Advanced Micro Devices (AMD) Capstone Project-Internship

ARYAMAN MISHRA – 19BCE1027

Internship Completion Certificate

AMD

AMD India Private Limited
Plot # 102 & 103,
EPIP Zone, Whitefield.
Bangalore - 560066
Tel: +91 33 23 0000 Fax: 91-80-3325 0555
CIN #U72200KA1997PTC094389

Date: 18 May 2023

INTERNSHIP CERTIFICATE

We hereby certify that **Aryaman Mishra** bearing **Employee Code : 50063932** has been an Co-Op with **AMD India Private Limited** Bangalore, for the period starting **08 August 2022** to **05 May 2023**.

We wish you all the best in your future endeavours.

For AMD India Private Limited

Fathima Farouk
Sr. Director HR Business Partner

HISTORY

- Advanced Micro Devices (AMD) is a global semiconductor company.
- Founded in 1969, with headquarters in Santa Clara, California.
- Leader in high-performance computing, graphics, and visualization technologies.

KEY VALUES

- Empower people with innovative computing technologies.
- Deliver superior performance, energy efficiency, and reliability.
- Transforming industries and pushing the boundaries of what's possible.

Key Products

- Ryzen™ Processors

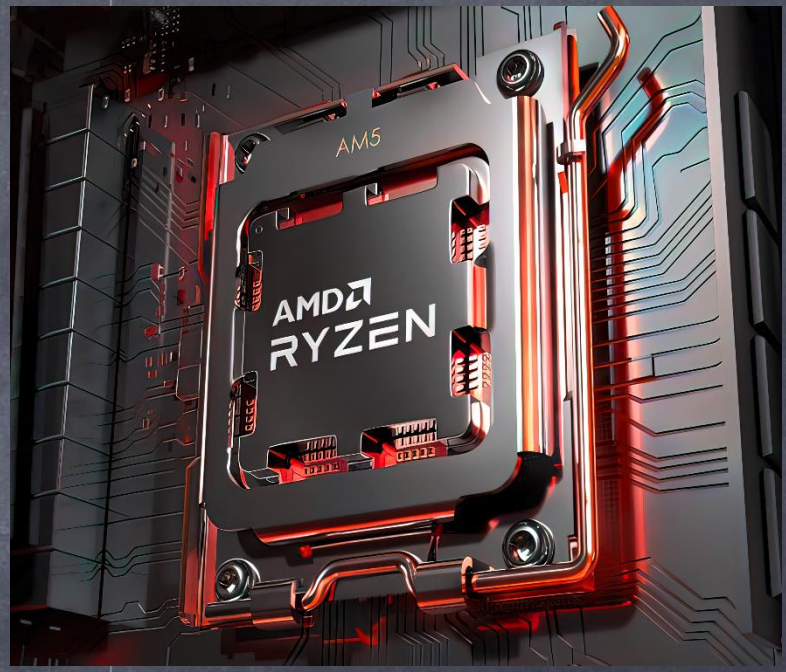
High-performance CPUs for desktop, mobile, and server applications.
Unleash the power of multi-core processing and efficient architectures.

- Radeon™ Graphics Cards

Cutting-edge GPUs for gaming, content creation, and professional workloads.
Provide exceptional visual fidelity, real-time rendering, and AI capabilities.

- EPYC™ Processors

Enterprise-grade server CPUs offering superior performance and security.
Enable efficient data centers and cloud computing solutions.



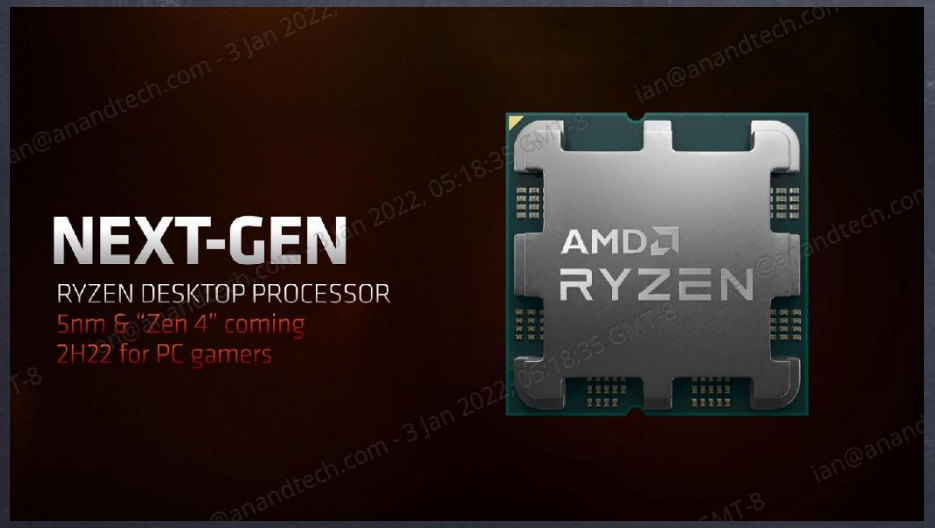
Technological Advancements

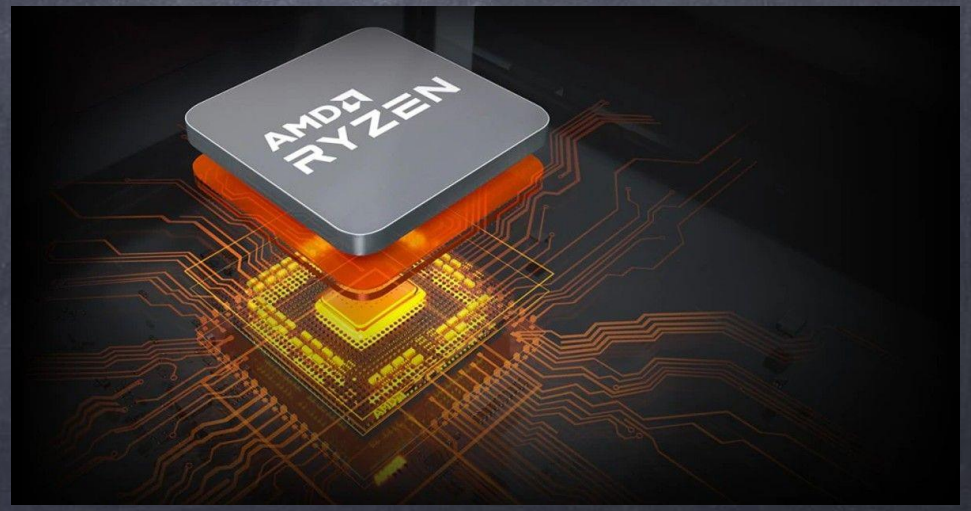
Zen Architecture

- Groundbreaking CPU architecture delivering remarkable performance.
- Enhanced multi-threading, increased core count, and improved power efficiency.
- RDNA Architecture

Next-generation GPU architecture optimizing gaming and visual workloads.

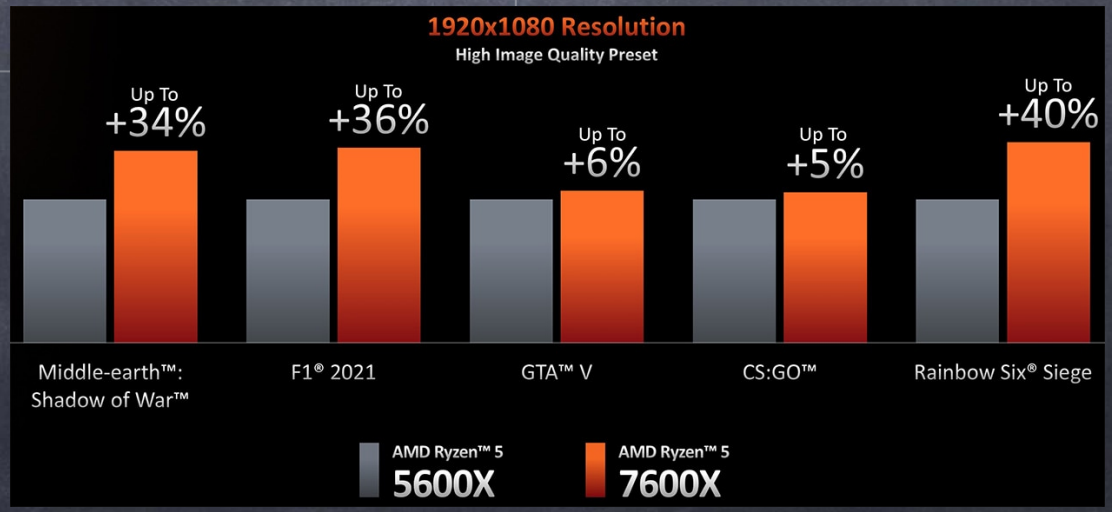
- Advanced features like hardware-accelerated ray tracing and variable rate shading.
- Infinity Fabric
- Advanced data fabric interconnect technology.
- Enables high-bandwidth communication between CPUs, GPUs, and memory.





AMD's Impact

- Gaming and Entertainment
- Unparalleled gaming experiences with smooth frame rates and realistic visuals.
- Powering popular gaming consoles, enhancing virtual reality, and streaming.
- Data Centers and Cloud Computing
- Accelerating data processing, machine learning, and AI workloads.
- Offering scalable solutions for cloud-based services and big data analytics.
- Workstations and Professional Applications
- Enabling professionals to tackle complex tasks with ease.
- Superior performance for content creation, 3D modeling, and scientific simulations.



Industry Recognition

- Widely recognized for technological advancements and innovation.
- AMD products have received numerous awards and positive reviews.
- Trusted by leading technology companies and industry experts.

Future Innovations

- Continuously pushing the boundaries of computing technology.
- Investing in research and development to drive further breakthroughs.
- Collaborating with partners to shape the future of computing.

- AMD is a global leader in high-performance computing and graphics.
- Transforming industries with innovative processors and graphics solutions.
- Empowering the world with next-generation computing technologies.

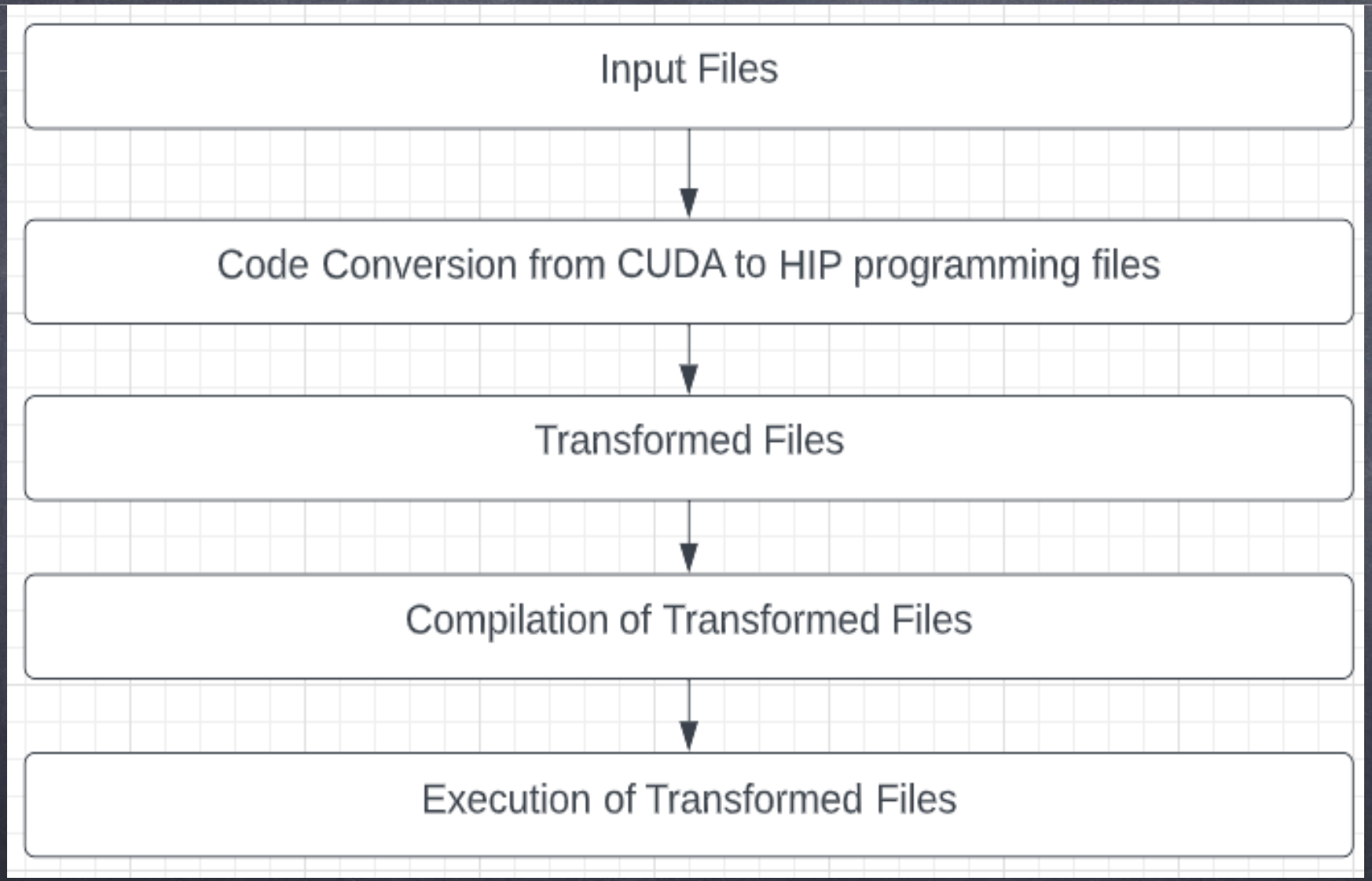
Background

- A GPU, or Graphics Processing Unit, is a specialized type of processor designed for performing complex calculations needed for rendering graphics on a computer screen. GPUs can process large amounts of data in parallel, making them useful for tasks that involve heavy computation, such as machine learning, scientific simulations, and cryptography.
- CUDA, or Compute Unified Device Architecture, is a parallel computing platform and programming model created by NVIDIA for use with their GPUs. It allows developers to use C or C++ to write programs that run on the GPU, taking advantage of the parallel processing power of the GPU to perform calculations faster than they could on a CPU. CUDA has become very popular among researchers and developers working in fields such as artificial intelligence, scientific simulations, and data analysis.
- ROCm, or Radeon Open Compute, is an open-source software platform created by AMD for GPU computing. It provides a set of open-source tools and libraries for developers to use with AMD GPUs, similar to CUDA for NVIDIA GPUs. The ROCm platform is designed to support a variety of programming languages, including C++, Python, and Fortran, and can be used for ROCm, or Radeon Open Compute, is an open-source software platform created by AMD for GPU computing. It provides a set of open-source tools and libraries for developers to use with AMD GPUs, similar to CUDA for NVIDIA GPUs. The ROCm platform is designed to support a variety of programming languages, including C++, Python, and Fortran, and can be used for

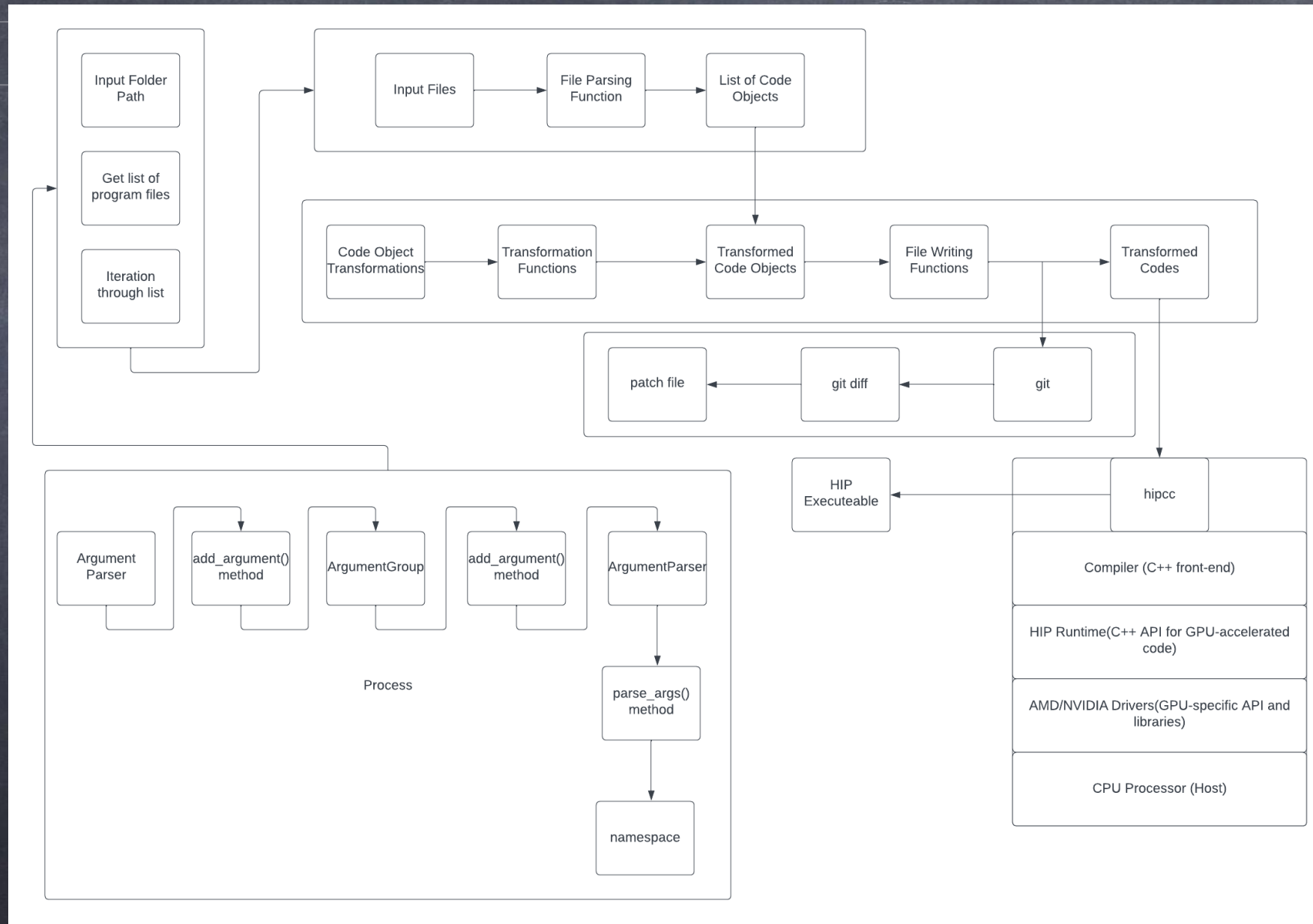
Project TestHIPIFY

- HIP (Heterogeneous-compute Interface for Portability) is an open-source C++ programming interface developed by AMD that allows developers to write code that can be compiled for either AMD or NVIDIA GPUs. HIP includes a set of libraries and tools that provide a similar programming model to CUDA, allowing developers to use familiar CUDA programming concepts to write code that can be run on both AMD and NVIDIA GPUs.
- To convert CUDA code to HIP, tools called hipify-clang is used. This tool is a modified version of the Clang C/C++ compiler that can automatically convert CUDA code to HIP code. The conversion process involves replacing CUDA-specific functions and libraries with their equivalent HIP functions and libraries, so that the resulting code can be compiled for either AMD or NVIDIA GPUs.It’s better alternative which is applied in the project is its Perl script alternative by the name-HIPIFY-Perl.
- Project TestHIPIFY is a program that helps developers to convert their code from one programming language to another. It uses a process called "hipifying" which makes the code conform to certain standards and conventions. This can be helpful when working on a project that involves multiple programming languages or when transitioning from one language to another.
- The program takes in the source code, either in a file or a directory, and transforms it using various rules and patterns to make it compatible with the desired programming language. The transformed code is then outputted to a new file with a new name.

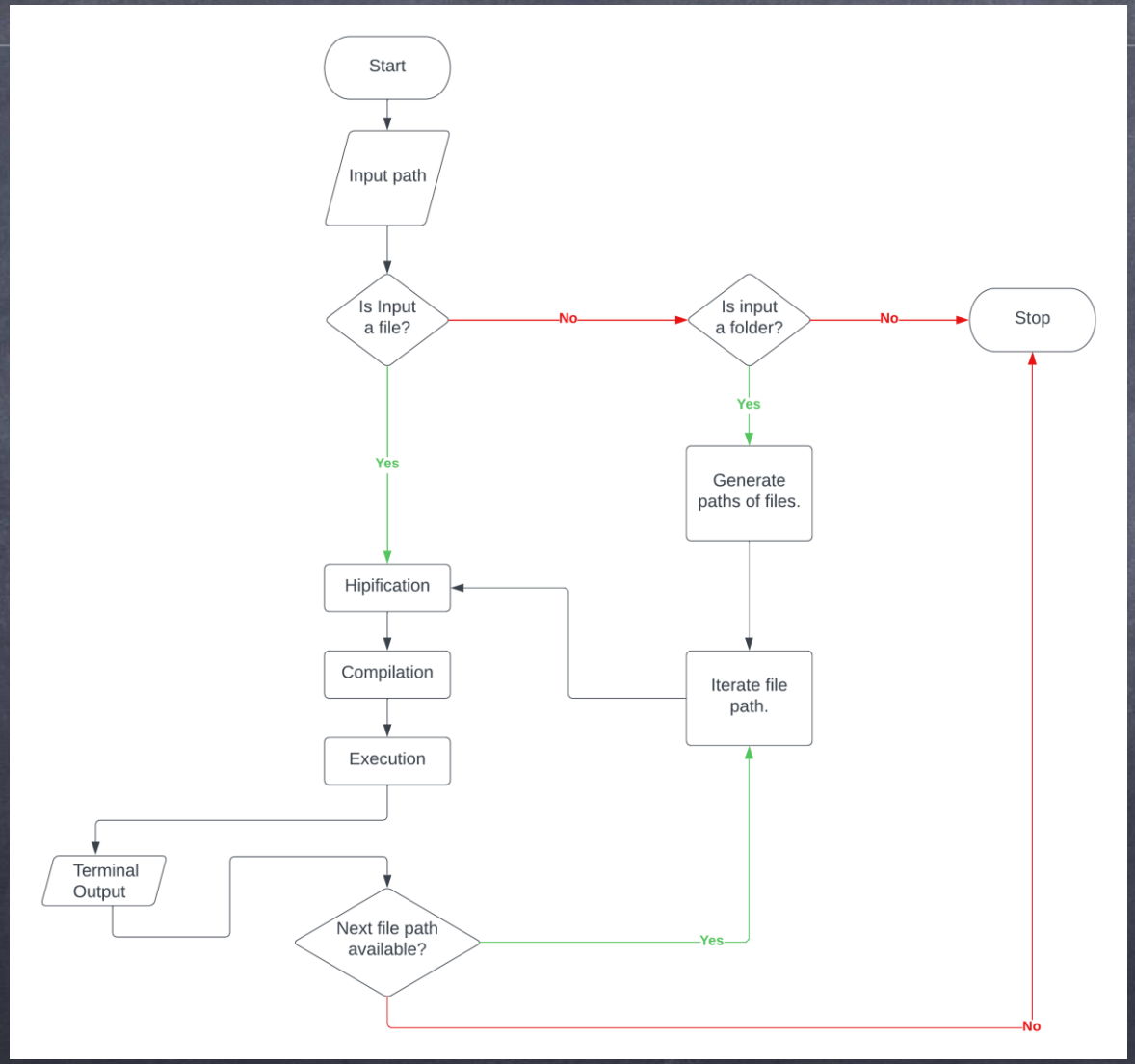
High Level Diagram



Low Level Diagram



Flowchart


Start
Input path
Is Input a file?
No
Is input a folder?
No
Stop
Yes
Yes
Generate paths of files.
Hipification
Compilation
Execution
Iterate file path.
Terminal Output
Next file path available?
Yes
No

Brief Recap

- HIP (Heterogeneous-compute Interface for Portability) is an API and programming model for writing portable and scalable code for heterogeneous systems, such as those consisting of CPUs and GPUs. It was developed by AMD and provides a C++ interface for programming GPU accelerators, similar to Nvidia's CUDA API.
- HIP allows developers to write code once and run it on different GPU architectures, such as those from AMD and Nvidia, without having to maintain separate codebases. This makes it easier to target multiple GPU platforms and reduces the amount of code that needs to be maintained.
- HIP provides a set of high-level abstractions and primitives for GPU programming, including support for parallelism, memory management, and synchronization. It also provides interoperability with other APIs, such as CUDA, OpenCL, and OpenGL, allowing for integration with a wide range of tools and libraries.
- In summary, HIP is an open-source and cross-platform solution for GPU programming that enables developers to write code that is portable, scalable, and interoperable with other APIs.
- TestHIPIFY is a command-line based tool developed in Python Programming language, which allows its users to simply pass relative paths of a CUDA sample to convert using either HIPIFY-Perl or HIPIFY-Clang, compile them with or without using -static-lib flags and finally generate an executable. TestHIPIFY’s repository already contains the updated versions of CUDA samples, for the user’s convenience, allowing them to execute them on AMD devices having other hardware specifications.

Usage

Installation

Setting up dependancies

```
git clone https://github.com/AryamanAMD/testhipify.git
```

```
cd testhipify
```

```
python testhipify.py -s
```

This command would ensure installation of CUDA,GCC,OpenMP and OpenMPI[Installation may take upto 10 minutes].

Usage

How to Process Samples

For a single sample:

```
python testhipify.py -t '[Path to that Sample]'
```

Example:

```
python testhipify.py -t 'src/samples/Samples/0_Introduction/Clock/clock.cu'
```

For all samples:

```
python testhipify.py -a 'src/samples/Samples'
```

CUDA Samples

- The CUDA Samples repository at https://github.com/NVIDIA/cuda-samples is a collection of CUDA code samples and accompanying documentation. The code samples are meant to illustrate the use of various CUDA APIs and demonstrate how to utilize CUDA to perform general purpose computing on GPUs. The code samples are organized into several categories, such as Matrix multiplication, FFT, Monte Carlo, and Image Processing. The samples are designed to run on a CUDA-capable GPU and can be built on Windows, Linux, and Mac OS. The repository also includes a set of tutorials and guides to help users understand the concepts and techniques used in the code samples.
- CUDA Samples are a collection of code examples provided by NVIDIA for learning and using the CUDA platform for parallel computing on GPUs. The samples demonstrate various concepts of parallel programming and algorithms that can be accelerated using GPUs. The CUDA Samples are meant to help developers understand the basic concepts of parallel computing and develop efficient GPU-accelerated applications. The samples are written in C/C++ and cover a wide range of application domains such as image processing, scientific simulations, and machine learning.
- CUDA is a parallel computing platform and API model developed by Nvidia for programming GPUs. The CUDA samples provided by Nvidia are designed to help developers learn how to use CUDA to build high performance, parallel computing applications. The samples cover a wide range of topics, including:
- Basic CUDA concepts and techniques, such as memory transfers and kernel launches

1. Image and video processing
2. Scientific and engineering simulations
3. Machine learning and deep learning
4. Database acceleration

Dependencies

- *CUDA*
- *ROCm Supported GPU*
- *GCC Compiler*
- *ROCm Stack*
- *OpenMP*
- *OpenMPI*

Deployment Tools	ROCm Validation Suite	ROCm Data Center Tool	ROCm SMI			
Libraries	rocBLAS	rocRAND	rocFFT	MIGraphX	MIVisionX	rocPRIM
	rocSOLVER	rocALUTION	rocSPARSE	rocTHRUST	MIOpen	RCCL
Compilers & Tools	Compiler	Profiler	Debugger	hipify	TENSILE	
Programming Models	OpenMP® API	HIP API	OpenCL™			
Drivers/Runtimes	RedHat®, SLES & Ubuntu® Device Drivers and Run-Time					

Features

This code is a Python script that uses the argparse library to parse command-line arguments. The script sets up an argparse.ArgumentParser object with a description of the script's purpose, which is to "HIPIFY Cuda Samples." The script then adds several arguments that the user can specify when running the script, such as:

- "-a" or "--all" argument runs the hipify-perl tool on all samples within a specified folder.
- "-b" or "--generate" argument generates .hip files from .cu files.
- "-c" or "--compile1" argument compiles .hip files generated by the hipify-perl tool.
- "-d" or "--compile2" argument compiles .hip files with static libraries.
- "-e" or "--execute" argument executes the compiled .out files.
- "-f" or "--generate_all" argument generates all .hip files from .cu files.
- "-g" or "--compile1_all" argument compiles all .hip files generated by the hipify-perl tool.
- "-i" or "--compile2_all" argument compiles all .hip files with static libraries.
- "-j" or "--execute_all" argument executes all compiled .out files.
- "-k" or "--parenthesis_check" argument removes last parts from cu.hip files which are out of bounds.
- "-l" or "--parenthesis_check_all" argument removes all last parts from cu.hip files which are out of bounds.
- **"-n" or "--nvidia_compile" argument compiles and executes via nvcc.***
- **"-p" or "--patch" argument applies all patches in the src/patches directory.***
- "-t" or "--tale" argument runs the hipify-perl tool on a single specified sample.
- "-x" or "--remove" argument removes any sample relating to graphical operations e.g. DirectX, Vulcan, OpenGL, OpenCL, and so on.
- **"-s" or "--setup1" argument configures dependencies automatically.***
- **"-v" or "--setup2" argument configures dependencies manually.***
- **"-u" or "--new_samples" argument downloads the latest samples from the NVIDIA CUDA Samples repository.***

Note:Arguments marked with * don’t require a parameter.

HIPIFIED (PORTED) CODES

- QUICKSILVER
- HACC
- SW4LITE
- HPL
- PENNANT
- LAGHOS

AND (ALREADY) MANY MORE...

HIPify-PERL

- Sits in $HIP/bin/ (export PATH=$PATH:[MYHIP]/bin)
- Command line tool: hipify-perl foo.cu > new_foo.cpp
- Compile: hipcc new_foo.cpp
- How does this this work in practice?
  - Hipify source code
  - Check it in to your favorite version control
  - Try to build
  - Manually work on the rest

PORTING HACC

CUDA

```
cudaMemcpyAsync(d_npos,h_npos,sizeof(float4)*SIZE,cudaMemcpy
HostToDevice,stream);

cudaMemcpyAsync(d_mask,h_mask,sizeof(MASK_T)*cnt,cudaMemcp
yHostToDevice,stream);

calcHHCullenDehnen<<<blocksPerGrid, threadsPerBlock, 0,
stream>>>(cnt, SIZE, d_npos, d_mask, rsm);

cudaMemcpyAsync(h_pos,d_npos+(SIZE-
cnt),sizeof(float4)*cnt,cudaMemcpyDeviceToHost,stream);

cudaMemcpyAsync(h_mask,d_mask,sizeof(MASK_T)*cnt,cudaMemcp
yDeviceToHost,stream);
```

HIP

```
hipMemcpyAsync(d_npos,h_npos,sizeof(float4)*SIZE,hipMemcpyHos
tToDevice,stream);

hipMemcpyAsync(d_mask,h_mask,sizeof(MASK_T)*cnt,hipMemcpyH
ostToDevice,stream);

hipLaunchKernelGGL((calcHHCullenDehnen), dim3(blocksPerGrid),
dim3(threadsPerBlock), 0, stream, cnt, SIZE, d_npos, d_mask, rsm);

hipMemcpyAsync(h_pos,d_npos+(SIZE-
cnt),sizeof(float4)*cnt,hipMemcpyDeviceToHost,stream);

hipMemcpyAsync(h_mask,d_mask,sizeof(MASK_T)*cnt,hipMemcpyD
eviceToHost,stream);
```

```cpp
namespace kernel {
  template <int value>
  __global__ void set_value(
      const int num, int *__restrict__ array) {
    auto tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < num) {
      array[tid] = value;
    }
  }
}
int main() {
  // allocation of memory
  // calculation of grid/block_size
  constexpr int value = 3;
  kernel::set_value<value>
    <<<dim3(grid_size), dim3(block_size)>>> (
        num, array);
  return 0;
}
```

CUDA

```cpp
namespace kernel {
  template <int value>
  __global__ void set_value(
      const int num, int *__restrict__ array) {
    auto tid = blockDim.x * blockIdx.x + threadIdx.x;
    if (tid < num) {
      array[tid] = value;
    }
  }
}
int main() {
  // allocation of memory
  // calculation of grid/block_size
  constexpr int value = 3;
  hipLaunchKernelGGL(
    HIP_KERNEL_NAME(kernel::set_value<value>),
    dim3(grid_size), dim3(block_size), 0, 0,
    num, array);
  return 0;
}
```

HIP

Screenshots

```
./src/samples/Samples/0_Introduction/vectorAdd/vectorAdd.out
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
./src/samples/Samples/0_Introduction/simpleOccupancy/simpleOccupancy.out
starting Simple Occupancy

[ Manual configuration with 32 threads per block ]
Potential occupancy: 40%
Elapsed time: 0.39264ms

[ Automatic, occupancy-based configuration ]
Suggested block size: 1024
Minimum grid size for maximum occupancy: 120
Potential occupancy: 80%
Elapsed time: 0.03584ms

Test PASSED
```

```
./src/samples/Samples/5_Domain_Specific/p2pBandwidthLatencyTest/p2pBandwidthLatencyTest.out
[P2P (Peer-to-Peer) GPU Bandwidth Latency Test]
Device: 0, , pciBusID: 19, pciDeviceID: 0, pciDomainID:0
Device: 1, , pciBusID: 67, pciDeviceID: 0, pciDomainID:0
Device=0 CANNOT Access Peer Device=1
Device=1 CANNOT Access Peer Device=0

***NOTE: In case a device doesn't have P2P access to other one, it falls back to normal memcopy procedure.
So you can see lesser Bandwidth (GB/s) and unstable Latency (us) in those cases.

P2P Connectivity Matrix
     D\D     0     1
     0       1     0
     1       0     1
Unidirectional P2P=Disabled Bandwidth Matrix (GB/s)
Cuda failure src/samples/Samples/5_Domain_Specific/p2pBandwidthLatencyTest/p2pBandwidthLatencyTest.cu.hip:211: 'out of memory'
```

```
./src/samples/Samples/0_Introduction/simpleCooperativeGroups/simpleCooperativeGroups.out

Launching a single block with 64 threads...

 Sum of all ranks 0..63 in threadBlockGroup is 2016 (expected 2016)

 Now creating 4 groups, each of size 16 threads:

   Sum of all ranks 0..15 in this tiledPartition16 group is 120 (expected 120)
   Sum of all ranks 0..15 in this tiledPartition16 group is 120 (expected 120)
   Sum of all ranks 0..15 in this tiledPartition16 group is 120 (expected 120)
   Sum of all ranks 0..15 in this tiledPartition16 group is 120 (expected 120)

...Done.
```

Screenshots

```
usage: testhipify.py [-h] [-a ALL] [-b GENERATE] [-c COMPILE1] [-d COMPILE2] [-e EXECUTE] [-f GENERATE_ALL] [-g COMPILE1_ALL]
                     [-i COMPILE2_ALL] [-j EXECUTE_ALL] [-k PARENTHESIS_CHECK] [-l PARENTHESIS_CHECK_ALL] [-p] [-t TALE]
                     [-x REMOVE] [-s]

HIPIFY Cuda Samples.Please avoid and ignore samples with graphical operations

options:
  -h, --help            show this help message and exit
  -a ALL, --all ALL     To run hipify-perl for all sample:python testhipify.py --all "[PATH TO SAMPLE FOLDER]"
  -b GENERATE, --generate GENERATE
                        Generate .hip files
  -c COMPILE1, --compile1 COMPILE1
                        Compile .hip files
  -d COMPILE2, --compile2 COMPILE2
                        Compile .hip files with static libraries
  -e EXECUTE, --execute EXECUTE
                        Execute .out files
  -f GENERATE_ALL, --generate_all GENERATE_ALL
                        Generate all .hip files
  -g COMPILE1_ALL, --compile1_all COMPILE1_ALL
                        Compile all .hip files
  -i COMPILE2_ALL, --compile2_all COMPILE2_ALL
                        Compile all .hip files with static libraries
  -j EXECUTE_ALL, --execute_all EXECUTE_ALL
                        Execute all .out files
  -k PARENTHESIS_CHECK, --parenthesis_check PARENTHESIS_CHECK
                        Remove last parts from cu.hip files which are out of bounds.
  -l PARENTHESIS_CHECK_ALL, --parenthesis_check_all PARENTHESIS_CHECK_ALL
                        Remove all last parts from cu.hip files which are out of bounds.
  -p, --patch           Apply all patches in src/patches
  -t TALE, --tale TALE  To run hipify-perl for single sample:python testhipify.py -t "[PATH TO SAMPLE]"
  -x REMOVE, --remove REMOVE
                        Remove any sample relating to graphical operations e.g.DirectX,Vulcan,OpenGL,OpenCL and so on.
  -s, --setup           Configure dependencies.
```

Conclusion

- Testhipify adheres to not be dependent on the makefiles of any sample.It has provisions for processing single or multiple samples in a folder.The -h command would give the methods of usage of every function.
- The repository also contains the executable files of samples which were successfully processed under HIP Architecture.The project helped in reporting underlying features, which when solved,would aid in execution of increased amount of samples which currently reside in the Ignored Samples Category.
- Our project ignored 94 Samples and processed 96 Samples in the end out of a total of 190 Samples.Unlike the original guideline which requires manipulation of the make file,the script will generate the executeable without specifying the hardware information as long as the stated dependencies are met.
- It is open-source and the included report explains every system calls and functions to the minutest of details.

Links:

- https://github.com/AryamanAMD/testhipify
- https://github.com/AryamanAMD/testhipify-cli
- https://github.com/ROCm-Developer-Tools/HIPIFY

THANK YOU!