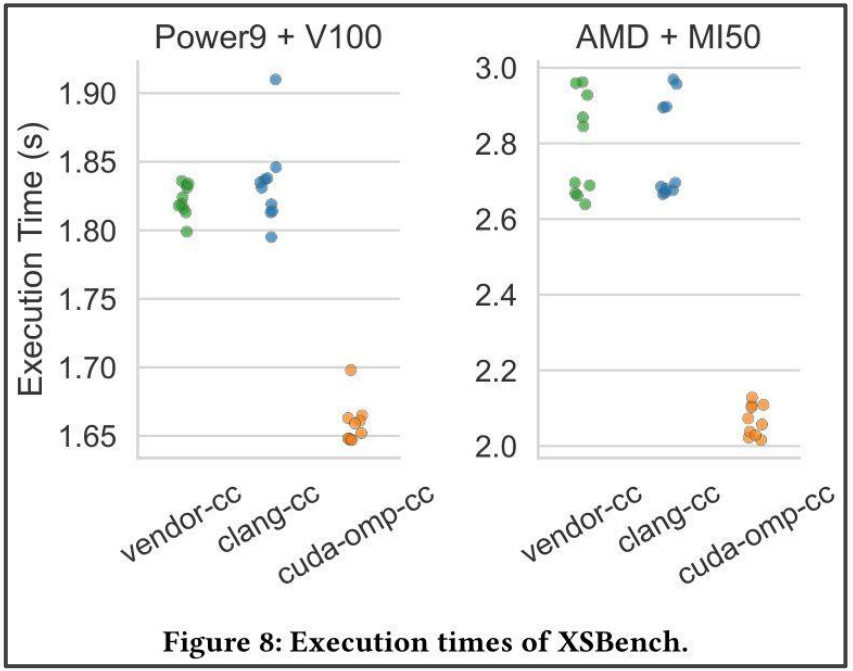


Figure 8: Execution times of XSBench.

Breaking the Vendor Lock — Performance Portable Programming Through OpenMP as Target Independent Runtime Layer (PACT'22)

LLVM/OpenMP as Target Independent Runtime Layer

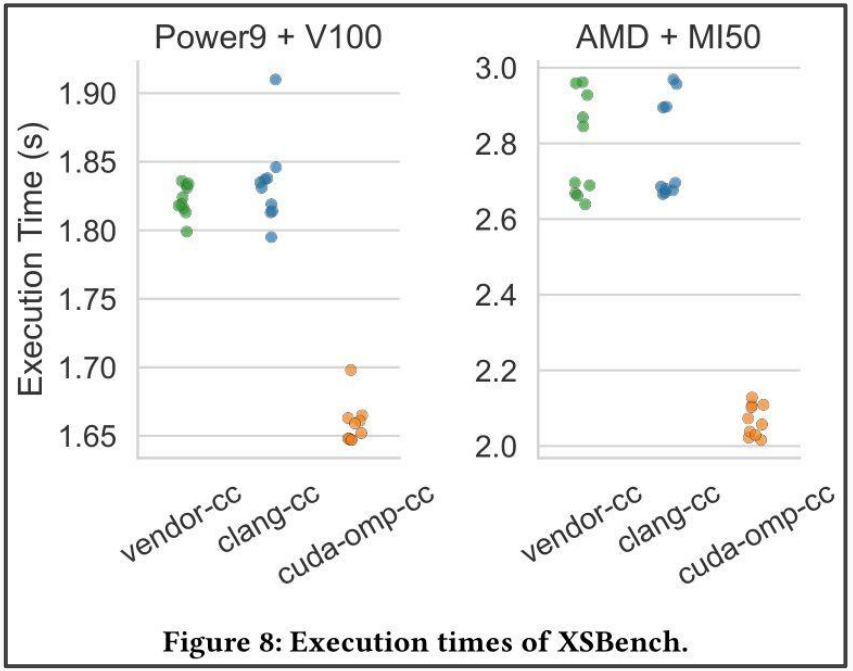


Figure 8: Execution times of XSBench.

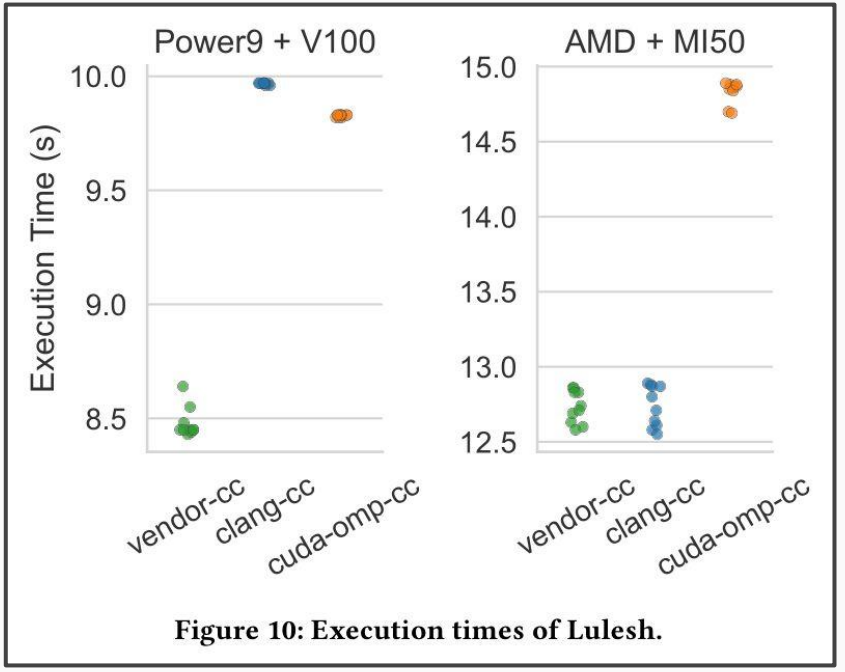


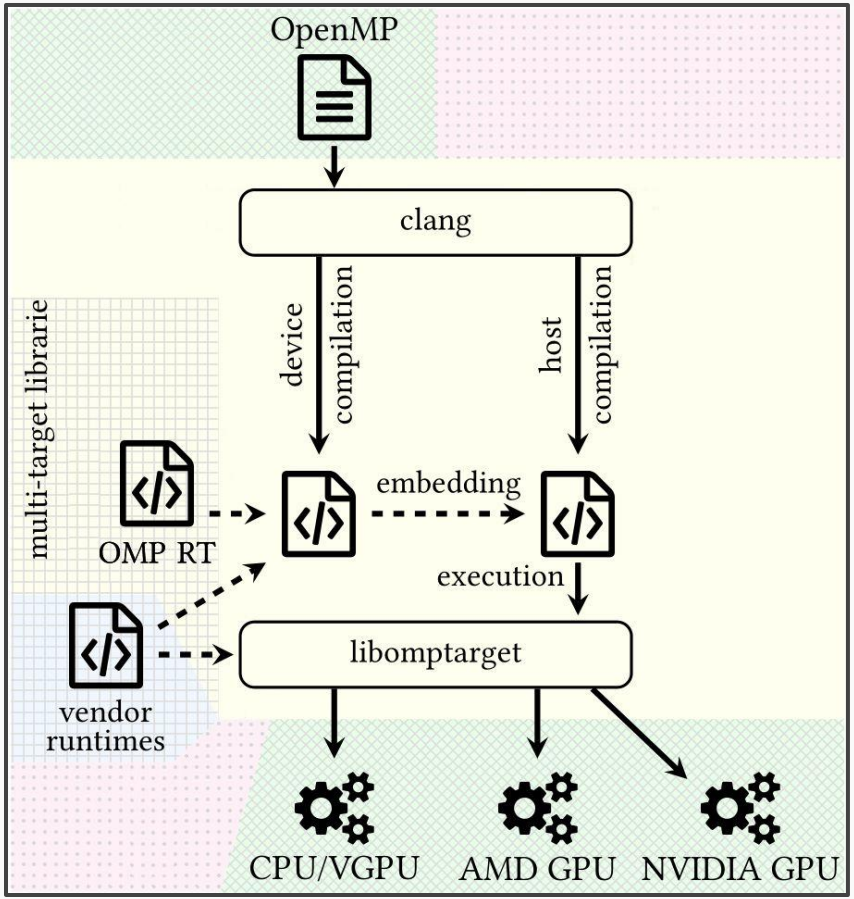
Figure 10: Execution times of Lulesh.

Breaking the Vendor Lock — Performance Portable Programming Through OpenMP as Target Independent Runtime Layer (PACT'22)

Questions?

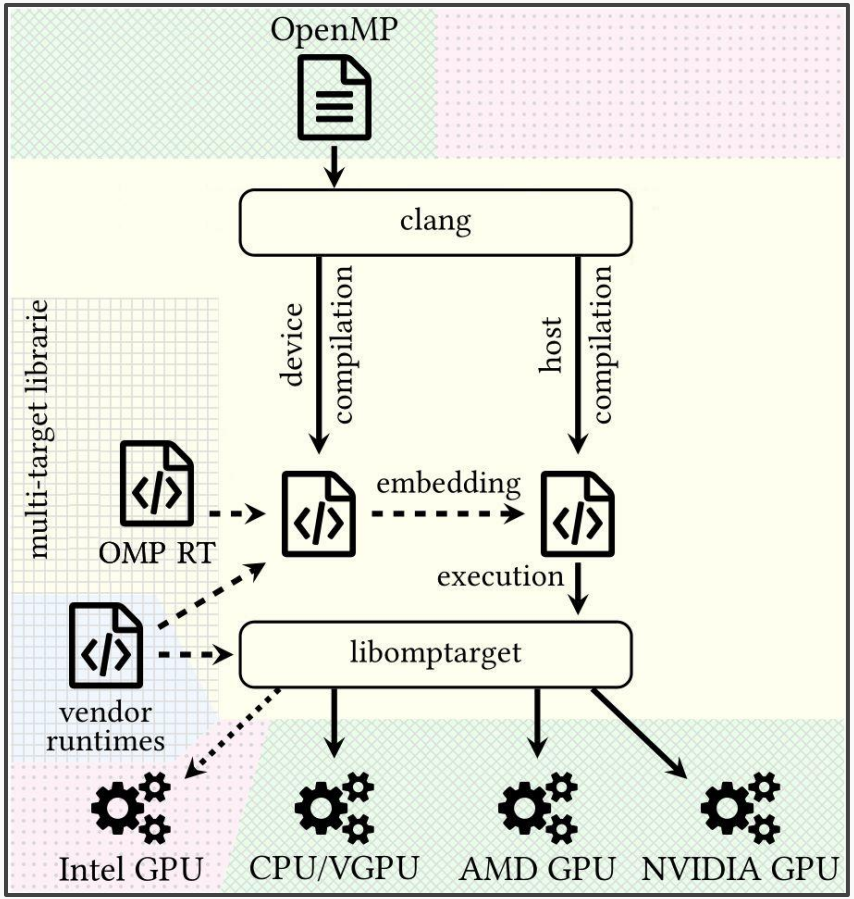
OpenMP as Intermediate Layer

LLVM/OpenMP Target Offloading



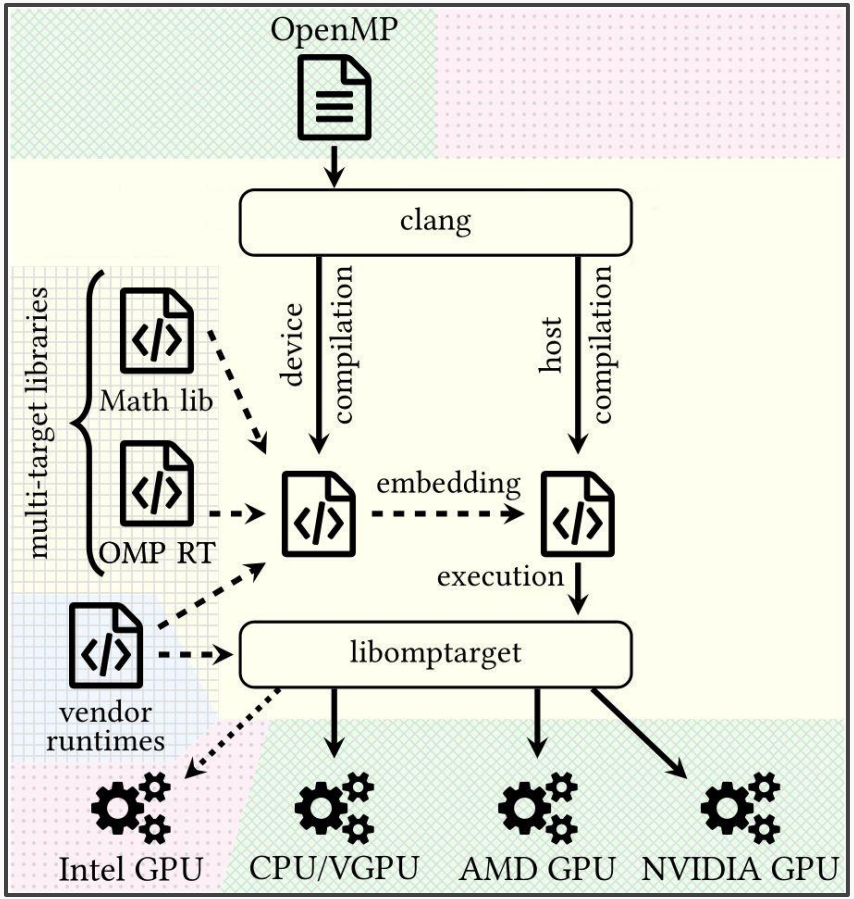
- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs

LLVM/OpenMP Target Offloading



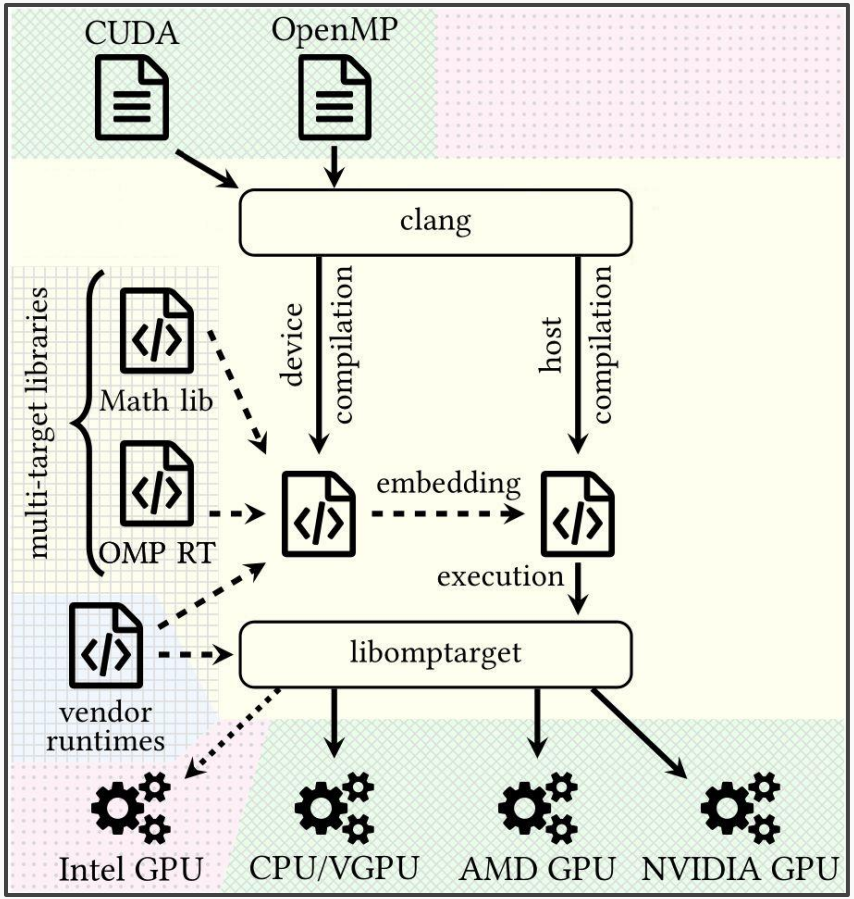
- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP

LLVM/OpenMP Target Offloading + Math Runtimes



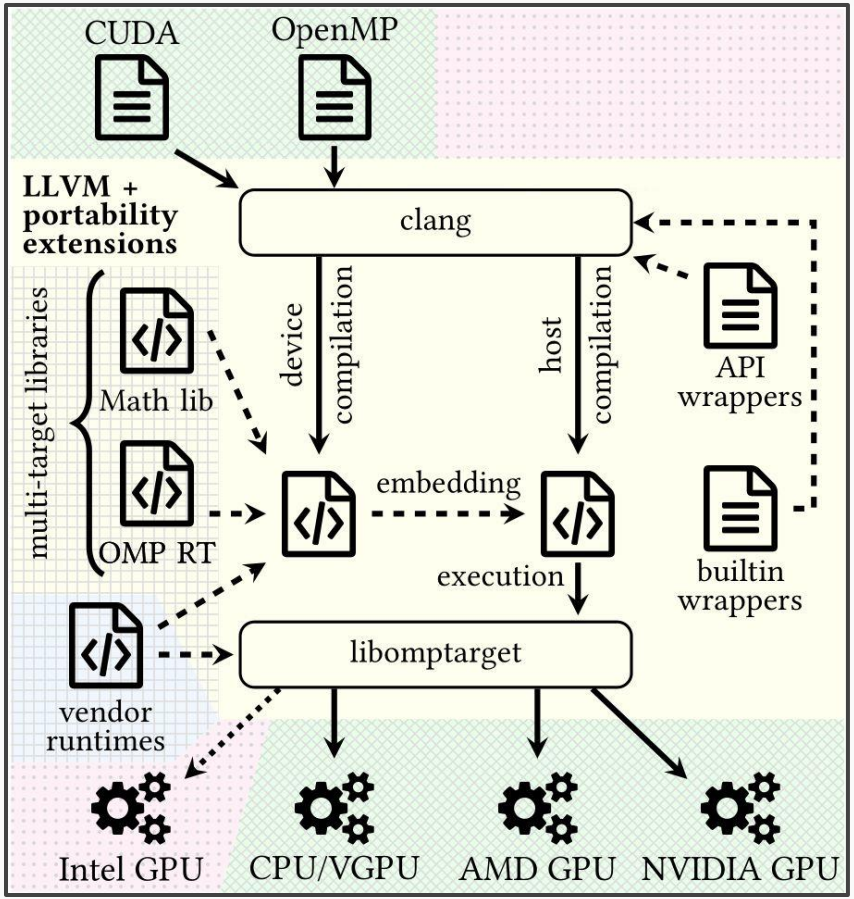
- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP
- Target independent math library (libm.a) for all supported architectures. Defines `sin(...), etc.`

LLVM/OpenMP Target Offloading + CUDA Device Compilation



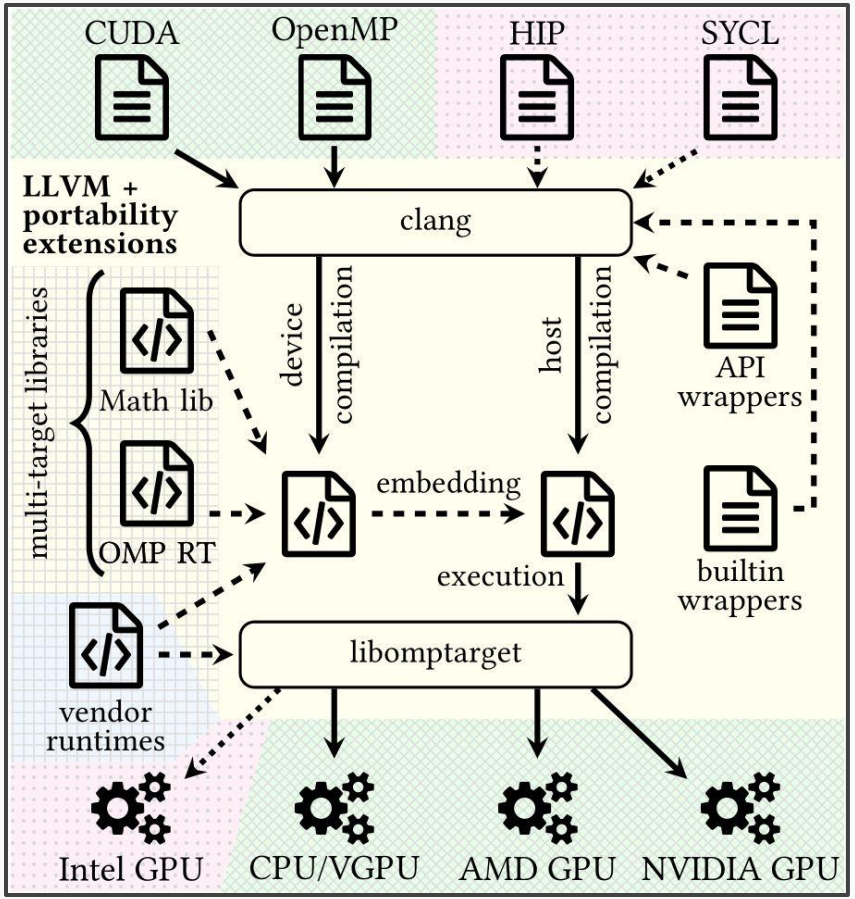
- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP
- Target independent math library (libm.a) for all supported architectures. Defines `sin(...)`, etc.
- CUDA device code interoperability with OpenMP target. Link in CUDA device runtimes e.g., Thrust.

LLVM/OpenMP as Target Independent Runtime Layer (for CUDA)



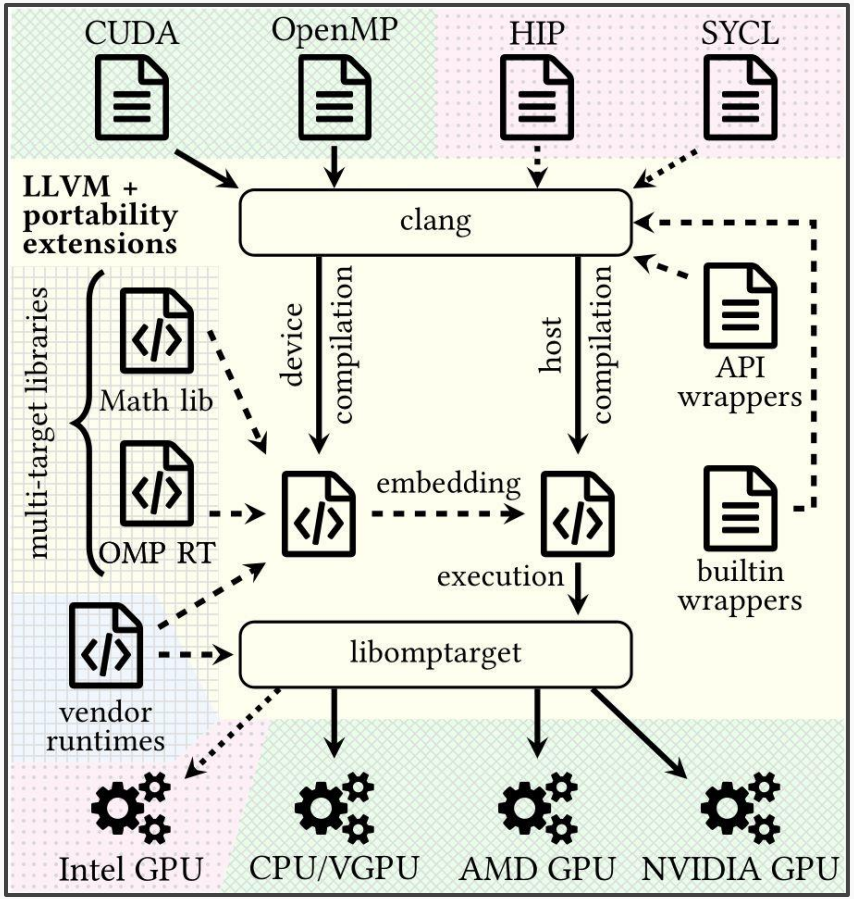
- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP
- Target independent math library (libm.a) for all supported architectures. Defines `sin(...)`, etc.
- CUDA device code interoperability with OpenMP target. Link in CUDA device runtimes e.g., Thrust.
- Define CUDA API and builtins through OpenMP runtime functions. Allow to retarget CUDA codes.

LLVM/OpenMP as Target Independent Runtime Layer (WIP)



- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP
- Target independent math library (libm.a) for all supported architectures. Defines `sin(...)`, etc.
- CUDA device code interoperability with OpenMP target. Link in CUDA device runtimes e.g., Thrust.
- Define CUDA API and builtins through OpenMP runtime functions. Allow to retarget CUDA codes.
- HIP, SYCL, and other languages can be added as needed. Full interoperability and portability.

LLVM/OpenMP as Target Independent Runtime Layer (WIP)



- OpenMP offload code compilation for CPUs, virtual GPU (VGPU), AMD and NVIDIA GPUs
- Intel GPU support is WIP
- Target independent math library (libm.a) for all supported architectures. Defines `sin(...)`, etc.
- CUDA device code interoperability with OpenMP target. Link in CUDA device runtimes e.g., Thrust.
- Define CUDA API and builtins through OpenMP runtime functions. Allow to retarget CUDA codes.
- HIP, SYCL, and other languages can be added as needed. Full interoperability and portability.
- Overall WIP but proof-of-concept is ready and under review (PACT) right now. Parts have been upstreamed (incl. Driver) or are prepared to be.

LLVM/OpenMP as Target Independent Runtime Layer (WIP)

```
Thread 17 "su3" hit Breakpoint 1, k_mat_nn (a=0x[...]8b10, b=0x[...]
      cb20, c=0x[...]cc50, total_sites=256) at ./mat_nn_cuda.hpp:22
22 int myThread = blockDim.x * blockIdx.x + threadIdx.x;
(gdb) bt
#0 k_mat_nn (a=0x[...]8b10, b=0x[...]cb20, c=0x[...]cc50,
      total_sites=256) at ./mat_nn_cuda.hpp:22
#1 0x[...]d6dd in ?? () from /usr/lib64/libffi.so.7
#2 0x[...]9a69 in VGPUTy::VGPUTy()::{lambda()#2}::operator()() ()
   from [...]/lib/libomptarget.rtl.vgpu.so
(gdb) print myThread
$2 = 15
(gdb) next
25 if (mySite < total_sites) {
(gdb) cont
Continuing.

Thread 17 "su3" hit Breakpoint 2, k_mat_nn (a=<opt out>, b=<opt out>,
       c=0x[...]cc50, total_sites=256) at ./mat_nn_cuda.hpp:32
32 CMULSUM(a[mySite].link[j].e[k][m], b[j].e[m][l], cc);
(gdb) next
36 c[mySite].link[j].e[k][l] = cc;
(gdb) print cc
$3 = {real = 1, imag = 0}
```

Host GDB running the SU3 bench CUDA code via the OpenMP layer on the virtual GPU.