Introduction to HIP-Python

NRIS
Norwegian research infrastructure services

Hicham Agueny
Scientific Computing Group
University of Bergen/NRIS

Motivation

- **Python** is **widely used** in ML, data science, scientific computing
- Writing codes with **HIP C++** can be **challenging**
- Making GPU programming accessible to Python Users

- **Need of a python interface to interact with HIP libraries (C++ API)**

Learning outcomes

- ➢ Access the GPU properties from a python interface
- ➢ Transfer data between host and device
- ➢ Call hipBLAS library from a python interface
- ➢ Compile and launch GPU Kernels from a python interface

Examples:
https://github.com/HichamAgueny/HIP-Python_examples

Overview

- ❑ **What is HIP-Python ?**

- ❑ **GPU management**
  - Get the device properties **hip.hipGetDeviceProperties**
  - Get the device attributes **hip.hipDeviceGetAttribute**

- ❑ **Memory management**
  - Allocate GPU memory **hip.hipMalloc()**
  - Free GPU memory **hip.hipFree()**
  - Direct memory copy **hip.hipMemcpy()**

- ❑ **Call HIP libraries from python**
  - Example: hipBLAS

- ❑ **GPU Kernels**
  - HIP RTC API for compiling
  - HIP runtime API functions for launching

What is HIP-Python ?

HIP-Python is a **python wrapper** for:

- **HIP* runtime** API (*HIP-Heterogeneous-Compute Interface for Portability)
- **HIP RunTime Compilation (HIPRTC)** API
- Various **math libraries**, & communication library RCCL
- **Supports** both AMD and NVIDIA GPUs

- **HIP runtime API** provides functions to:
  - Access Device properties
  - Allocate and Free Device memory
  - Transfer Data between Host and Device
  - Launch Device Kernels (Device kernel is a function that is executed on the GPU)

- **HIPRTC API** provides functions to:
  - Create functions and managing GPU programs dynamically
  - Compile the Device Kernels at the runtime (i.e. during the execution of the application)

https://rocm.docs.amd.com/projects/hip-python/en/latest/index.html

Device properties via hip.hipGetDeviceProperties()

The object **hip.hipDeviceProp_t()** **is passed** as an argument to **hip.hipGetDeviceProperties()**

```
from hip import hip

props = hip.hipDeviceProp_t()
hip.hipGetDeviceProperties(props,0)
# Get selected properties

print(f"props.name = {props.name}")
print(f"props.gcnArchName = {props.gcnArchName}")
print(f"props.pciDeviceID = {props.pciDeviceID}")
print(f"props.totalGlobalMem = {props.totalGlobalMem}")
print(f"props.l2CacheSize = {props.l2CacheSize}")
```

Out:

```
props.name = b'AMD Instinct MI250X'
props.gcnArchName = b'gfx90a:sramecc+:xnack-'
props.pciDeviceID = 0
props.totalGlobalMem = 68702699520
props.l2CacheSize = 8388608
```

Device properties via hip.hipGetDeviceProperties()

The object **hip.hipDeviceProp_t() is passed** as an argument to **hip.hipGetDeviceProperties()**

```
from hip import hip

props = hip.hipDeviceProp_t()
hip.hipGetDeviceProperties(props,0)

# Get all the device properties

for attrib in sorted(props.PROPERTIES()):
    print(f"props.{attrib}={getattr(props,attrib)}")
```

https://rocm.docs.amd.com/projects/hip-python/en/latest/index.html

Device properties via hip.hipDeviceGetAttribute

The outcome of **hip.hipDeviceAttribute_t()** **is passed** as an argument to **hip.hipDeviceGetAttribute()**

```
from hip import hip

for attrib in (
hip.hipDeviceAttribute_t.hipDeviceAttributeMaxBlockDimX,
hip.hipDeviceAttribute_t.hipDeviceAttributeMaxBlockDimY,
hip.hipDeviceAttribute_t.hipDeviceAttributeMaxBlockDimZ,
hip.hipDeviceAttribute_t.hipDeviceAttributeWarpSize,
hip.hipDeviceAttribute_t.hipDeviceAttributeMaxThreadsPerBlock,
):
result_attr = hip.hipDeviceGetAttribute(attrib,device_id)
print(f"{attrib.name}: {result_attr[1]}")
```

Out:

```
hipDeviceAttributeMaxBlockDimX: 1024
hipDeviceAttributeMaxBlockDimY: 1024
hipDeviceAttributeMaxBlockDimZ: 1024
hipDeviceAttributeWarpSize: 64
hipDeviceAttributeMaxThreadsPerBlock: 1024
```

https://rocm.docs.amd.com/projects/hip-python/en/latest/python_api/hip.html#hip.hip.hipDeviceAttribute_t

Memory management

- **Allocate Device memory**

Ptr = hip.hipMalloc(*unsigned long sizeBytes*)

- **Ptr**: Pointer to the memory to be allocated on the GPU
- **sizeBytes**: Data size in Bytes.

- **Copy data from src (in) to dst (out)**

hip.hipMemcpy(*dst*, *src*, *unsigned long sizeBytes*, *kind*)

- **dst**: This is the destination where the data will be copied to.
- **src**: This is the source from where the data will be copied.
- **kind**: Direction of transfer HostToDevice or DeviceToHost.

- **Free Device memory**

hip.hipFree(Ptr)

- **Ptr: Pointer to the memory to be freed**

Memory management - Example 2

```python
# Import some modules
import numpy as np
from hip import hip

# Generate random 1D-array
N = 10 #length
host_data = np.random.rand(N).astype(np.float32)

# Allocate device memory
num_bytes = N * np.float32().itemsize
device_data = hip.hipMalloc(num_bytes)

# Copy data from host to device
hip.hipMemcpy(device_data, host_data, num_bytes, hip.hipMemcpyKind.hipMemcpyHostToDevice)

# Copy data from device to host
host_data_b = np.empty_like(host_data)
hip.hipMemcpy(host_data_b, device_data, num_bytes, hip.hipMemcpyKind.hipMemcpyDeviceToHost)



# Free device memory
hip.hipFree(device_data)
```

Calling HIP library from python

List of HIP libraries that can be called from a python interface through HIP-Python API.

- **hip.rccl**: Collective communication library (e.g. broadcast, reduce, ...) for multiple GPUs
- **hip.hiprtc**: HIP RunTime Compilation for compiling GPU-kernels (HIP C++) at runtime

➢ Math libraries

- **hip.hiprand**: Random number generation library optimized for AMD GPU
- **hip.hipfft**: Fast Fourier Transform library optimized for AMD GPU
- **hip.hipsparse**: Sparse matrix operations library (sparse matrix-vector and matrix-matrix operations) optimized for AMD GPU
- **hip.hipsolver**: Dense linear algebra operations (solving linear systems) optimized for AMD GPU
- **hip.hipblas**: Basic Linear Algebra Subprograms (e.g. vector addition, matrix multiplication) optimized for AMD GPU

Calling HipBLAS - Example 3

Calling a **hipBLAS function** from python interface consists of 3 steps

- Initiate hipBLAS
- Call a hipBLAS function to do computation on the GPU
- Destroy the library handle

Example: **hipblasSasum** is a function designed to compute the **sum of absolute values** of the elements in a single-precision.

- **Initiate hipBlas**

Handle = hipblas.hipblasCreate()

- **Call a hipblasSasum function**

hipblas.hipblasSasum(*handle, int n, x, int incx, result*)

- **Destroy handle**

hipblas.hipblasDestroy(handle)

- **handle**: This is the handle to the hipBLAS library context
- **n**: The number of elements in the input vector x
- **x**: The pointer to the input vector x storing the **n** elements
- **incx**: The increment between each element of x
  - If incx is 1, the function will use each element of x.
  - If incx is 2, the function will use every second element of x.
- **result**: The pointer to the variable where the result will be stored

Take-away

- ❑ HIP-Python provides python users with a simple way to interact with HIP libraries and API runtimes.

- ❑ HIP libraries include:

  - **Math libraries**: hipBLAS, hipRAND, hipFFT, hipSPARSE, hipSOLVER.

  - **HIP Runtime API**: Allocate & free GPU memory, Copy data between Host and Device and launch GPU kernels.

  - **HIPRTC API**: Compile GPU kernels written with HIP C++.

- ❑ **Supports** both AMD and NVIDIA GPUs

Hands-on Examples

Hands-on examples

1-Download examples:

$ git clone https://github.com/HichamAgueny/HIP-Python_examples.git

2-Launch an interactive session

$ salloc -A project_465001310 -t 00:30:00 -p dev-g -N 1 --gpus 1

3-Load modules

module load LUMI/24.03 partition/G

module load cray-python/3.11.7

4-Source the virtual env. where hip-python and numpy are installed

$ source /project/project_465001310/workshop_software/HIP-Python_examples/MyVirtEnv_hip_pyt/bin/activate

5-Run examples

$ cd HIP-Python_examples

$ srun python example1_DeviceProp/deviceProp.py

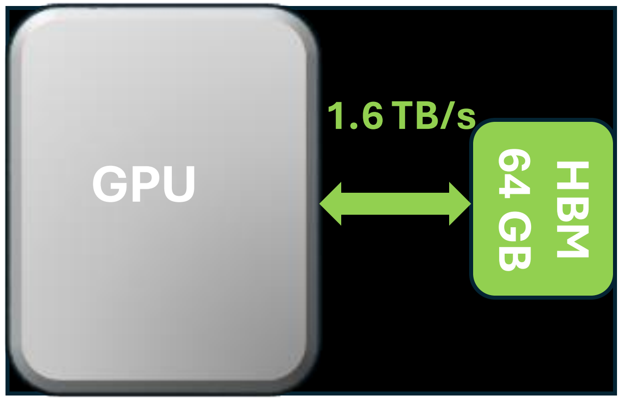
$ srun python example2_MemoryManagemnt/memoryManagemnt.py

$ srun python example3_HipBlas/hipblasSum.py

$ srun python example4_DeviceKernels/deviceKernels.py

GPU Kernel: Creating and compiling a program

Computation on the GPU



Compiling and launching GPU kernels:

➢ **Creating and compiling a program with HIPRTC API**

HIP allows to compile a program at runtime with HIPRTC API.

- Create the program **hiprtc.hiprtcCreateProgram()**
- Compile the program **hiprtc.hiprtcCompileProgram()**

➢ **Launching a program with HIP runtime API**

- Build module from object **hip.hipModuleLoadData()**
- Get the kernel function **hip.hipModuleGetFunction()**
- Specify grid and block dimensions **hip.dim3()**
- Launch the kernel **hip.hipModuleLaunchKernel()**

GPU Kernel: Creating and compiling a program Example 3

- HIP allows to compile a program at runtime with HIPRTC API.
- Kernels can be stored as a text string and passed as an argument to hiprtc.hiprtcCreateProgram()
- The output from creating program is passed as input to hiprtc.hiprtcCompileProgram()

HIP C++ Kernel stored in a file named "kernel.hip"

```
extern "C" __global__ void Kernel_test(int n, float factor, float *A)
 {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;

    A[idx] = factor*A[idx];
 }
```

Create program

```
prog =                          # hiprtc program hiprtc.hiprtcCreateProgram(kernel_source.encode(),#
Source code
               b"Kernel_test",      # Name of Kernel
               0,                 # Number of Headers
               [],                # Names of Headers
               [])                # Names of Includes
```

Compile program

```
arch = b'gfx90a'

cflags = [b"--offload-arch="+arch]

hiprtc.hiprtcCompileProgram(prog, len(cflags), cflags)
```

GPU Kernel: Launching a program Example 4

Build module and Get the kernel

```
code_size = hiprtc.hiprtcGetCodeSize(prog)
code = bytearray(code_size)
hiprtc.hiprtcGetCode(prog, code)
module = hip.hipModuleLoadData(code)
kernel = hip.hipModuleGetFunction(module, b"Kernel_test")
```

Specify the block and grid dimensions

```
block = hip.dim3(x=16, y=1, z=1)
grid = hip.dim3(math.ceil(N/block.x))
```

Launch the program

```
hip.hipModuleLaunchKernel(
                    kernel,
                    *grid,
                    *block,
                    sharedMemBytes=0,
                    stream=None,
                    kernelParams=None,
                    extra=(
                            ctypes.c_int(N),
                            ctypes.c_float(factor),
                            device_data,
                    ))
```

References

https://rocm.docs.amd.com/projects/hip-python/en/latest/user_guide/1_usage.html#basic-usage-python

https://github.com/ROCm/hip-python

https://github.com/HichamAgueny/HIP-Python_examples