A project report on

TESTHIPIFY TOOL

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

by

ARYAMAN MISHRA (19BCE1027)

SCOPE

May,2023

TESTHIPIFY TOOL

Submitted in partial fulfillment for the award of the degree of

Bachelor of Technology

by

ARYAMAN MISHRA (19BCE1027)

SCOPE

May,2023

DECLARATION

I here by declare that the thesis entitled “TESTHIPIFY TOOL” submitted by me, for the award of the degree of Specify the name of the degree VIT is a record of bonafide work carried out by me under the supervision of Prof. Suguna M.

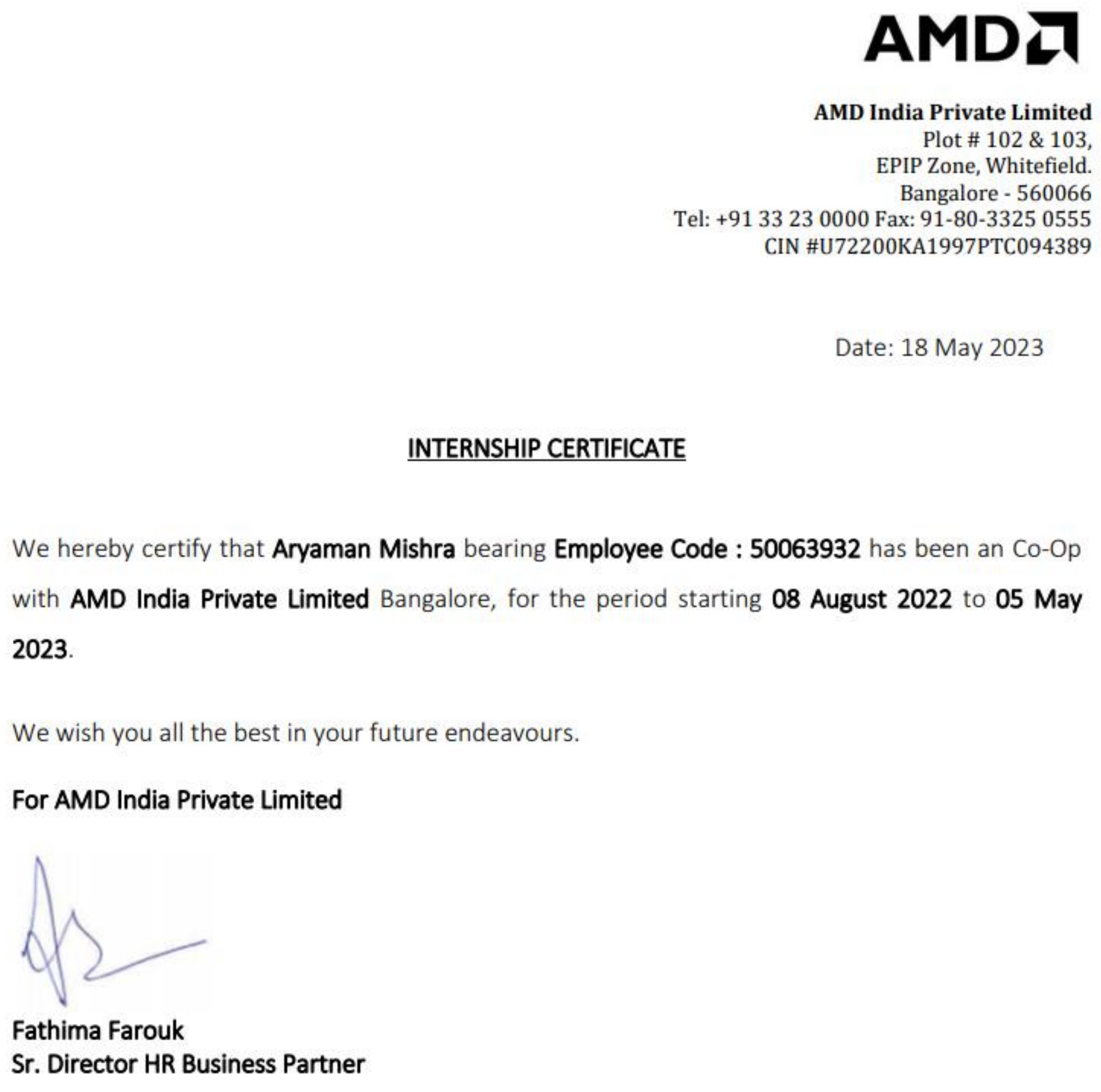
I further declare that the work reported in this thesis has not been submitted and will not be submitted, either in part or in full, for the award of any other degree or diploma in this institute or any other institute or university.

Aryaman

Place: Chennai

Date:5th May,2023 Signature of the Candidate

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

<u>INTERNSHIP CERTIFICATE</u>

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

[illegible]

Fathima Farouk
Sr. Director HR Business Partner

ABSTRACT

The project titled "Testhipify" is a comprehensive software tool developed to facilitate the process of migrating legacy C++ codebases to modern C++ standards. The primary objective of this project is to automate the process of converting outdated C++ code that uses the Hipify-Clang toolchain to leverage modern GPU programming frameworks such as HIP (Heterogeneous-Compute Interface for Portability). By automating the code transformation process, testhipify significantly reduces the time and effort required for developers to upgrade their codebases.This project utilizes the Hipify-Clang tool, which is a modified version of the Clang compiler that enables the transformation of CUDA code to HIP code. The testhipify tool extends the capabilities of Hipify-Clang by providing a user-friendly interface and enhanced automation features. Developers can use testhipify to specify the input directory containing the legacy C++ code and the output directory where the transformed code will be generated.

The testhipify tool effectively translates the CUDA-specific code constructs and CUDA libraries to their HIP equivalents, ensuring that the migrated code remains functionally equivalent. It also performs necessary adaptations and updates to comply with modern C++ standards and best practices. Moreover, the project offers extensive customization options, allowing developers to tailor the migration process according to their specific requirements.Through the utilization of testhipify, developers can significantly accelerate the migration of their C++ codebases to HIP, enabling them to leverage the benefits of GPU programming frameworks efficiently. The project's open-source nature encourages collaboration and contribution from the developer community, further enhancing its capabilities and expanding its compatibility with different codebases.
CUDA samples are code examples provided by NVIDIA that demonstrate various features and capabilities of the CUDA programming model. These samples are designed to help developers understand and learn how to utilize CUDA effectively in their applications. They showcase different CUDA APIs, programming techniques, and GPU-accelerated algorithms.

On the other hand, testhipify is a software tool specifically developed to automate the migration process of legacy C++ codebases to modern GPU programming frameworks, with a focus on converting CUDA code to HIP code. While CUDA samples serve as valuable resources for learning CUDA programming, testhipify complements this by providing a systematic and automated approach to transform existing CUDA code to HIP code.

Developers can use testhipify to migrate their own code, including code derived from CUDA samples, to HIP, enabling them to leverage the benefits of GPU programming frameworks other than CUDA. By automatically translating CUDA-specific constructs and libraries to their HIP equivalents, testhipify simplifies the migration process and reduces the manual effort required.

ACKNOWLEDGEMENT

It is my pleasure to express with deep sense of gratitude to Dr. M. Suguna, M.E., Ph.D., Assistant Professor Senior Grade 2, School of Computer Science and Engineering (SCOPE), Vellore Institute of Technology, for her constant guidance, continual encouragement, understanding; more than all, she taught me patience in my endeavor. My association with her is not confined to academics only, but it is a great opportunity on my part of work with an intellectual and expert in the field of HPC and GPU Operations.

I would like to extend my gratitude to Phanee Sir,Sayak Sir and Mohan Sir for helping me with solving issues regarding dependencies, patch creation, sample processing and endowing information regarding HIP Architecture, Hardware Usage, Sample Purposes and Project layout.

I would also thank Jayshree Ma'am for allowing me to take part in what was probably my first high-order project task as part of my tenure as an intern at AMD

I would like to express my gratitude to Dr.G.Viswanathan,Mr. Sankar Viswanathan,Dr.. Sekar Viswanathan,Mr.G V Selvam,Dr. Rambabu Kodali,Dr. Partha Sharathi Mallick, Dr. Jayabarathi T and Dr.P.Nithyanandam,SCOPE, for providing with an environment to work in and for his inspiration during the tenure of the course.

In jubilant mood I express ingeniously my whole-hearted thanks to Dr. Ganesan R. Dean of SCOPE and all teaching staff and members working as limbs of our university for their not-self-centered enthusiasm coupled with timely encouragements showered on me with zeal, which prompted the acquirement of the requisite knowledge to finalize my course study successfully. I would like to thank my parents for their support.

It is indeed a pleasure to thank my friends who persuaded and encouraged me to take up and complete this task. At last but not least, I express my gratitude and appreciation to all those who have helped me directly or indirectly toward the successful completion of this project.

Place: Chennai **ARYAMAN MISHRA**

Date:5th May 2023. Name of the student

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	**Expansion**
HIP	Heterogeneous-compute Interface for Portability
API	Application Programming Interface
GPU	Graphical Processing Unit
CPU	Central Processing Unit
OpenMP	Open Multi-Processing
CUDA	Compute Unified Device Architecture
HPC	High Performance Computing
MPI	Message Passing Interface
ROCm	Radeon Open Compute
AMD	Advanced Micro Devices
SMI	System Management Interface
CLI	Command Line Interface

LIST OF SCREENSHOTS

Chapter 1:Preamble

1.1 Introduction to ROCm

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.

The need for GPUs has grown in recent years due to the increasing demand for processing power required by modern applications. CPUs, or Central Processing Units, are the primary processors in most computers, but they are not optimized for the type of parallel processing required by many modern applications. GPUs can perform many calculations simultaneously, making them much faster for many tasks.

A GPU, or Graphics Processing Unit, is a specialized electronic circuit designed to handle the processing of images and graphics-related tasks.

In contrast to a CPU, which is a general-purpose processor designed to handle a wide variety of tasks, a GPU is specifically designed to handle tasks that involve parallel processing, such as rendering and displaying 3D graphics, video processing, and machine learning.

GPUs can perform many calculations simultaneously, which makes them ideal for tasks that require a lot of processing power, such as running complex simulations, mining cryptocurrencies, or training neural networks.

GPUs are commonly found in gaming and multimedia devices, as well as high-performance computing systems used in scientific research and data analysis. They are also used in mobile devices, such as smartphones and tablets, to handle graphics-intensive tasks like gaming and video playback.

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.

CUDA allows programmers to use the parallel processing power of NVIDIA GPUs to accelerate computationally intensive tasks in a wide range of applications, including scientific simulations, data analytics, image and video processing, and machine learning.It provides a set of APIs and tools that allow programmers to write code in high-level languages such as C, C++, and Python, which can be compiled to run on NVIDIA GPUs. The CUDA platform provides access to the GPU hardware architecture, allowing programmers to write code that executes in parallel on thousands of GPU cores, which can significantly speed up computations compared to running them on a CPU.

CUDA also includes libraries for common mathematical operations, such as linear algebra and fast Fourier transforms, which are optimized for GPU performance. This allows developers to use these libraries to accelerate their applications without having to write low-level GPU code themselves.

Overall, CUDA has become a popular platform for GPU computing, and it is widely used in scientific research, industry, and academia

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

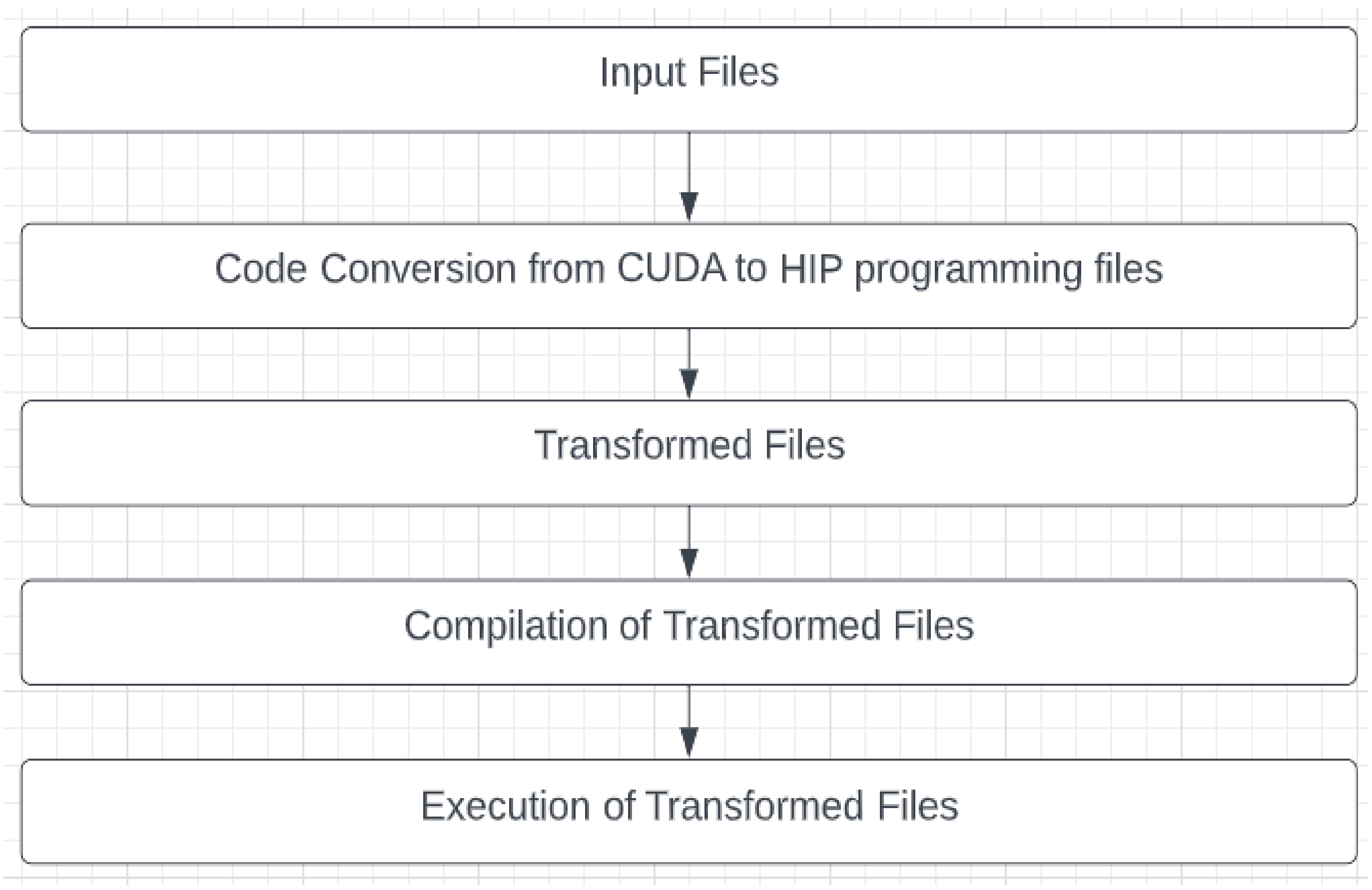
a wide range of applications, including machine learning, scientific simulations, and data analytics.
The ROCm stack includes several components, including the ROCm runtime, which provides a low-level interface for programming AMD GPUs, and the ROCm OpenCL runtime, which supports the OpenCL standard for parallel computing. Other components of the ROCm stack include the ROCm SMI (System Management Interface) for monitoring and managing AMD GPUs, and the ROCm Math Libraries for performing common mathematical operations on GPUs. Overall, the ROCm stack provides a comprehensive set of tools for developers working with AMD GPUs.It is an open-source software platform developed by AMD for GPU computing.
Like NVIDIA's CUDA platform, ROCm provides a programming environment for developers to accelerate their applications using AMD GPUs. The ROCm platform includes a suite of tools, compilers, and libraries that allow programmers to write code in a variety of languages, including C, C++, and Python, and execute it on AMD GPUs.
ROCm also provides a framework for running and managing machine learning workloads, called MIOpen, which includes optimized libraries for deep learning tasks such as convolutional neural networks and recurrent neural networks.
In addition, ROCm supports popular machine learning frameworks such as TensorFlow and PyTorch, allowing users to easily run these frameworks on AMD GPUs.
One of the unique features of ROCm is that it is an open-source platform, meaning that users have access to the source code and can modify it to suit their needs. This makes it an attractive option for users who want more control over their GPU computing environment or who want to contribute to the development of the platform.
Overall, ROCm has become a popular choice for users who want to take advantage of the GPU computing capabilities of AMD GPUs and is widely used in scientific research, industry, and academia.

1.2 What is HIPification?

HIPification is a process of converting CUDA code to a vendor-agnostic format that can be run on both NVIDIA and AMD GPUs.
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, a tool 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.
The HIPification process can help to improve the portability of CUDA code and reduce vendor lock-in, allowing developers to run their applications on a wider range of hardware. This can be particularly useful for users who want to take advantage of the computing power of AMD GPUs, which can be more cost-effective than NVIDIA GPUs in some cases.
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.

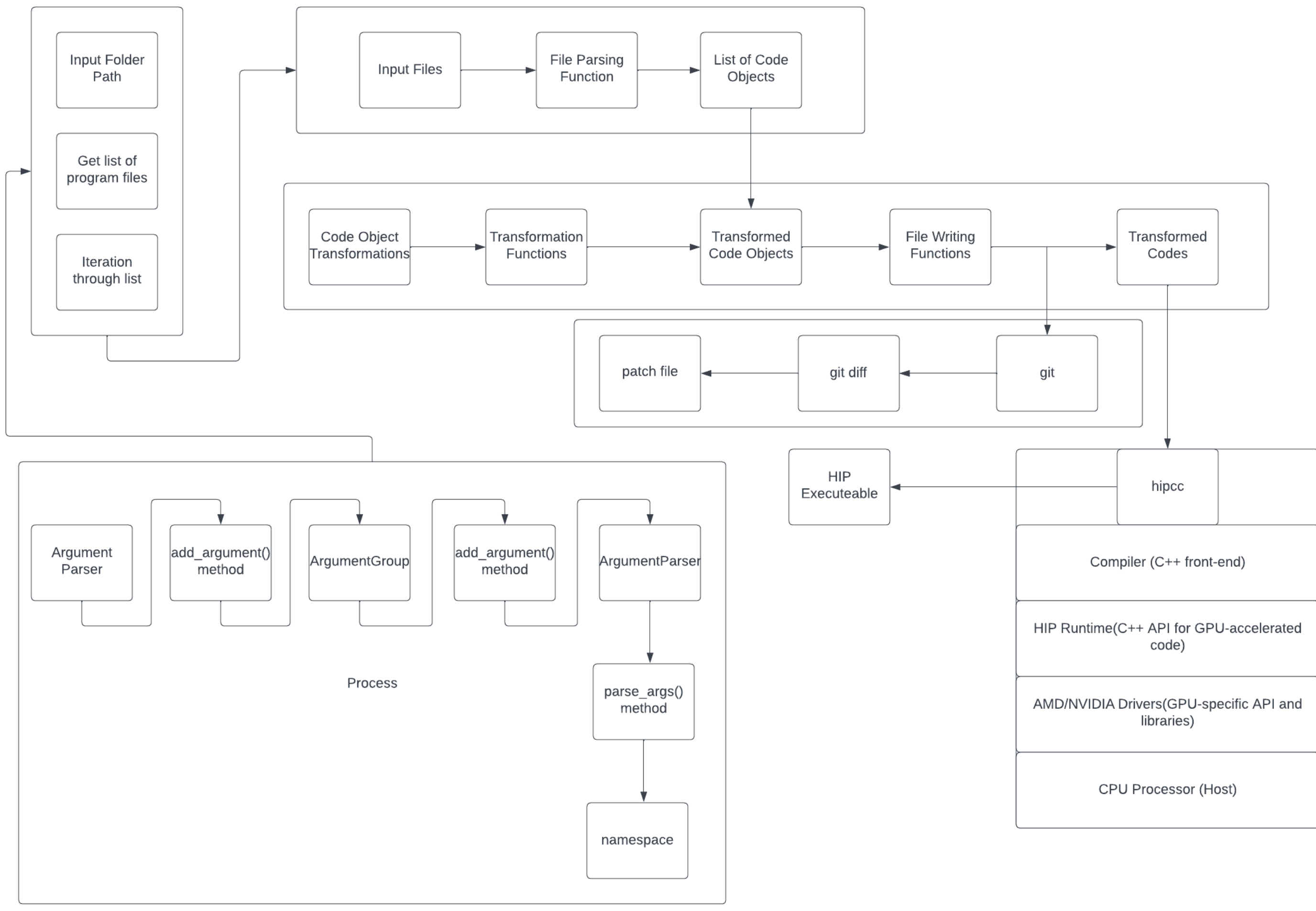
For example, if a developer has some CUDA-based C++ code that they want to convert to a different language(HIP), they can use testhipify to perform the conversion. The program will modify the code according to the rules and patterns for the new language, and output the transformed code to a new file. This can save the developer a lot of time and effort, as they don't have to manually rewrite the code from scratch.
Overall, testhipify is a useful tool for developers who work with multiple programming languages or who want to convert their code from one language to another.

1.3 High and Low Level Design Diagram


Input Files
Code Conversion from CUDA to HIP programming files
Transformed Files
Compilation of Transformed Files
Execution of Transformed Files

The system takes the source code files as input, which can be in the form of one or more files. The code conversion component of the system takes the input code files and applies various rules and patterns to transform the code from one programming language to another. This may involve changing the syntax, formatting, and other aspects of the code to make it compatible with the target language.
Once the code has been converted, the transformed files are outputted, which can be in the form of one or more files or directories. These transformed files can then be used in place of the original code, or as part of a larger project.The transformed files can then be compiled using HIP's equivalent of gcc (hipcc) and executed as any other conventional programming language.
The input to the system is one or more source code files, which are first parsed by a file parsing function to create a list of code objects. Each code object represents a section of code that can be transformed independently of the others.
Next, the code object transformation component of the system takes each code object and applies various rules and patterns to transform it from the original language to the target

language. This may involve modifying the syntax, changing the structure, or applying other changes to make the code compatible with the target language.
Once the code object has been transformed, the transformed code object is written to a new file using a file writing function. This process is repeated for each code object in the list, until all code objects have been transformed and written to new files.



At the bottom right of the diagram is hipcc itself, which takes C++ source code and compiles it into object code for execution on a GPU
Beneath that is the HIP Runtime, which provides a C++ API for GPU-accelerated code. This API is used by the code generated by hipcc to interact with the GPU and its associated memory.
Below the HIP Runtime are the AMD/NVIDIA Drivers, which provide a GPU-specific API and libraries that the HIP Runtime uses to communicate with the GPU.

Chapter 2:Hardware Specs and Software Design

2.1 Hardware Specs

The MI50 SMC is a server GPU manufactured by AMD. Here are the hardware specifications for the MI50 SMC:

- GPU Architecture: Vega 7nm
- Stream Processors: 3840
- Peak Single Precision Performance: 13.44 TFLOPS
- Peak Double Precision Performance: 6.72 TFLOPS
- Memory: 16 GB of HBM2 ECC memory
- Memory Bandwidth: 1 TB/s
- TDP: 300W
- Form Factor: Single Slot, Full Height
- Interface: PCIe 4.0 x16

Please note that these specifications are for the MI50 SMC GPU only and not a complete server system.

The specifications for the device on which the tool was developed is as follows:

- Processor: AMD Ryzen 7 PRO 5850U Processor (up to 4.4 GHz, 16 MB L3 cache, 8 cores)
- Operating System: Windows 10 Pro 64
- Display: 14-inch diagonal, FHD (1920 x 1080), IPS, anti-glare, low power, 1000 nits, with Sure View Reflect privacy screen
- Memory: 16 GB DDR4-3200 SDRAM (2 x 8 GB), up to 64 GB maximum
- Storage: 512 GB PCIe NVMe SSD, up to 2 TB maximum
- Graphics: Integrated AMD Radeon Graphics
- Wireless: Intel Wi-Fi 6 AX200 (2x2) and Bluetooth 5.2
- Ports: 2 USB Type-C with Thunderbolt 4 support, 2 USB 3.2 Gen 1 (1 charging), HDMI 2.0b, headphone/microphone combo jack, AC power
- Camera: 720p HD IR webcam with privacy shutter
- Audio: Bang & Olufsen, dual stereo speakers, 3 multi-array microphones
- Battery: HP Long Life 3-cell, 53 Wh Li-ion polymer battery, supports HP Fast Charge (50% in 30 minutes)
- Dimensions (W x D x H): 12.72 x 8.14 x 0.7 in
- Weight: Starting at 2.99 lb

2.2 Flowchart

Start: The program starts at the top of the file, with the import statements for necessary libraries and the definition of a hipify function.

Input: The program takes input in the form of a file path or directory path provided by the user.

Decision: The program checks whether the input path is a file or a directory. If it is a file, the program proceeds to the next step. If it is a directory, the program enters a loop to process all files within the directory.

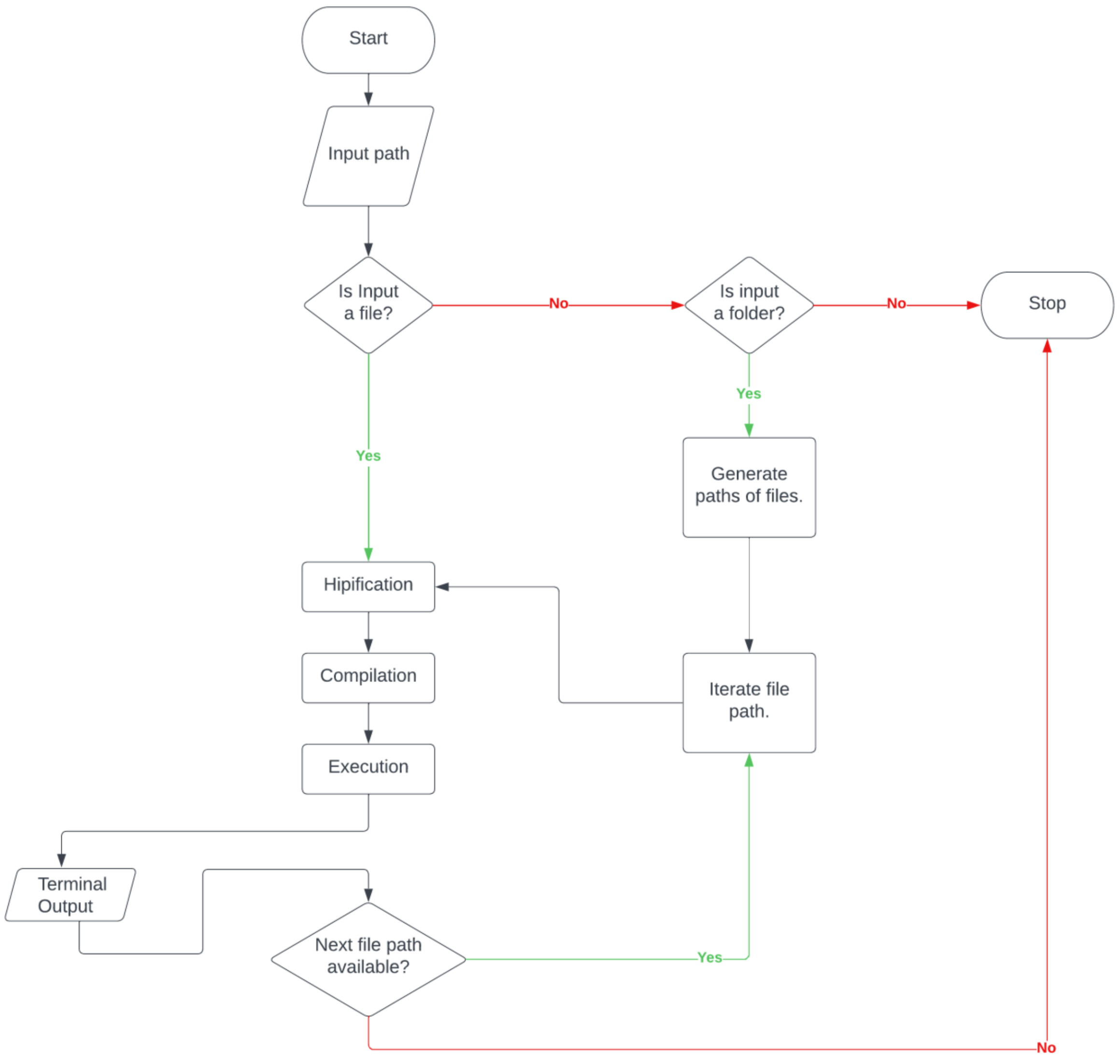
Processing: The program reads the input file and passes it to the hipify function for processing.

Hipify: The hipify function performs the transformation of the input code, using regular expressions and string manipulations to find and replace specific patterns.

Output: The program writes the transformed code to an output file, with a name derived from the original file name.

Loop: If the input path is a directory, the program loops back to the input step to process the next file in the directory.

End: Once all files have been processed, the program ends.



Chapter 3: Implementation

3.1 Layman's Terms Explanation

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.

3.2 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
- Image and video processing

- Scientific and engineering simulations
- Machine learning and deep learning
- Database acceleration

Each sample includes source code, documentation, and build instructions to help developers get started quickly. The purpose of the samples is to provide a starting point for developers to learn how to write their own CUDA-based applications and accelerate their development process.

AMD HIPIFY is a tool that converts CUDA code written for Nvidia GPUs to run on AMD GPUs using the HIP API. The aim of the project is to allow developers to write code once for both GPU platforms, making it easier to target both Nvidia and AMD GPUs without having to maintain separate codebases.

HIPIFY can be used as a command-line tool to convert CUDA code to HIP. It requires the CUDA code to be written in a standard format and depends on the CUDA and HIP API libraries. Before using HIPIFY, the user must have a development environment set up with the proper tools and dependencies installed, including a compiler that supports C++11, CMake, and the CUDA and HIP API libraries.

To use HIPIFY, the user specifies the CUDA source code file(s) as input, and the tool generates the equivalent HIP code. The generated code can then be compiled and executed on AMD GPUs using the HIP API.

HIPIFY is not a perfect converter, and some manual adjustments to the generated HIP code may be necessary to ensure correct functionality on AMD GPUs. The repository accounts for the following limitations of HIPIFY:

1. Conversion Accuracy: HIPIFY may not accurately convert all CUDA code, especially code that uses non-standard CUDA features or idioms. Some manual adjustments to the generated HIP code may be necessary to ensure correct functionality on AMD GPUs.
2. Limited Support for CUDA Libraries: HIPIFY may not support all CUDA libraries, and some libraries may need to be re-implemented using the HIP API.
3. Performance Differences: HIP and CUDA have different performance characteristics, and the performance of converted HIP code may differ from the original CUDA code.
4. Code Portability: HIP is not as widely adopted as CUDA, and some platforms may not support HIP.
5. Maintenance: HIP and CUDA are constantly evolving, and HIPIFY may need to be updated to keep up with changes to these APIs.

Overall, while HIPIFY can be a useful tool for porting CUDA code to AMD GPUs, it is not a perfect solution and may require some manual effort and testing to ensure correct and optimal functionality.

The GitHub repository linked at https://github.com/AryamanAMD/testHIPIFY is a code repository that contains various files related to the "testHIPIFY" project.

Based on the file structure in the repository, the following are the main components:

HIPIFY-perl: This directory contains Perl scripts that can be used to automatically convert CUDA code to HIP code.

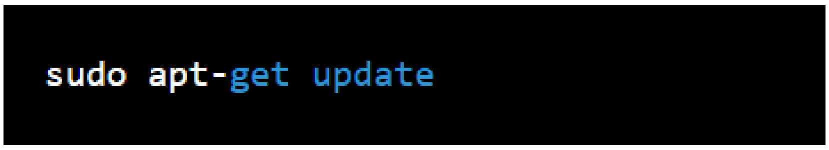
HIPIFY-clang: This directory contains the source code for a modified version of the Clang compiler that can be used to convert CUDA code to HIP code.

testHIPIFY: This directory contains various files and directories related to the testHIPIFY project, including:

3.3 Pre-requisites/Dependencies

Our first priority is to install Python on our devices. To install Python on a Linux system, follow these steps:

Open a terminal window.



```
sudo apt-get update
```

Install Python by running the following command:

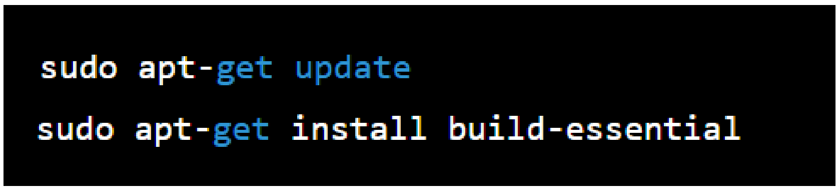
Verify the installation by checking the version of Python:

The above steps are for Debian-based distributions (e.g. Ubuntu). The steps may vary slightly for other distributions.

Update your package index by running the following command:

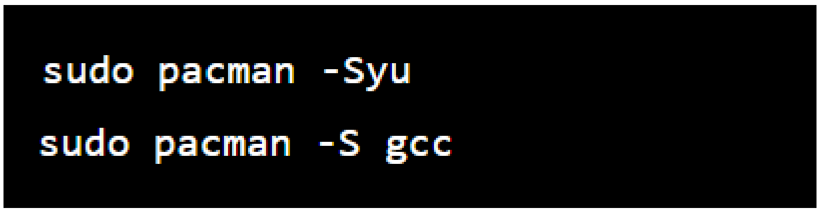
To install the GCC compiler in Linux, you can use the package manager specific to your distribution. Here are the steps for some popular distributions:

Ubuntu/Debian:



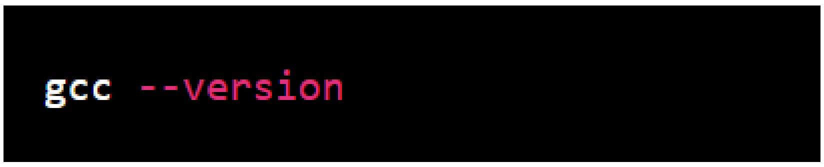
```
sudo apt-get update
sudo apt-get install build-essential
```

Fedora/RHEL/CentOS:



```
sudo pacman -Syu
sudo pacman -S gcc
```

Arch Linux:



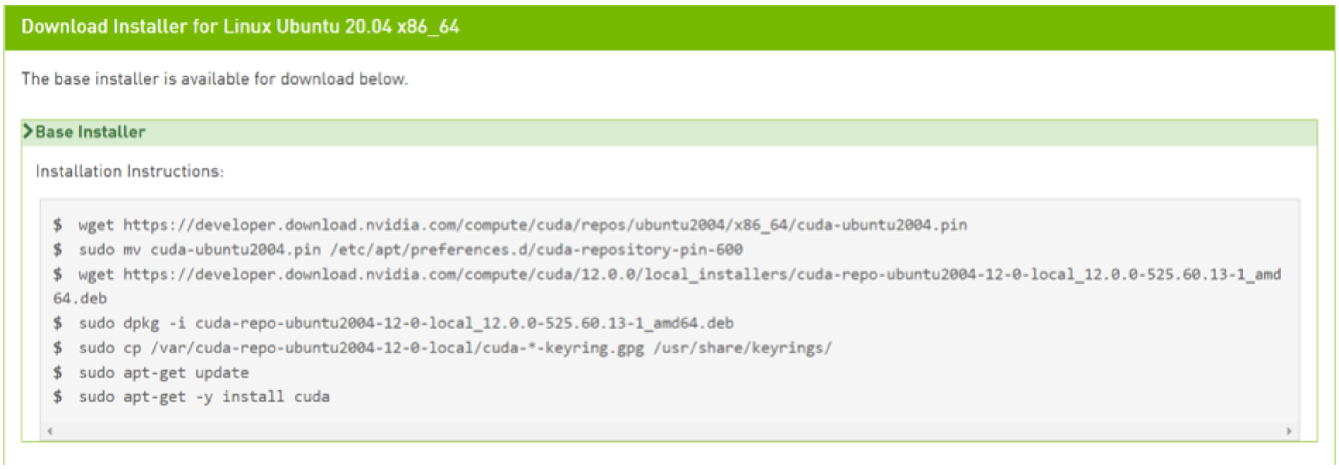
```
gcc --version
```

Once the installation is complete, you can check the version of GCC installed by running the following command: These are the basic steps to install the GCC compiler on Linux.

To install CUDA on Linux, you can follow these steps:

Check system requirements: Make sure your system meets the minimum requirements for CUDA installation, including a compatible NVIDIA GPU and a supported Linux distribution (such as Ubuntu or Fedora).

Download CUDA Toolkit: Download the CUDA Toolkit from the NVIDIA website (https://developer.nvidia.com/cuda-downloads) and select the version that matches your system configuration.


Download Installer for Linux Ubuntu 20.04 x86_64

The base installer is available for download below.

Base Installer

Installation Instructions:

```
$  wget https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2004/x86_64/cuda-ubuntu2004.pin
$  sudo mv cuda-ubuntu2004.pin /etc/apt/preferences.d/cuda-repository-pin-600
$  wget https://developer.download.nvidia.com/compute/cuda/12.0.0/local_installers/cuda-repo-ubuntu2004-12-0-local_12.0.0-525.60.13-1_amd
64.deb
$  sudo dpkg -i cuda-repo-ubuntu2004-12-0-local_12.0.0-525.60.13-1_amd64.deb
$  sudo cp /var/cuda-repo-ubuntu2004-12-0-local/cuda-*-keyring.gpg /usr/share/keyrings/
$  sudo apt-get update
$  sudo apt-get -y install cuda
```

Install NVIDIA Drivers: Install the latest NVIDIA drivers for your GPU. You can do this by downloading the driver package from the NVIDIA website or using the package manager for your Linux distribution.

Install CUDA Toolkit: Once you have the NVIDIA drivers installed, you can install the CUDA Toolkit by running the following commands:

```
sudo chmod +x cuda_<version>_linux.run
sudo ./cuda_<version>_linux.run
```

Update PATH and LD_LIBRARY_PATH environment variables: Add the CUDA bin and lib64 directories to your PATH and LD_LIBRARY_PATH environment variables, respectively. You can do this by adding the following lines to your shell profile file (e.g. ~/.bashrc or ~/.bash_profile):

```
export PATH=$PATH:/usr/local/cuda/bin
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/cuda/lib64
```

Verify Installation: Verify the installation by running the following command:

```
nvidia-smi
```

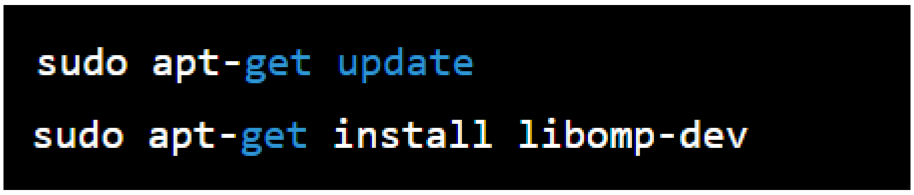
If the installation was successful, the output should display information about your NVIDIA GPU, including its name, memory, and driver version.

These are the basic steps to install CUDA on Linux.

To install OpenMP and Open MPI on Linux, you can use the package manager specific to your distribution. Here are the steps for some popular distributions:

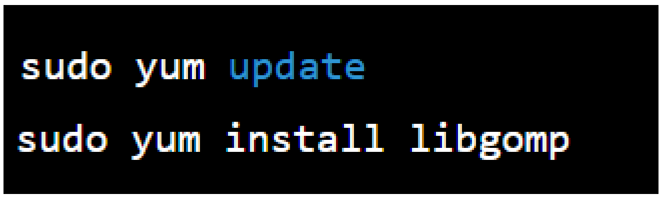
OpenMP:

Ubuntu/Debian:



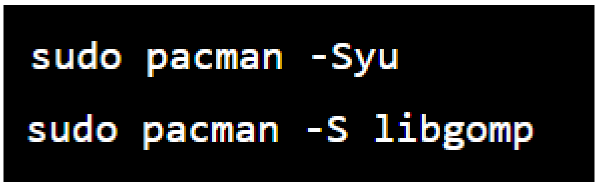
```
sudo apt-get update
sudo apt-get install libomp-dev
```

Fedora/RHEL/CentOS:



```
sudo yum update
sudo yum install libgomp
```

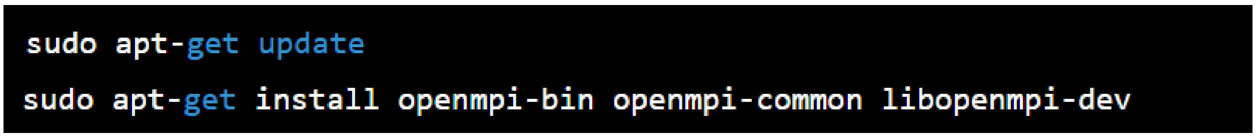
Arch Linux:



```
sudo pacman -Syu
sudo pacman -S libgomp
```

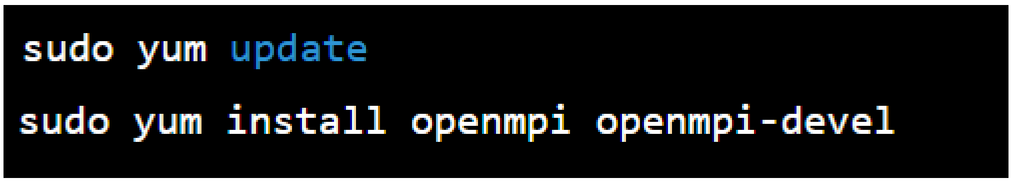
Open MPI:

Ubuntu/Debian:



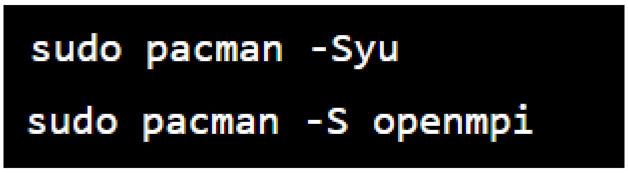
```
sudo apt-get update
sudo apt-get install openmpi-bin openmpi-common libopenmpi-dev
```

Fedora/RHEL/CentOS:



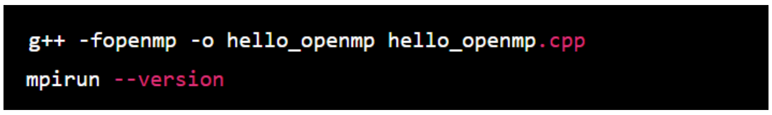
```
sudo yum update
sudo yum install openmpi openmpi-devel
```

Arch Linux:



```
sudo pacman -Syu
sudo pacman -S openmpi
```

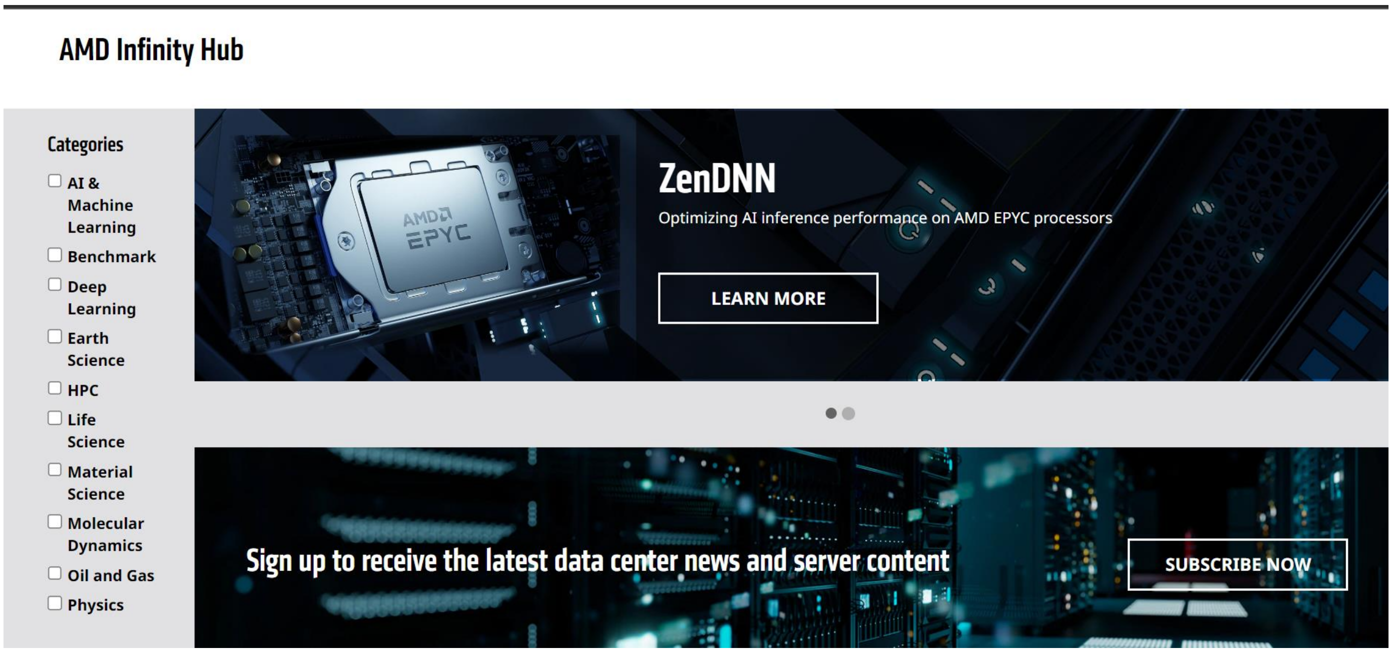
Once the installations are complete, you can check the versions of OpenMP and Open MPI installed by running the following commands:



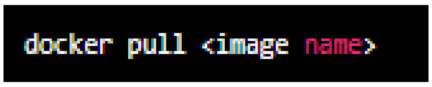
```
g++ -fopenmp -o hello_openmp hello_openmp.cpp
mpirun --version
```

These are the basic steps to install OpenMP and Open MPI on Linux. Depending on your distribution and configuration, the exact steps and packages may vary, but the overall process should be similar.

AMD Docker images are Docker images that are designed to run on AMD processors. Docker is a popular containerization platform that allows developers to package and deploy applications in lightweight, portable containers. AMD Docker images are built specifically for AMD processors and are optimized to take advantage of the performance and capabilities of these processors.AMD provides docker images on an open source environment known as Infinity Hub.The site (https://www.amd.com/en/technologies/infinity-hub) provides dedicated docker images for various frameworks for tasks as AI and Machine Learning, Deep Learning for the following products: AMD Instinct™ MI200, AMD Instinct™ MI100 and AMD Instinct™ MI50 .

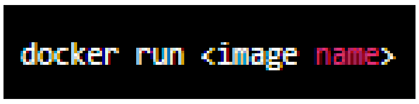


The user can click on the preferred docker image which would automatically copy the pull command. The user can then execute the following command to start the image pulling:



```
docker pull <image name>
```

After completion, the user can access the contents of the image by the following command:



```
docker run <image name>
```

Note:The repository has provisions to make samples run in baremetals as long as hipify-perl is built into it.

3.4 Function Modules and System Calls

The TestHIPIFY repository makes use of the following python-based packages:

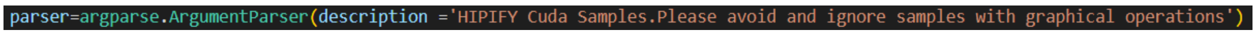
Argparse is a module in the Python standard library that makes it easy to write user-friendly command-line interfaces. The argparse module also automatically generates help and usage messages and issues errors when users give the program invalid arguments.

The module provides a way to parse command-line arguments. It is used to write user-friendly command-line interfaces for scripts and programs. The argparse module provides a convenient way to specify arguments and options, and to generate help messages and error messages.

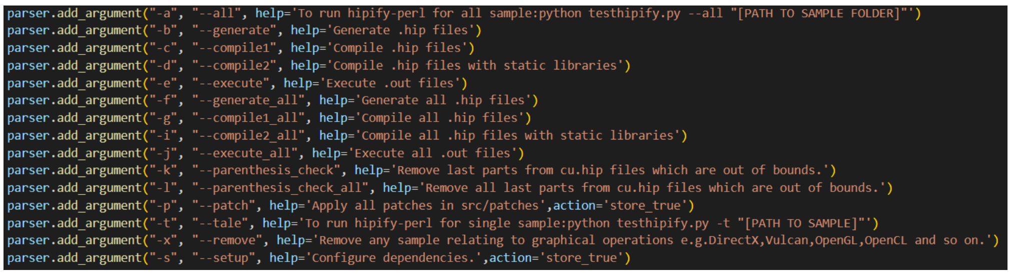
Some of the key features of argparse are:

1. Argument parsing: You can specify the expected arguments and options, along with their types and default values, and argparse will take care of parsing the actual arguments passed by the user.
2. Help messages: You can specify help messages for each argument and option, and argparse will automatically generate a help message for you.
3. Error messages: When the user passes invalid arguments or options, argparse will generate a clear and helpful error message.
4. Namespace object: The parsed arguments are stored in a namespace object, which you can access in your script.

To use argparse, you need to create an ArgumentParser object and specify the arguments and options. For example



```
parser=argparse.ArgumentParser(description ='HIPIFY Cuda Samples.Please avoid and ignore samples with graphical operations')
```



```
parser.add_argument("-a", "--all", help='To run hipify-perl for all sample:python testhipify.py --all "[PATH TO SAMPLE FOLDER]"')
parser.add_argument("-b", "--generate", help='Generate .hip files')
parser.add_argument("-c", "--compile1", help='Compile .hip files')
parser.add_argument("-d", "--compile2", help='Compile .hip files with static libraries')
parser.add_argument("-e", "--execute", help='Execute .out files')
parser.add_argument("-f", "--generate_all", help='Generate all .hip files')
parser.add_argument("-g", "--compile1_all", help='Compile all .hip files')
parser.add_argument("-i", "--compile2_all", help='Compile all .hip files with static libraries')
parser.add_argument("-j", "--execute_all", help='Execute all .out files')
parser.add_argument("-k", "--parenthesis_check", help='Remove last parts from cu.hip files which are out of bounds.')
parser.add_argument("-l", "--parenthesis_check_all", help='Remove all last parts from cu.hip files which are out of bounds.')
parser.add_argument("-p", "--patch", help='Apply all patches in src/patches',action='store_true')
parser.add_argument("-t", "--tale", help='To run hipify-perl for single sample:python testhipify.py -t "[PATH TO SAMPLE]"')
parser.add_argument("-x", "--remove", help='Remove any sample relating to graphical operations e.g.DirectX,Vulcan,OpenGL,OpenCL and so on.')
parser.add_argument("-s", "--setup", help='Configure dependencies.',action='store_true')
```

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: To run HIPIFY-perl for all samples
- -b or --generate: Generate .hip files
- -c or --compile1: Compile .hip files
- -d or --compile2: Compile .hip files with static libraries
- -e or --execute: Execute .out files
- -f or --generate_all: Generate all .hip files
- -g or --compile1_all: Compile all .hip files
- -i or --compile2_all: Compile all .hip files with static libraries
- -j or --execute_all: Execute all .out files
- -k or --parenthesis_check: Remove last parts from cu.hip files which are out of bounds.
- -l or --parenthesis_check_all: Remove all last parts from cu.hip files which are out of bounds.
- -p or --patch: Apply all patches in src/patches

- -t or --tale: To run HIPIFY-perl for a single sample
- -x or --remove: Remove any sample relating to graphical operations
- -s or --setup: Configure dependencies.

The code then parses the command-line arguments using parser.parse_args() and stores the result in a variable named args. The code then uses if statements to check which of the arguments were passed by the user and calls the appropriate functions, such as ftale, fall, generate, apply_patches, compilation_1, compilation_2, runsample, generate_all, compilation_1_all, compilation_2_all, runsample_all, parenthesis_check, parenthesis_check_all, or setup, with the specified argument as the input to the function.

The os module in Python is part of the standard library and provides a way to interact with the underlying operating system. It provides a variety of functionality, including:

- Interacting with the file system (e.g. creating and deleting directories, renaming files)
- Accessing environment variables
- Getting information about the system (e.g. the current working directory, user information)
- Launching other programs (e.g. running shell commands)

Some of the commonly used functions in the os module are:

- os.getcwd(): returns the current working directory.
- os.chdir(): changes the current working directory.
- os.mkdir(): creates a new directory.
- os.rmdir(): removes a directory.
- os.listdir(): returns a list of files in a directory.
- os.remove(): removes a file.
- os.rename(): renames a file or directory.
- os.environ: a dictionary-like object containing environment variables.
- os.system(): execute command line commands in script

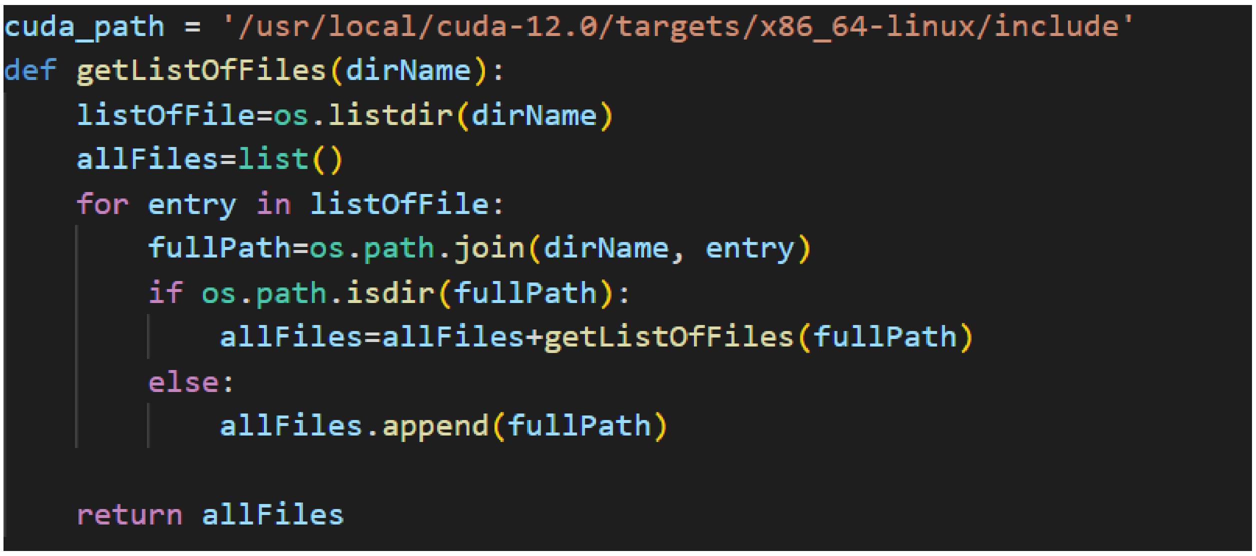
The os module is an essential tool for performing basic operations on the file system and accessing environment information in Python scripts.

The sys module in Python is part of the standard library and provides access to some variables and functions used or maintained by the Python interpreter. Some of the most commonly used features of the sys module are:

- sys.argv: A list of command-line arguments passed to the script.
- sys.exit(): Terminates the script and returns a specified exit code.
- sys.version: A string containing the version number of the Python interpreter.
- sys.platform: A string identifying the platform (e.g. "linux", "win32", "darwin").
- sys.path: A list of strings that specifies the search path for modules.

- sys.stdin, sys.stdout, and sys.stderr: File objects representing standard input, standard output, and standard error, respectively.

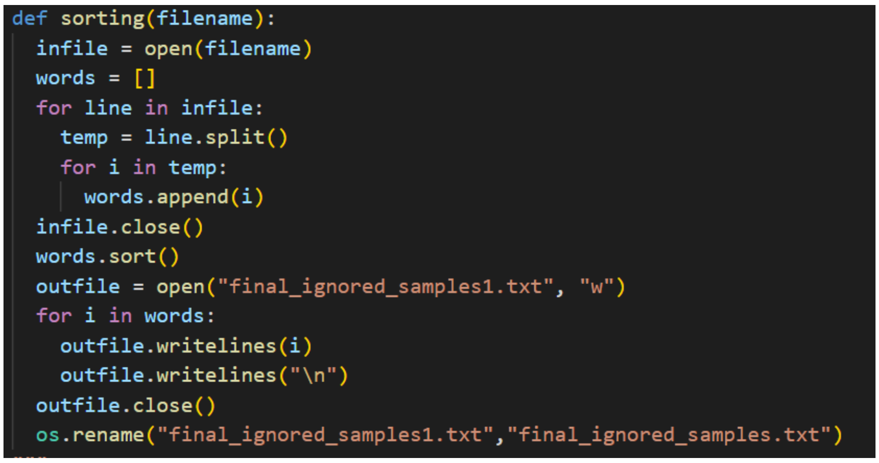
The sys module is often used to interact with the underlying operating system, to access command-line arguments, and to manipulate the environment in which the script runs. It is an essential tool for writing Python scripts that interact with the operating system or other programs.



```python
cuda_path = '/usr/local/cuda-12.0/targets/x86_64-linux/include'
def getListOfFiles(dirName):
    listOfFile=os.listdir(dirName)
    allFiles=list()
    for entry in listOfFile:
        fullPath=os.path.join(dirName, entry)
        if os.path.isdir(fullPath):
            allFiles=allFiles+getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)

    return allFiles
```

The code defines a Python function named getListOfFiles that takes a directory name as an input argument. The purpose of the function is to recursively search a directory and its subdirectories for all files, and return a list of the full paths of these files.The first line cuda_path = '/usr/local/cuda-12.0/targets/x86_64-linux/include' sets a variable cuda_path to a string value representing the path to a directory, but this line is not used in the function.The function starts by calling the os.listdir() function, which takes a directory name as an argument and returns a list of the names of the files and directories in that directory. The list of names is stored in a variable listOfFile.Next, the function creates an empty list allFiles. It then iterates over the elements in listOfFile, using a for loop.

For each element entry, the function creates a full path to the file or directory by calling os.path.join(dirName, entry). The full path is stored in a variable fullPath.If fullPath is a directory (as determined by calling os.path.isdir(fullPath)), the function recursively calls itself with fullPath as the argument to get a list of all the files in that directory and its subdirectories. The list is appended to allFiles.If fullPath is a file, it is added to allFiles using the allFiles.append(fullPath) statement.Finally, the function returns the list allFiles

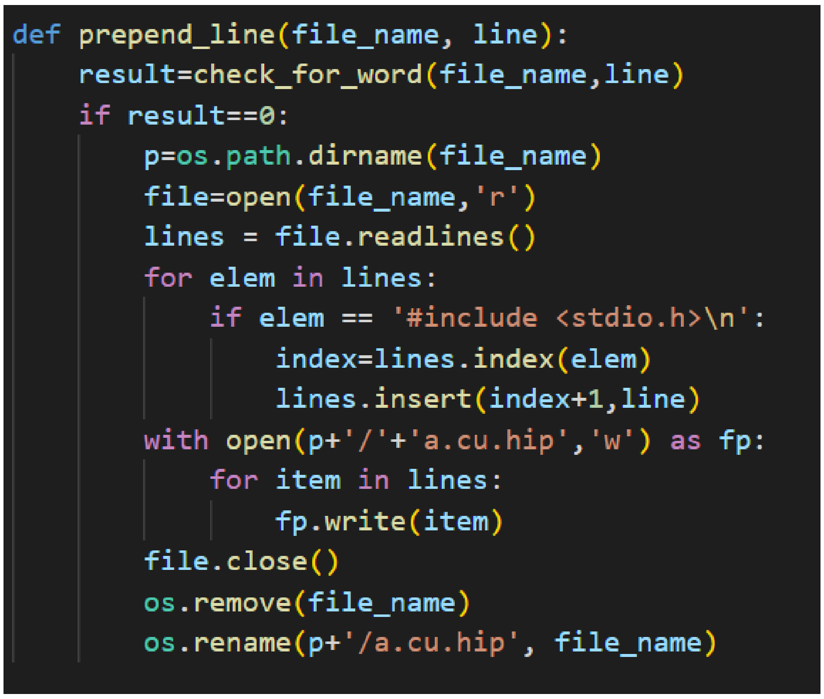


```python
def sorting(filename):
  infile = open(filename)
  words = []
  for line in infile:
    temp = line.split()
    for i in temp:
      words.append(i)
  infile.close()
  words.sort()
  outfile = open("final_ignored_samples1.txt", "w")
  for i in words:
    outfile.writelines(i)
    outfile.writelines("\n")
  outfile.close()
  os.rename("final_ignored_samples1.txt","final_ignored_samples.txt")
```

The code defines a Python function named sorting that takes a filename as an input argument. The purpose of the function is to sort the words in the file alphabetically and write the sorted words to a new file.The function starts by opening the input file specified by filename using infile = open(filename).

It then creates an empty list words to store the words from the input file.The function uses a for loop to iterate over the lines in the input file and split each line into individual words using the line.split() method. The words are then appended to the list words using the words.append(i) statement.After the for loop, the function closes the input file using infile.close().The list words is then sorted using the built-in sort method.

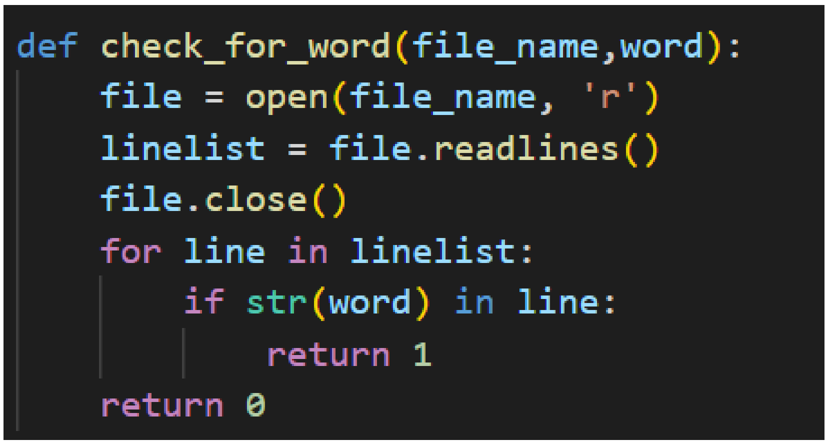
The function then opens a new file named "final_ignored_samples1.txt" for writing using outfile = open("final_ignored_samples1.txt", "w").It uses another for loop to iterate over the sorted words in words and write each word to the new file, followed by a newline character.After the for loop, the function closes the output file using outfile.close().Finally, the function uses the os.rename function to rename the file "final_ignored_samples1.txt" to "final_ignored_samples.txt".



```python
def prepend_line(file_name, line):
    result=check_for_word(file_name,line)
    if result==0:
        p=os.path.dirname(file_name)
        file=open(file_name,'r')
        lines = file.readlines()
        for elem in lines:
            if elem == '#include <stdio.h>\n':
                index=lines.index(elem)
                lines.insert(index+1,line)
        with open(p+'/'+'a.cu.hip','w') as fp:
            for item in lines:
                fp.write(item)
        file.close()
        os.remove(file_name)
        os.rename(p+'/a.cu.hip', file_name)
```

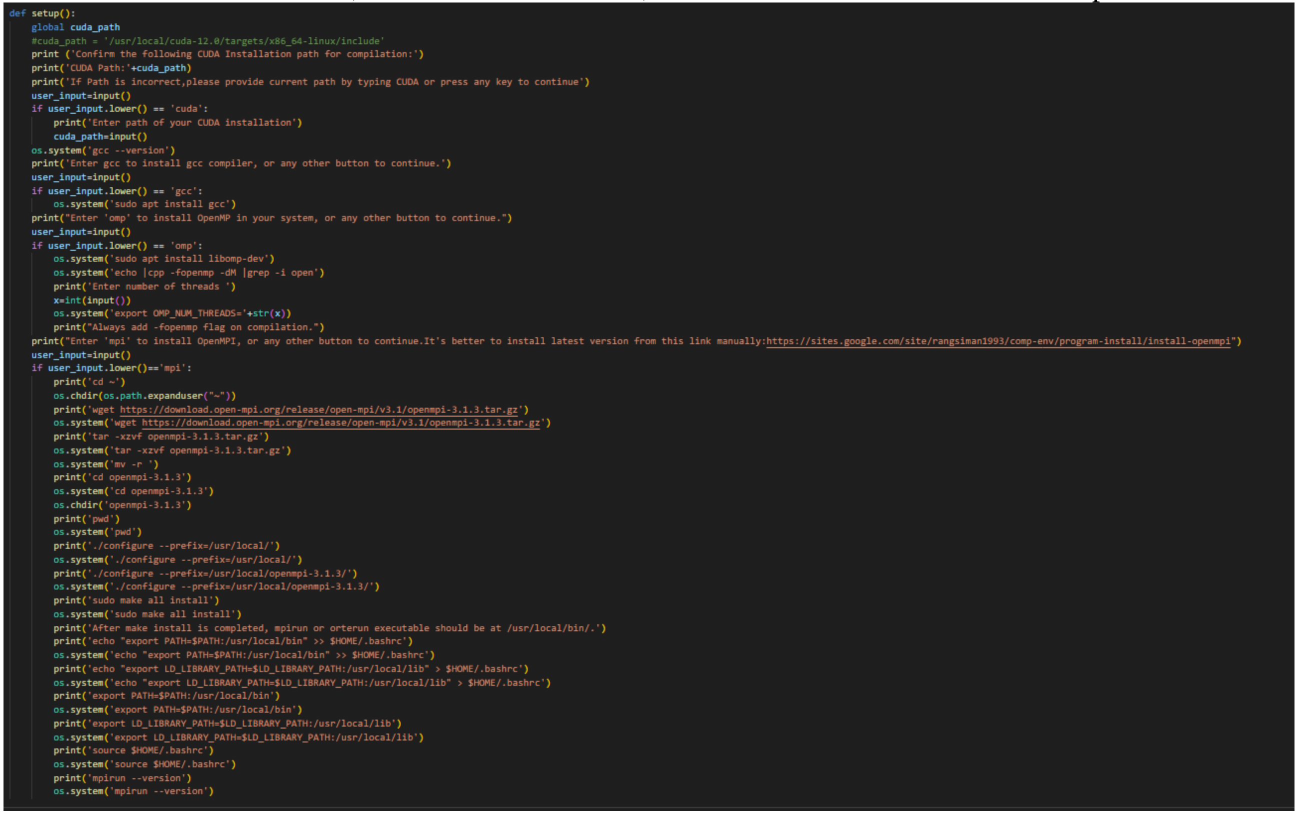
The code defines a Python function named prepend_line that takes two arguments: file_name and line. The purpose of the function is to add a new line of text to an existing file.The function first calls another function named check_for_word to check whether the specified line already exists in the file. If the line is already present in the file, the function returns and does not add the line to the file.If the line is not present in the file, the function uses the os.path.dirname method to extract the directory path from the file_name argument.The function then opens the file specified by file_name in read mode using file=open(file_name,'r').It reads the lines of the file using lines = file.readlines().The function then uses a for loop to iterate over the lines in lines and searches for a line that matches the string '#include <stdio.h>\n'. If it finds this line, it uses the index method to get the index of the line in lines, and inserts the specified line immediately after this line using the lines.insert method.The function then opens a new file named "a.cu.hip" for writing using with open(p+'/'+'a.cu.hip','w') as fp.It uses another for loop to iterate over the lines in lines and write each line to the new file.After the for loop, the function closes the original file using file.close().

It then removes the original file using the os.remove method and renames the new file "a.cu.hip" to the original filename using os.rename.



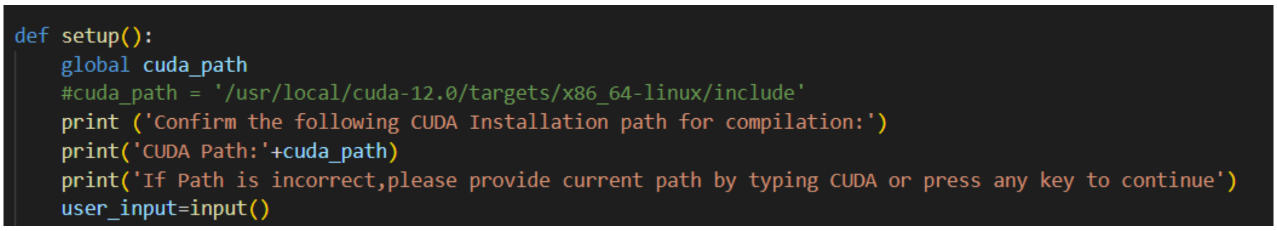
```python
def check_for_word(file_name,word):
    file = open(file_name, 'r')
    linelist = file.readlines()
    file.close()
    for line in linelist:
        if str(word) in line:
            return 1
    return 0
```

This code takes a file name and a word as input, opens the file for reading, reads all lines into a list of lines, and closes the file. Then it iterates through each line in the list and checks if the word is in the line. If it is, the function returns 1, otherwise it returns 0 after the loop ends.

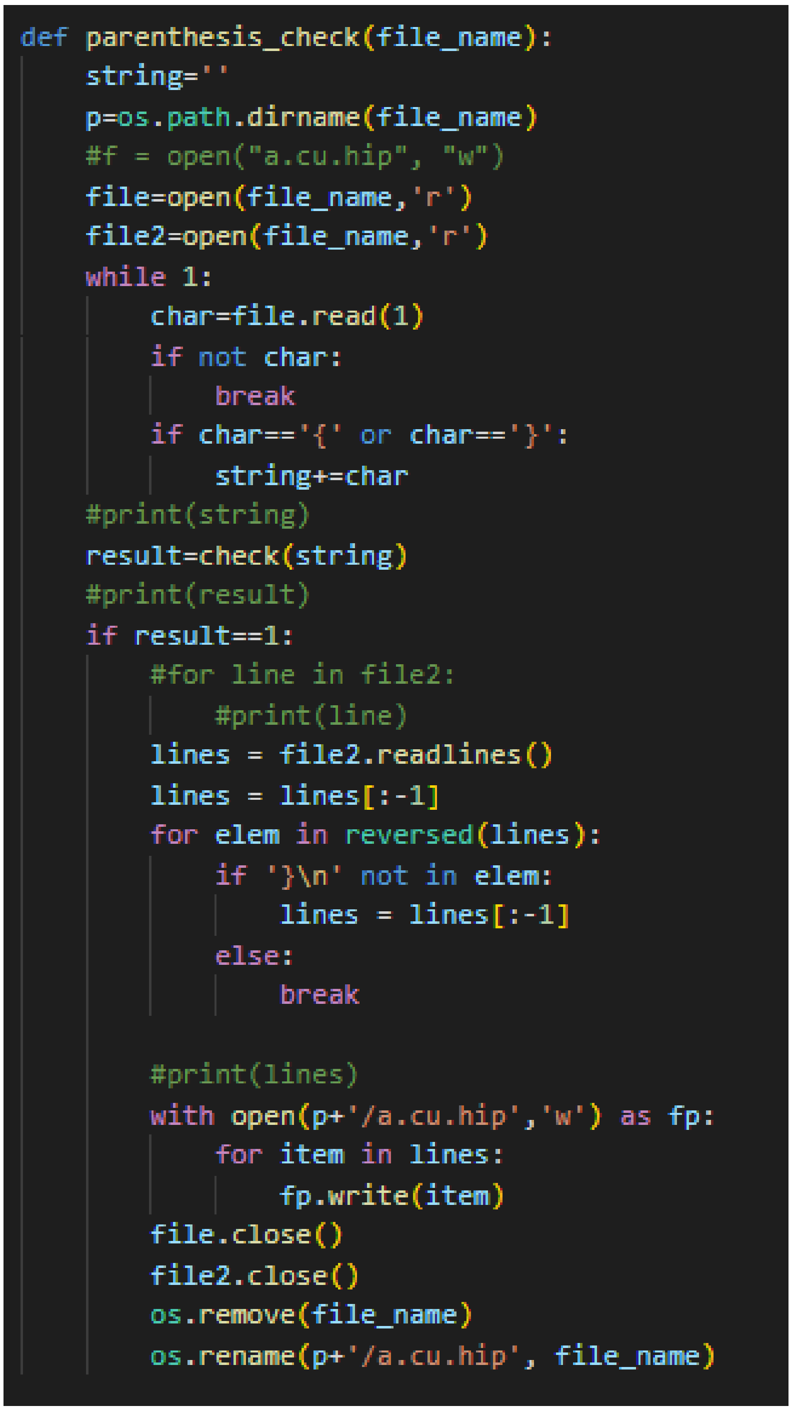


```python
def setup():
    global cuda_path
    #cuda_path = '/usr/local/cuda-12.0/targets/x86_64-linux/include'
    print ('Confirm the following CUDA Installation path for compilation:')
    print('CUDA Path:'+cuda_path)
    print('If Path is incorrect,please provide current path by typing CUDA or press any key to continue')
    user_input=input()
    if user_input.lower() == 'cuda':
        print('Enter path of your CUDA installation')
        cuda_path=input()
    os.system('gcc --version')
    print('Enter gcc to install gcc compiler, or any other button to continue.')
    user_input=input()
    if user_input.lower() == 'gcc':
        os.system('sudo apt install gcc')
    print("Enter 'omp' to install OpenMP in your system, or any other button to continue.")
    user_input=input()
    if user_input.lower() == 'omp':
        os.system('sudo apt install libomp-dev')
        os.system('echo |cpp -fopenmp -dM |grep -i open')
        print('Enter number of threads ')
        x=int(input())
        os.system('export OMP_NUM_THREADS='+str(x))
        print("Always add -fopenmp flag on compilation.")
    print("Enter 'mpi' to install OpenMPI, or any other button to continue.It's better to install latest version from this link manually:https://sites.google.com/site/rangsiman1993/comp-env/program-install/install-openmpi")
    user_input=input()
    if user_input.lower()=='mpi':
        print('cd ~')
        os.chdir(os.path.expanduser("~"))
        print('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        os.system('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        print('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('mv -r ')
        print('cd openmpi-3.1.3')
        os.system('cd openmpi-3.1.3')
        os.chdir('openmpi-3.1.3')
        print('pwd')
        os.system('pwd')
        print('./configure --prefix=/usr/local/')
        os.system('./configure --prefix=/usr/local/')
        print('./configure --prefix=/usr/local/openmpi-3.1.3/')
        os.system('./configure --prefix=/usr/local/openmpi-3.1.3/')
        print('sudo make all install')
        os.system('sudo make all install')
        print('After make install is completed, mpirun or orterun executable should be at /usr/local/bin/.')
        print('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        os.system('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        print('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" > $HOME/.bashrc')
        os.system('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" > $HOME/.bashrc')
        print('export PATH=$PATH:/usr/local/bin')
        os.system('export PATH=$PATH:/usr/local/bin')
        print('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        os.system('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        print('source $HOME/.bashrc')
        os.system('source $HOME/.bashrc')
        print('mpirun --version')
        os.system('mpirun --version')
```

This code sets up the environment for running parallel programs in a Linux system. The environment includes CUDA, GCC compiler, OpenMP, and OpenMPI. The user can specify the path for CUDA, install GCC compiler, OpenMP, and OpenMPI based on their needs. The code also checks the version of GCC compiler and sets the number of threads for OpenMP. For OpenMPI, the code downloads and installs the latest version from a specified link, sets the environment variables for PATH and LD_LIBRARY_PATH, and confirms the installation by checking the version of OpenMPI. The code uses the os library to execute shell commands, and input() for user interaction.

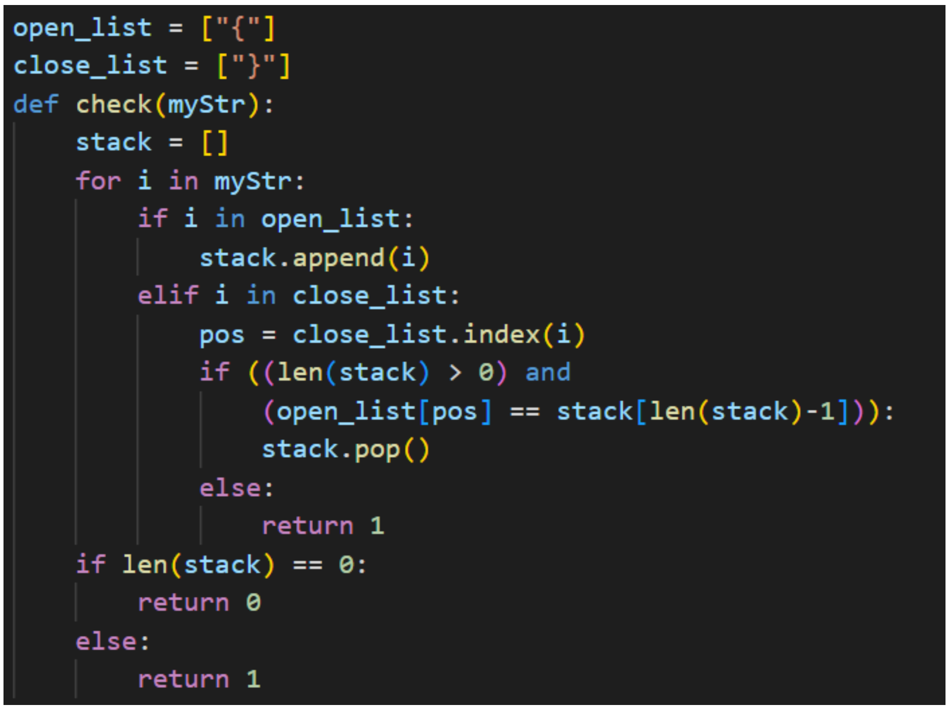


```python
def setup():
    global cuda_path
    #cuda_path = '/usr/local/cuda-12.0/targets/x86_64-linux/include'
    print ('Confirm the following CUDA Installation path for compilation:')
    print('CUDA Path:'+cuda_path)
    print('If Path is incorrect,please provide current path by typing CUDA or press any key to continue')
    user_input=input()
```



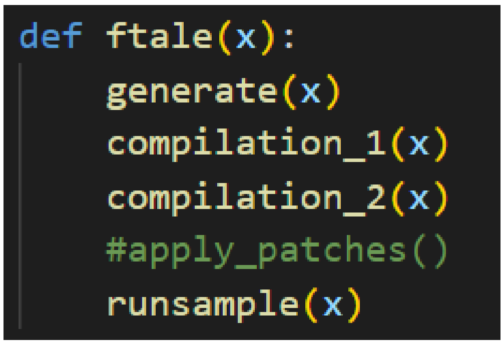
```python
def parenthesis_check(file_name):
    string=''
    p=os.path.dirname(file_name)
    #f = open("a.cu.hip", "w")
    file=open(file_name,'r')
    file2=open(file_name,'r')
    while 1:
        char=file.read(1)
        if not char:
            break
        if char=='{' or char=='}':
            string+=char
    #print(string)
    result=check(string)
    #print(result)
    if result==1:
        #for line in file2:
            #print(line)
        lines = file2.readlines()
        lines = lines[:-1]
        for elem in reversed(lines):
            if '}\n' not in elem:
                lines = lines[:-1]
            else:
                break

        #print(lines)
        with open(p+'/a.cu.hip','w') as fp:
            for item in lines:
                fp.write(item)
        file.close()
        file2.close()
        os.remove(file_name)
        os.rename(p+'/a.cu.hip', file_name)
```

The code is checking the correct use of curly braces "{" and "}" in a file and editing the file if necessary. It first reads the file and adds each curly brace into a string. Then, it calls the "check" function on the string, which verifies if the curly braces are used correctly. If they are, the code reads the file again, removes the last lines if they do not contain a closing curly brace, and writes the remaining lines to a new file with the same name but with a ".cu.hip" extension added to it. The original file is then deleted and replaced with the new file. The "parenthesis_check_all" function does the same process for all files in a directory specified by the argument "y", where only files with ".cu.hip" extension are processed.



```python
open_list = ["{"]
close_list = ["}"]
def check(myStr):
    stack = []
    for i in myStr:
        if i in open_list:
            stack.append(i)
        elif i in close_list:
            pos = close_list.index(i)
            if ((len(stack) > 0) and
                (open_list[pos] == stack[len(stack)-1])):
                stack.pop()
            else:
                return 1
    if len(stack) == 0:
        return 0
    else:
        return 1
```

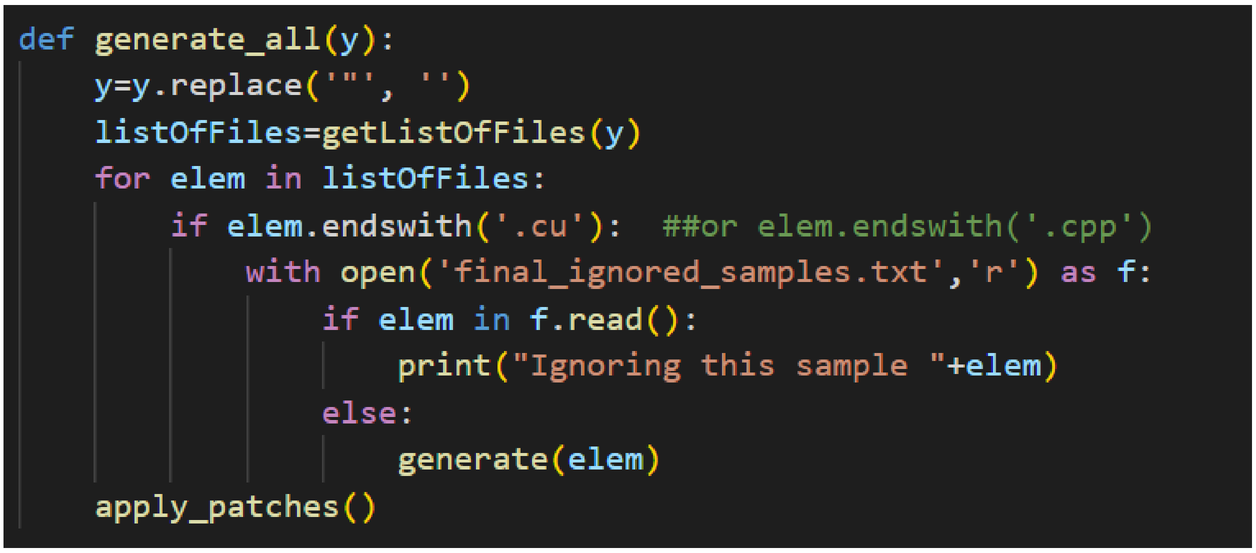
This code defines a function named "check" which takes a string input "myStr". The function checks if the string contains balanced parentheses. The open parentheses are represented by "{" and the close parentheses are represented by "}". The function uses a stack data structure to keep track of the opening parentheses. For each character in the input string, if it is an opening parenthesis, it is pushed onto the stack. If it is a closing parenthesis, the function checks if the last item on the stack is the corresponding opening parenthesis. If it is, it is popped off the stack. If the character is not a valid opening or closing parenthesis, the function returns 1. After processing all characters, if the stack is empty, the function returns 0, indicating a balanced parentheses. If the stack is not empty, the function returns 1, indicating unbalanced parentheses.



```python
def ftale(x):
    generate(x)
    compilation_1(x)
    compilation_2(x)
    #apply_patches()
    runsample(x)
```

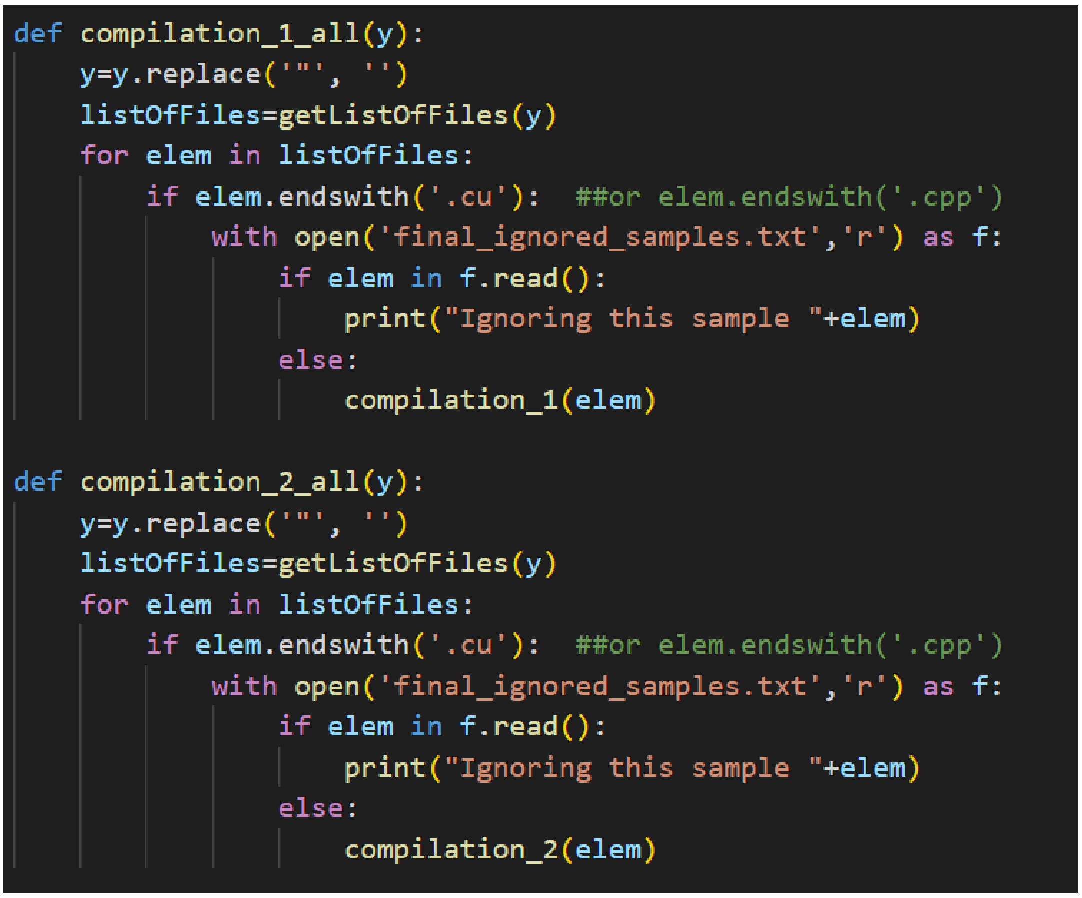
The code defines four functions: ftale(x), generate_all(y), compilation_1_all(y), and compilation_2_all(y).

ftale(x) function calls four other functions: generate(x), compilation_1(x), compilation_2(x), and runsample(x).



```python
def generate_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    generate(elem)
    apply_patches()
```

generate_all(y) function takes in a string y as an input, removes all quotes in the string, gets a list of files using getListOfFiles(y), and for each file in the list, if it ends with '.cu', it opens the 'final_ignored_samples.txt' file and checks if the current file is in it. If it is not in the 'final_ignored_samples.txt', the function generate(elem) is called with the current file as an argument. Finally, the function apply_patches() is called.



```python
def compilation_1_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    compilation_1(elem)

def compilation_2_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    compilation_2(elem)
```

compilation_1_all(y) and compilation_2_all(y) functions work similarly to generate_all(y), but call compilation_1(elem) and compilation_2(elem) respectively instead of generate(elem) for each file in the list if it is not in the 'final_ignored_samples.txt' file.

The code runs a function runsample_all that takes in a path y as an argument and performs the following actions:

Removes all the " characters from the y string.

Calls a function getListOfFiles(y) to get a list of files in the given path.

Loops through each file in the list and performs the following actions:

a. Checks if the file extension is .cu (or .cpp, but it's commented out).

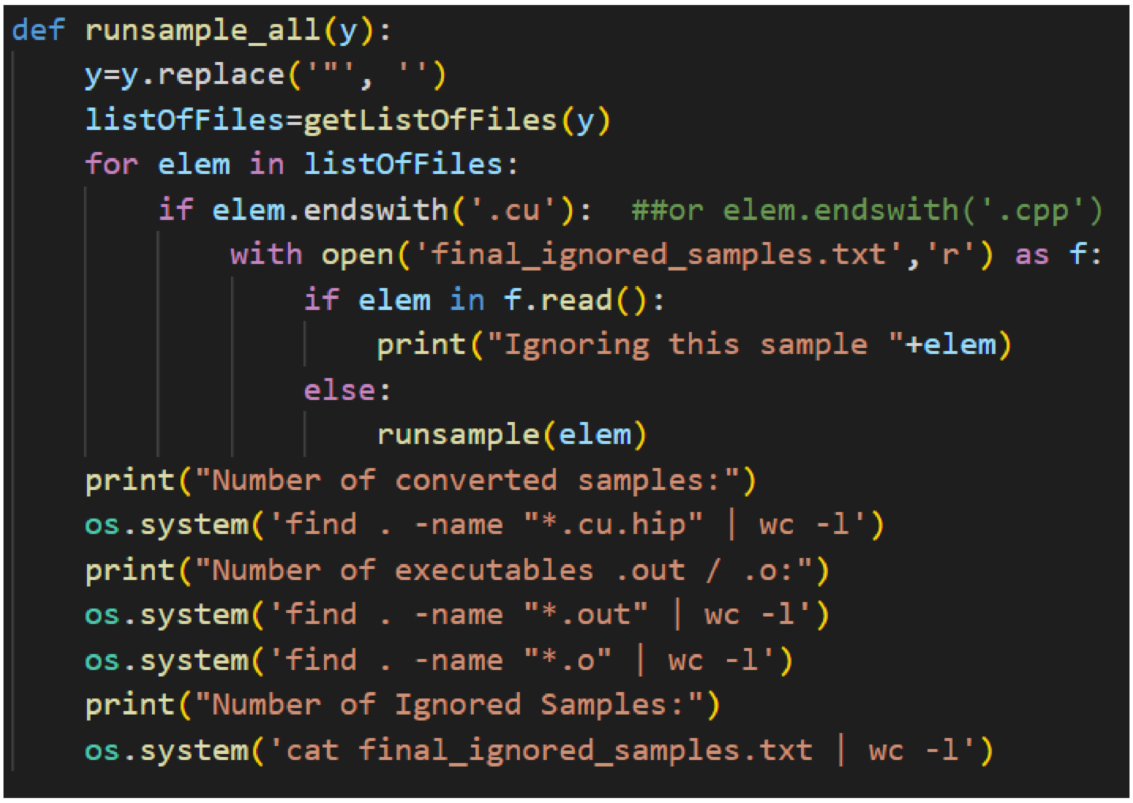
b. Opens a file final_ignored_samples.txt and checks if the current file is in it.

c. If the current file is not in the final_ignored_samples.txt file, it calls the runsample function and passes the current file as an argument.

Prints the number of converted samples (by running a shell command find . -name "*.cu.hip" | wc -l)

Prints the number of executables (.out or .o) by running shell commands find . -name "*.out" | wc -l and find . -name "*.o" | wc -l respectively.

Prints the number of ignored samples by running a shell command cat final_ignored_samples.txt | wc -l.



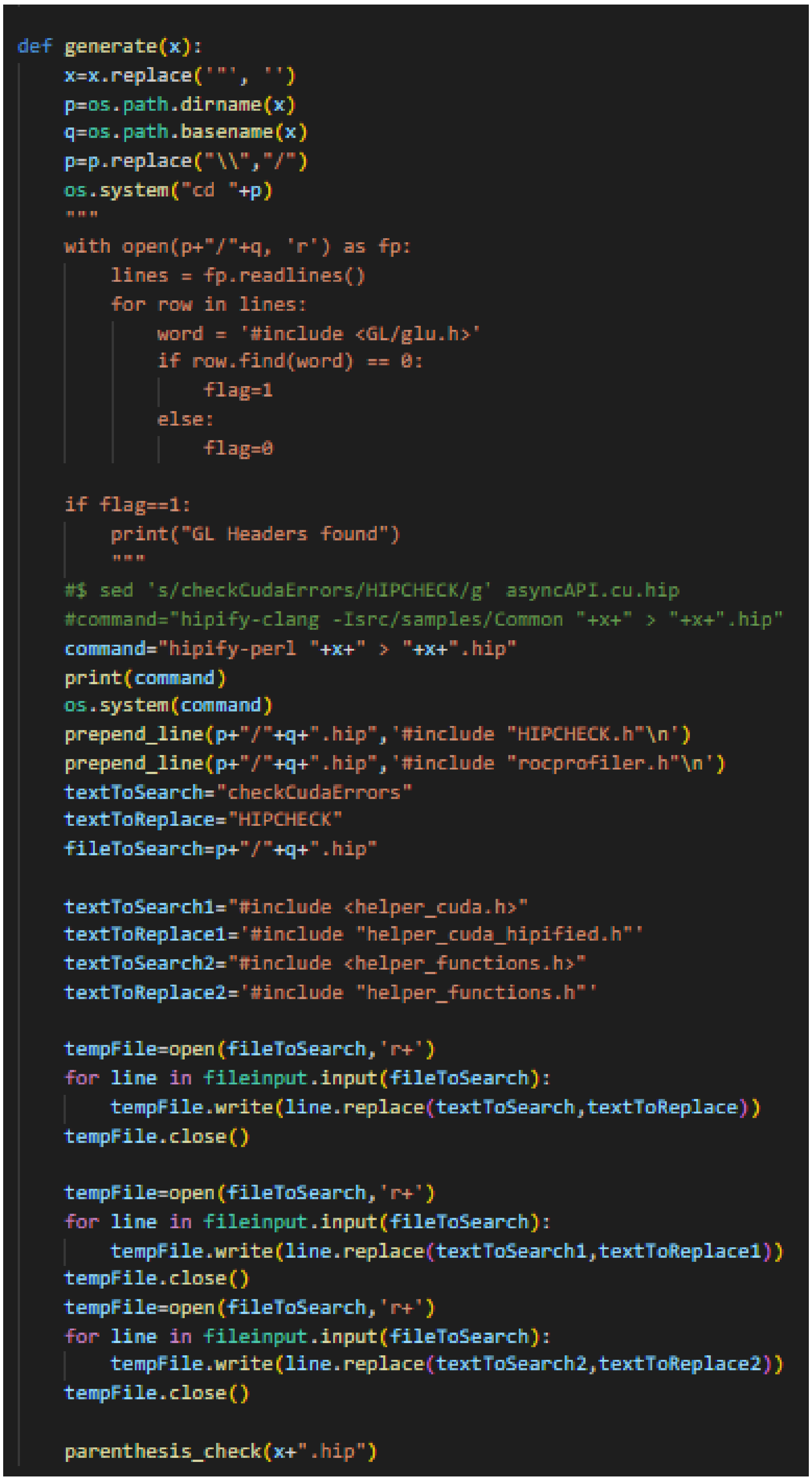
```python
def runsample_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    runsample(elem)
    print("Number of converted samples:")
    os.system('find . -name "*.cu.hip" | wc -l')
    print("Number of executables .out / .o:")
    os.system('find . -name "*.out" | wc -l')
    os.system('find . -name "*.o" | wc -l')
    print("Number of Ignored Samples:")
    os.system('cat final_ignored_samples.txt | wc -l')
```

The code defines a function generate that takes as input a file path x. It removes the double quotes from the file path, separates the directory path and the file name and replaces all the backslashes in the directory path with forward slashes.

It then runs a Perl script HIPIFY-perl on the input file to create a file with ".hip" extension. It then adds two lines at the beginning of the file, one includes the file "HIPCHECK.h" and the other includes "rocprofiler.h".

The code then performs three string replacements in the hipified file, changing "checkCudaErrors" to "HIPCHECK", "#include <helper_cuda.h>" to "#include "helper_cuda_hipified.h"", and "#include <helper_functions.h>" to "#include "helper_functions.h"".

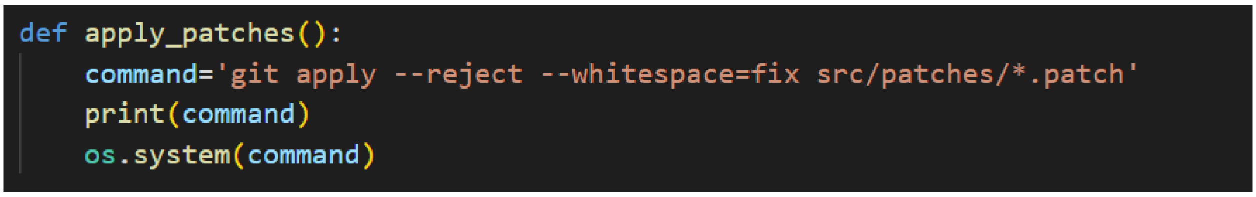
Finally, it calls a function parenthesis_check on the hipified file.



```python
def generate(x):
    x=x.replace('"', '')
    p=os.path.dirname(x)
    q=os.path.basename(x)
    p=p.replace("\\","/")
    os.system("cd "+p)
    """
    with open(p+"/"+q, 'r') as fp:
        lines = fp.readlines()
        for row in lines:
            word = '#include <GL/glu.h>'
            if row.find(word) == 0:
                flag=1
            else:
                flag=0

    if flag==1:
        print("GL Headers found")
        """
    #$ sed 's/checkCudaErrors/HIPCHECK/g' asyncAPI.cu.hip
    #command="hipify-clang -Isrc/samples/Common "+x+" > "+x+".hip"
    command="hipify-perl "+x+" > "+x+".hip"
    print(command)
    os.system(command)
    prepend_line(p+"/"+q+".hip",'#include "HIPCHECK.h"\n')
    prepend_line(p+"/"+q+".hip",'#include "rocprofiler.h"\n')
    textToSearch="checkCudaErrors"
    textToReplace="HIPCHECK"
    fileToSearch=p+"/"+q+".hip"

    textToSearch1="#include <helper_cuda.h>"
    textToReplace1='#include "helper_cuda_hipified.h"'
    textToSearch2="#include <helper_functions.h>"
    textToReplace2='#include "helper_functions.h"'

    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch,textToReplace))
    tempFile.close()

    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch1,textToReplace1))
    tempFile.close()
    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch2,textToReplace2))
    tempFile.close()

    parenthesis_check(x+".hip")
```

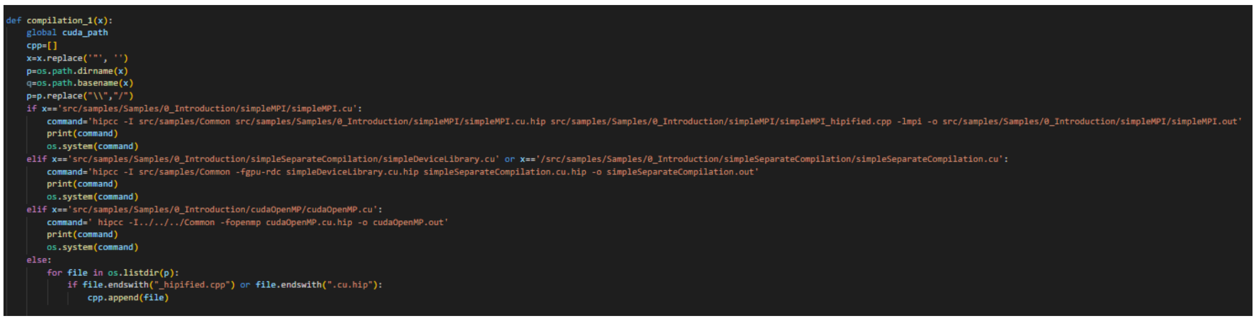
This is a python function named apply_patches(). It runs a shell command using os.system() method. The shell command git apply --reject --whitespace=fix src/patches/*.patch is executed.

This shell command applies patches located in the src/patches/ directory with the .patch extension, and the option --reject is used to write rejected hunks to files with a .rej extension. The --whitespace=fix option is used to fix white space errors in patch files. The purpose of the function is to apply patches to the files in the system using Git.



```python
def apply_patches():
    command='git apply --reject --whitespace=fix src/patches/*.patch'
    print(command)
    os.system(command)
```

The code is a python script that compiles CUDA files into executables using the hipcc command line tool. There are two functions, compilation_1 and compilation_2 that take a file path to a CUDA file as input and compile it into an executable.



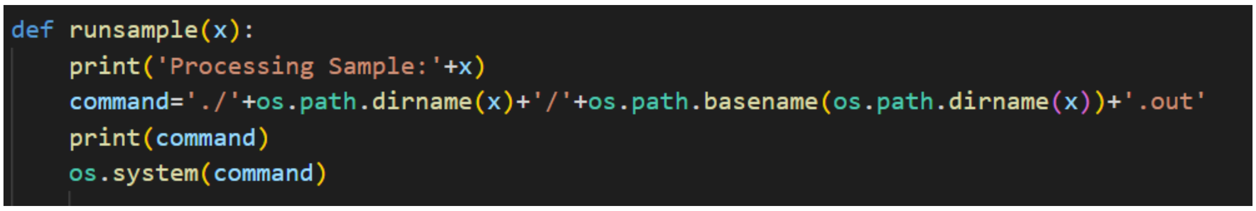
```python
def compilation_1(x):
    global cuda_path
    cpp=[]
    x=x.replace('"', '')
    p=os.path.dirname(x)
    q=os.path.basename(x)
    p=p.replace("\\","/")
    if x=='src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu':
        command='hipcc -I src/samples/Common src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu.hip src/samples/Samples/0_Introduction/simpleMPI/simpleMPI_hipified.cpp -lmpi -o src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.out'
        print(command)
        os.system(command)
    elif x=='src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleDeviceLibrary.cu' or x=='/src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleSeparateCompilation.cu':
        command='hipcc -I src/samples/Common -fgpu-rdc simpleDeviceLibrary.cu.hip simpleSeparateCompilation.cu.hip -o simpleSeparateCompilation.out'
        print(command)
        os.system(command)
    elif x=='src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu':
        command=' hipcc -I../../../Common -fopenmp cudaOpenMP.cu.hip -o cudaOpenMP.out'
        print(command)
        os.system(command)
    else:
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.hip"):
                cpp.append(file)
```

Both functions handle different cases of CUDA file paths, where each case is specified with an if statement. The commands to compile the file into an executable are stored in a string command and then executed using the os.system function.

The compilation_1 function compiles CUDA files without the -use-staticlib flag, whereas the compilation_2 function compiles with the flag. If the file path is either "src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu", "src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleDeviceLibrary.cu", or "src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu", the respective command string is constructed and executed.

Note:It is possible for multiple samples and programs to be linked by a single executable.

For other file paths, the function lists all the files in the directory specified by the path, adds files that end with either "_hipified.cpp" or ".cu.hip" to a list cpp, and then compiles the list of files into an executable. The command string is constructed by concatenating all elements in cpp separated by a space and adding the -I and -o flags.



```python
def runsample(x):
    print('Processing Sample:'+x)
    command='./'+os.path.dirname(x)+'/'+os.path.basename(os.path.dirname(x))+'.out'
    print(command)
    os.system(command)
```

The code defines a function runsample that takes in a string argument x. The function then prints a string "Processing Sample: " followed by the value of x.

The function then defines a variable command which is set to the result of concatenating several strings:

"./" - a string with a period and a forward slash os.path.dirname(x) - the directory component of the file path specified by x

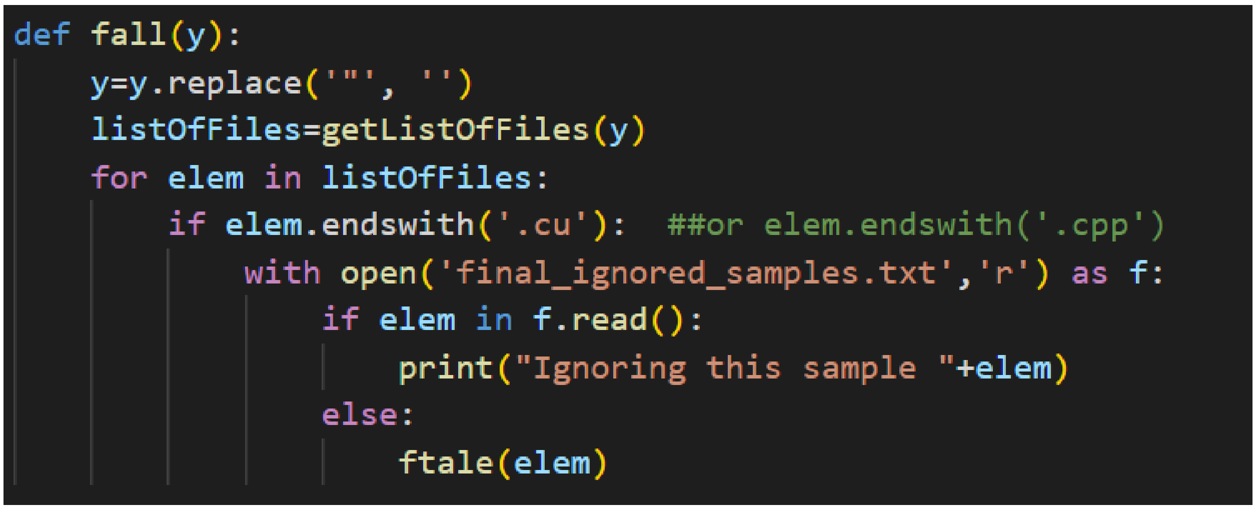
"/" - a forward slash

os.path.basename(os.path.dirname(x)) - the base name of the directory component of the file path specified by x

".out" - a string with the extension ".out"

The value of the command variable is then printed.

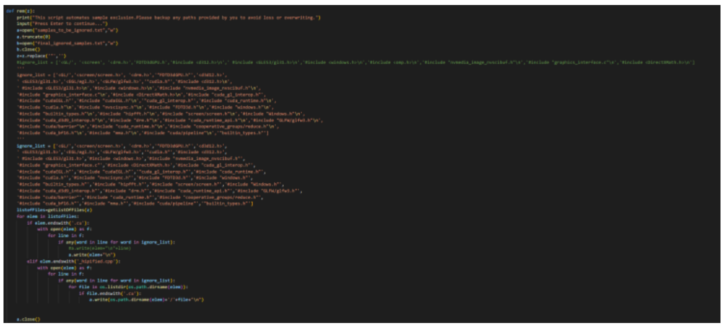
Finally, the function calls os.system(command), which is a function that runs the command specified as a string in the shell. This will run the file specified by command.



```
def fall(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    ftale(elem)
```

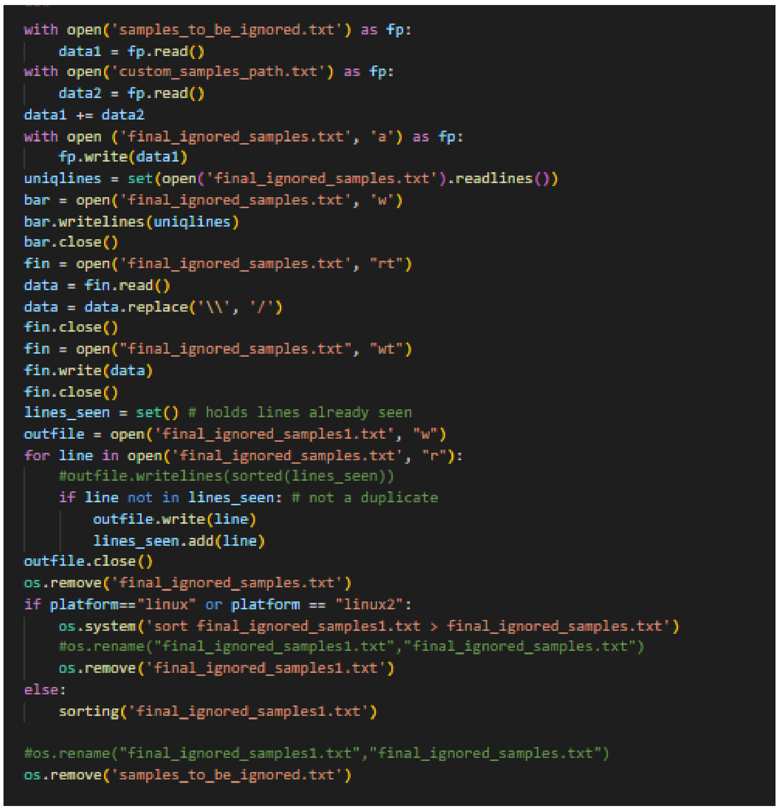
The code defines a Python function "fall(y)" that takes a string argument "y".The string "y" has its double-quote characters removed using the replace() method.The getListOfFiles() function is called with the modified string "y" as its argument, and the result is stored in a variable "listOfFiles".The function then enters a for-loop that iterates over each element in the list "listOfFiles".

For each iteration, the code checks if the current element ends with '.cu' using the endswith() method.If the current element ends with '.cu', the code opens the file 'final_ignored_samples.txt' in read mode, and uses the in operator to check if the current element is contained in the file.If the current element is contained in the file, the code prints a message indicating that this sample is being ignored.If the current element is not contained in the file, the code calls another function ftale() with the current element as its argument.



```
def rem(z):
    print("This script automates sample exclusion.Please backup any paths provided by you to avoid loss or overwriting.")
    input("Press Enter to continue...")
    a=open("samples_to_be_ignored.txt","w")
    a.truncate(0)
    b=open("final_ignored_samples.txt","w")
    b.close()
    z=z.replace('"','')
    #ignore_list = ['<GL/', '<screen', '<drm.h>','FDTD3dGPU.h','#include <d3d12.h>\n',' #include <GLES3/gl31.h>\n','#include <windows.h>\n','#include <omp.h>\n','#include "nvmedia_image_nvscibuf.h"\n','#include "graphics_interface.c"\n','#include <DirectXMath.h>\n']
    '''
    ignore_list = ['<GL/','<screen/screen.h>', '<drm.h>','"FDTD3dGPU.h"','<d3d12.h>',
    ' <GLES3/gl31.h>','<EGL/egl.h>','<GLFW/glfw3.h>','"cudla.h"','#include <d3d12.h>\n',
    ' #include <GLES3/gl31.h>\n','#include <windows.h>\n','#include "nvmedia_image_nvscibuf.h"\n',
    '#include "graphics_interface.c"\n','#include <DirectXMath.h>\n','#include "cuda_gl_interop.h"',
    '#include "cudaEGL.h"','#include "cudaEGL.h"\n','"cuda_gl_interop.h"','#include "cuda_runtime.h"\n',
    '#include "cudla.h"\n','#include "nvscisync.h"\n','#include "FDTD3d.h"\n','#include "windows.h"\n',
    '#include "builtin_types.h"\n','#include "hipfft.h"\n','#include "screen/screen.h"\n','#include "Windows.h"\n',
    '#include "cuda_d3d9_interop.h"\n','#include "drm.h"\n','#include "cuda_runtime_api.h"\n','#include "GLFW/glfw3.h"\n',
    '#include "cuda/barrier"\n','#include "cuda_runtime.h"\n','#include "cooperative_groups/reduce.h"\n',
    '#include "cuda_bf16.h"\n','#include "mma.h"\n','#include "cuda/pipeline"\n','"builtin_types.h"']
    '''
    ignore_list = ['<GL/','<screen/screen.h>', '<drm.h>','"FDTD3dGPU.h"','<d3d12.h>',
    ' <GLES3/gl31.h>','<EGL/egl.h>','<GLFW/glfw3.h>','"cudla.h"','#include <d3d12.h>',
    ' #include <GLES3/gl31.h>','#include <windows.h>','#include "nvmedia_image_nvscibuf.h"',
    '#include "graphics_interface.c"','#include <DirectXMath.h>','#include "cuda_gl_interop.h"',
    '#include "cudaEGL.h"','#include "cudaEGL.h"','"cuda_gl_interop.h"','#include "cuda_runtime.h"',
    '#include "cudla.h"','#include "nvscisync.h"','#include "FDTD3d.h"','#include "windows.h"',
    '#include "builtin_types.h"','#include "hipfft.h"','#include "screen/screen.h"','#include "Windows.h"',
    '#include "cuda_d3d9_interop.h"','#include "drm.h"','#include "cuda_runtime_api.h"','#include "GLFW/glfw3.h"',
    '#include "cuda/barrier"','#include "cuda_runtime.h"','#include "cooperative_groups/reduce.h"',
    '#include "cuda_bf16.h"','#include "mma.h"','#include "cuda/pipeline"','"builtin_types.h"']
    listOfFiles=getListOfFiles(z)
    for elem in listOfFiles:
        if elem.endswith('.cu'):
            with open(elem) as f:
                for line in f:
                    if any(word in line for word in ignore_list):
                        #a.write(elem+"\n"+line)
                        a.write(elem+"\n")
        elif elem.endswith('_hipified.cpp'):
            with open(elem) as f:
                for line in f:
                    if any(word in line for word in ignore_list):
                        for file in os.listdir(os.path.dirname(elem)):
                            if file.endswith('.cu'):
                                a.write(os.path.dirname(elem)+'/'+file+"\n")


    a.close()
```

This script is an automated sample exclusion tool for rem function. It prints a message to the user to backup any paths they provide to avoid loss or overwriting. The user is prompted to press Enter to continue. The script then opens two files "samples_to_be_ignored.txt" and "final_ignored_samples.txt", the first file is truncated, and the second file is closed. The input "z" is passed to the function, and any quotes in it are replaced. A list of ignore_list items is defined, which includes certain strings such as include statements for various header files. A list of files is obtained by calling the function getListOfFiles(z) on the input "z". The list of files is then processed, and if any of the files ends with '.cu' or '_hipified.cpp', they are opened, and the contents are searched for any words in the ignore_list. If a match is found, the file name is written to "samples_to_be_ignored.txt". The contents of "samples_to_be_ignored.txt" and "custom_samples_path.txt" are combined and written to "final_ignored_samples.txt". The file is then processed to exclude duplicate entries, and all backslashes in the file are replaced with forward slashes. Finally, the file is processed to exclude duplicates, and the final output is written to "final_ignored_samples1.txt".



```python
with open('samples_to_be_ignored.txt') as fp:
    data1 = fp.read()
with open('custom_samples_path.txt') as fp:
    data2 = fp.read()
data1 += data2
with open ('final_ignored_samples.txt', 'a') as fp:
    fp.write(data1)
uniqlines = set(open('final_ignored_samples.txt').readlines())
bar = open('final_ignored_samples.txt', 'w')
bar.writelines(uniqlines)
bar.close()
fin = open('final_ignored_samples.txt', "rt")
data = fin.read()
data = data.replace('\\', '/')
fin.close()
fin = open("final_ignored_samples.txt", "wt")
fin.write(data)
fin.close()
lines_seen = set() # holds lines already seen
outfile = open('final_ignored_samples1.txt', "w")
for line in open('final_ignored_samples.txt', "r"):
    #outfile.writelines(sorted(lines_seen))
    if line not in lines_seen: # not a duplicate
        outfile.write(line)
        lines_seen.add(line)
outfile.close()
os.remove('final_ignored_samples.txt')
if platform=="linux" or platform == "linux2":
    os.system('sort final_ignored_samples1.txt > final_ignored_samples.txt')
    #os.rename("final_ignored_samples1.txt","final_ignored_samples.txt")
    os.remove('final_ignored_samples1.txt')
else:
    sorting('final_ignored_samples1.txt')

#os.rename("final_ignored_samples1.txt","final_ignored_samples.txt")
os.remove('samples_to_be_ignored.txt')
```

List of Ignored Samples are as follows:

No	SAMPLES	REASON FOR EXCLUSION
1	src/samples/Samples/0_Introduction/simpleAssert_nvrtc/simpleAssert_kernel.cu	#include <cuda.h>
2	src/samples/Samples/0_Introduction/simpleCUDA2GL/simpleCUDA2GL.cu	#include <cuda_gl_interop.h>
3	src/samples/Samples/0_Introduction/simpleTexture3D/simpleTexture3D_kernel.cu	#include <cuda_gl_interop.h>

4	src/samples/Samples/2_Concepts_and_Techniques/EGLStream_CUDA_CrossGPU/kernel.cu	#include "cudaEGL.h"
5	src/samples/Samples/2_Concepts_and_Techniques/EGLSync_CUDAEvent_Interop/EGLSync_CUDAEvent_Interop.cu	#include <cudaEGL.h>
6	src/samples/Samples/2_Concepts_and_Techniques/FunctionPointers/FunctionPointers_kernels.cu	<cuda_gl_interop.h>
7	src/samples/Samples/2_Concepts_and_Techniques/boxFilter/boxFilter_kernel.cu	<cuda_gl_interop.h>
8	src/samples/Samples/2_Concepts_and_Techniques/imageDenoising/imageDenoising.cu	<cuda_gl_interop.h>
9	src/samples/Samples/2_Concepts_and_Techniques/particles/particleSystem_cuda.cu	#include <cuda_gl_interop.h>
10	src/samples/Samples/3_CUDA_Features/bindlessTexture/bindlessTexture_kernel.cu	#include "cuda_runtime.h"
11	src/samples/Samples/4_CUDA_Libraries/cuDLAErrorReporting/main.cu	#include "cudla.h"
12	src/samples/Samples/4_CUDA_Libraries/cuDLAHybridMode/main.cu	#include "cudla.h"
13	src/samples/Samples/4_CUDA_Libraries/cudaNvSciNvMedia/cuda_consumer.cu	#include <nvscisync.h>
14	src/samples/Samples/4_CUDA_Libraries/oceanFFT/oceanFFT_kernel.cu	#include <cuda_gl_interop.h>
15	src/samples/Samples/5_Domain_Specific/FDTD3d/src/FDTD3dGPU.cu	#include "FDTD3d.h"
16	src/samples/Samples/5_Domain_Specific/Mandelbrot/Mandelbrot_cuda.cu	#include <cuda_gl_interop.h>

17	src/samples/Samples/5_Domain_Specific/SLID3D10Texture/texture_2d.cu	#include <windows.h>
18	src/samples/Samples/5_Domain_Specific/SobelFilter/SobelFilter_kernels.cu	#include <cuda_gl_interop.h>
19	src/samples/Samples/5_Domain_Specific/VFlockingD3D10/VFlocking_kernel.cu	#include "cuda_runtime.h"
20	src/samples/Samples/5_Domain_Specific/bicubicTexture/bicubicTexture_cuda.cu	#include <cuda_gl_interop.h>
21	src/samples/Samples/5_Domain_Specific/bilateralFilter/bilateral_kernel.cu	#include "cuda_runtime.h"
22	src/samples/Samples/5_Domain_Specific/fluidsD3D9/fluidsD3D9_kernels.cu	#include <builtin_types.h>
23	src/samples/Samples/5_Domain_Specific/fluidsGL/fluidsGL_kernels.cu	#include <hipfft.h>
24	src/samples/Samples/5_Domain_Specific/fluidsGLES/fluidsGLES_kernels.cu	#include <hipfft.h>
25	src/samples/Samples/5_Domain_Specific/marchingCubes/marchingCubes_kernel.cu	#include "cuda_runtime.h"
26	src/samples/Samples/5_Domain_Specific/nbody/bodysystemcuda.cu	#include <cuda_gl_interop.h>
27	src/samples/Samples/5_Domain_Specific/nbody_opengles/bodysystemcuda.cu	#include <cuda_gl_interop.h>
28	src/samples/Samples/5_Domain_Specific/nbody_screen/bodysystemcuda.cu	#include <screen/screen.h>
29	src/samples/Samples/5_Domain_Specific/postProcessGL/postProcessGL.cu	#include <cuda_gl_interop.h>

30	src/samples/Samples/5_Domain_Specific/recursiveGaussian/recursiveGaussian_cuda.cu	#include "cuda_runtime.h"
31	src/samples/Samples/5_Domain_Specific/simpleD3D10/simpleD3D10_kernel.cu	#include <builtin_types.h>
32	src/samples/Samples/5_Domain_Specific/simpleD3D10RenderTarget/simpleD3D10RenderTarget_kernel.cu	#include <builtin_types.h>
33	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_2d.cu	#include <windows.h>
34	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_3d.cu	#include <windows.h>
35	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_cube.cu	#include <windows.h>
36	src/samples/Samples/5_Domain_Specific/simpleD3D11/sinewave_cuda.cu	#include <windows.h>
37	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_2d.cu	#include <windows.h>
38	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_3d.cu	#include <windows.h>
39	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_cube.cu	#include <windows.h>
40	src/samples/Samples/5_Domain_Specific/simpleD3D12/sinewave_cuda.cu	#include <windows.h>
41	src/samples/Samples/5_Domain_Specific/simpleD3D9/simpleD3D9_kernel.cu	#include <Windows.h>
42	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_2d.cu	#include <cuda_d3d9_interop.h>

43	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_cube.cu	#include <cuda_d3d9_interop.h>
44	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_volume.cu	#include <cuda_d3d9_interop.h>
45	src/samples/Samples/5_Domain_Specific/simpleGL/simpleGL.cu	#include <cuda_gl_interop.h>
46	src/samples/Samples/5_Domain_Specific/simpleGLES/simpleGLES.cu	#include <cuda_gl_interop.h>
47	src/samples/Samples/5_Domain_Specific/simpleGLES_EGLOutput/simpleGLES_EGLOutput.cu	#include <drm.h>
48	src/samples/Samples/5_Domain_Specific/simpleGLES_screen/simpleGLES_screen.cu	#include <screen/screen.h>
49	src/samples/Samples/5_Domain_Specific/simpleVulkan/SineWaveSimulation.cu	#include <cuda_runtime_api.h>
50	src/samples/Samples/5_Domain_Specific/simpleVulkanMMAP/MonteCarloPi.cu	#include <cuda_runtime_api.h>
51	src/samples/Samples/5_Domain_Specific/smokeParticles/ParticleSystem_cuda.cu	#include <cuda_gl_interop.h>
52	src/samples/Samples/5_Domain_Specific/volumeFiltering/volumeFilter_kernel.cu	#include "cuda_runtime.h"
53	src/samples/Samples/5_Domain_Specific/volumeFiltering/volumeRender_kernel.cu	#include "cuda_runtime.h"
54	src/samples/Samples/5_Domain_Specific/volumeRender/volumeRender_kernel.cu	#include <cuda_gl_interop.h>
55	src/samples/Samples/5_Domain_Specific/vulkanImageCUDA/vulkanImageCUDA.cu	#include <GLFW/glfw3.h>

56	src/samples/Samples/0_Introduction/c++11_cuda/c++11_cuda.cu	#include "cuda_runtime.h"
57	src/samples/Samples/0_Introduction/cppOverload/cppOverload.cu	#include <builtin_types.h>
58	src/samples/Samples/0_Introduction/matrixMulDrv/matrixMul_kernel.cu	#include <cuda.h>
59	src/samples/Samples/0_Introduction/matrixMul_nvrtc/matrixMul_kernel.cu	#include <cuda/barrier>
60	src/samples/Samples/0_Introduction/simpleAWBarrier/simpleAWBarrier.cu*	ROCm Deletion Bug
61	src/samples/Samples/0_Introduction/simpleAttributes/simpleAttributes.cu	#include <builtin_types.h>
62	src/samples/Samples/0_Introduction/simpleDrvRuntime/vectorAdd_kernel.cu	#include <cuda.h>
63	src/samples/Samples/0_Introduction/simpleTemplates_nvrtc/simpleTemplates_kernel.cu	#include <cuda_gl_interop.h>
64	src/samples/Samples/0_Introduction/simpleTexture3D/simpleTexture3D_kernel.cu	#include <builtin_types.h>
65	src/samples/Samples/0_Introduction/simpleTextureDrv/simpleTexture_kernel.cu	#include <cuda_gl_interop.h>
66	src/samples/Samples/0_Introduction/simpleVoteIntrinsics/simpleVoteIntrinsics.cu*	use of undeclared identifier '__any_sync'
67	src/samples/Samples/0_Introduction/systemWideAtomics/systemWideAtomics.cu*	Sample compiles successfully but causes ROCm deletion bug.
68	src/samples/Samples/0_Introduction/vectorAddMMAP/vectorAdd_kernel.cu	#include <cuda.h>
69	src/samples/Samples/0_Introduction/vectorAdd_nvrtc/vectorAdd_kernel.cu	#include <cuda.h>

70	src/samples/Samples/0_Introduction/vectorAdd_nvrtc/vectorAdd_nvrtc.cu	#include <cuda.h>
71	src/samples/Samples/2_Concepts_and_Techniques/EGLStream_CUDA_CrossGPU/kernel.cu	#include "cudaEGL.h"
72	src/samples/Samples/2_Concepts_and_Techniques/EGLSync_CUDAEvent_Interop/EGLSync_CUDAEvent_Interop.cu	#include "cudaEGL.h"
73	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiInlineP/src/piestimator.cu	#include <hiprand_kernel.h>
74	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiQ/src/piestimator.cu	#include <hiprand.h>
75	src/samples/Samples/2_Concepts_and_Techniques/convolutionTexture/convolutionTexture.cu	#include <cuda_runtime.h>
76	src/samples/Samples/2_Concepts_and_Techniques/dct8x8/dct8x8.cu	#include <cuda_runtime.h>
77	src/samples/Samples/2_Concepts_and_Techniques/inlinePTX/inlinePTX.cu	ASM Language
78	src/samples/Samples/2_Concepts_and_Techniques/inlinePTX_nvrtc/inlinePTX_kernel.cu	ASM Language
79	src/samples/Samples/2_Concepts_and_Techniques/particles/particleSystem_cuda.cu	#include <cuda_gl_interop.h>
80	src/samples/Samples/2_Concepts_and_Techniques/reductionMultiBlockCG/reductionMultiBlockCG.cu	#include <cooperative_groups/reduce.h>
81	src/samples/Samples/2_Concepts_and_Techniques/shfl_scan/shfl_scan.cu*	Unsupported shfl_sync functions.
82	src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationIPC/streamOrderedAllocationIPC.cu*	unknown type name 'CUresult', unsupported identifier "CU_DEVICE_ATTRIBUTE_HANDLE_TYP

		E_POSIX_FILE_DESCRIPTOR_SUPPORTED" unsupported identifier "CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_WIN32_HANDLE_SUPPORTED"
83	src/samples/Samples/3_CUDA_Features/bf16TensorCoreGemm/bf16TensorCoreGemm.cu	#include <cuda_bf16.h>
84	src/samples/Samples/3_CUDA_Features/binaryPartitionCG/binaryPartitionCG.cu	#include <cooperative_groups/reduce.h>
85	src/samples/Samples/3_CUDA_Features/cdpAdvancedQuicksort/cdpAdvancedQuicksort.cu*	call to __host__ function from __global__ function
86	src/samples/Samples/3_CUDA_Features/cdpAdvancedQuicksort/cdpBitonicSort.cu*	call to __host__ function from __global__ function
87	src/samples/Samples/3_CUDA_Features/cdpQuadtree/cdpQuadtree.cu*	no member named 'any' and ‘ballot’ in 'cooperative_groups
88	src/samples/Samples/3_CUDA_Features/cdpSimplePrint/cdpSimplePrint.cu*	reference to __global__ function 'cdp_kernel' in __global__ function, use of undeclared identifier 'checkCmdLineFlag'
89	src/samples/Samples/5_Domain_Specific/nbody/bodysystemcuda.cu	#include <cuda_gl_interop.h>
90	src/samples/Samples/5_Domain_Specific/quasirandomGenerator_nvrtc/quasirandomGenerator_kernel.cu	#include <cuda.h>
91	src/samples/Samples/5_Domain_Specific/recursiveGaussian/recursiveGaussian_cuda.cu	#include "cuda_runtime.h"
92	src/samples/Samples/5_Domain_Specific/simpleD3D10/simpleD3D10_kernel.cu	#include <builtin_types.h>
93	src/samples/Samples/5_Domain_Specific/simpleD3D10RenderTarget/simpleD3D10RenderTarget_kernel.cu	#include <builtin_types.h>
94	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_cube.cu	#include <windows.h>

95	src/samples/Samples/5_Domain_Specific/simpleD3D11/sinewave_cuda.cu	#include <windows.h>
96	src/samples/Samples/5_Domain_Specific/simpleD3D12/sinewave_cuda.cu	#include <windows.h>
97	src/samples/Samples/5_Domain_Specific/simpleGL/simpleGL.cu	#include <cuda_gl_interop.h>
98	src/samples/Samples/5_Domain_Specific/simpleVulkan/SineWaveSimulation.cu	#include <cuda_runtime_api.h>
99	src/samples/Samples/5_Domain_Specific/simpleVulkanMMAP/MonteCarloPi.cu	#include <cuda_runtime_api.h>
100	src/samples/Samples/5_Domain_Specific/stereoDisparity/stereoDisparity.cu	ASM Language
101	/src/samples/Samples/3_CUDA_Features/cdpBezierTessellation/BezierLineCDP.cu*	no member named 'any' and ‘ballot’ in 'cooperative_groups
102	src/samples/Samples/3_CUDA_Features/cdpSimpleQuicksort/cdpSimpleQuicksort.cu*	call to __host__ function from __global__ function
103	src/samples/Samples/3_CUDA_Features/cudaTensorCoreGemm/cudaTensorCoreGemm.cu	#include <mma.h>
104	src/samples/Samples/3_CUDA_Features/dmmaTensorCoreGemm/dmmaTensorCoreGemm.cu	#include <mma.h>
105	src/samples/Samples/3_CUDA_Features/globalToShmemAsyncCopy/globalToShmemAsyncCopy.cu	#include <cuda/pipeline>
106	src/samples/Samples/3_CUDA_Features/graphMemoryFootprint/graphMemoryFootprint.cu*	unsupported identifier "cudaGraphAddMemFreeNode","cudaMemAllocNodeParams"
107	src/samples/Samples/3_CUDA_Features/graphMemoryNodes/graphMemoryNodes.cu	use of undeclared identifier 'cudaGraphAddMemFreeNode, unknown type name 'cudaMemAllocNodeParams'

108	src/samples/Samples/3_CUDA_Features/immaTensorCoreGemm/immaTensorCoreGemm.cu	#include <mma.h>
109	src/samples/Samples/3_CUDA_Features/memMapIPCDrv/memMapIpc_kernel.cu	<builtin_types.h>
110	src/samples/Samples/3_CUDA_Features/ptxjit/ptxjit_kernel.cu*	No equivalent function for findCudaDeviceDRV
111	src/samples/Samples/0_Introduction/vectorAddDrv/vectorAdd_kernel.cu	#include <builtin_types.h>
112	src/samples/Samples/3_CUDA_Features/tf32TensorCoreGemm/tf32TensorCoreGemm.cu	#include <mma.h>
113	src/samples/Samples/3_CUDA_Features/warpAggregatedAtomicsCG/warpAggregatedAtomicsCG.cu	#include <cooperative_groups/reduce.h>
114	src/samples/Samples/4_CUDA_Libraries/conjugateGradientCudaGraphs/conjugateGradientCudaGraphs.cu	#include <hipblas.h>
115	src/samples/Samples/4_CUDA_Libraries/conjugateGradientMultiBlockCG/conjugateGradientMultiBlockCG.cu	#include <cooperative_groups/reduce.h>
116	src/samples/Samples/4_CUDA_Libraries/conjugateGradientMultiDeviceCG/conjugateGradientMultiDeviceCG.cu	#include <cooperative_groups/reduce.h>
117	src/samples/Samples/4_CUDA_Libraries/cudaNvSci/imageKernels.cu	#include <cuda_runtime.h>
118	src/samples/Samples/4_CUDA_Libraries/cudaNvSciNvMedia/cuda_consumer.cu	#include <nvscisync.h>
119	src/samples/Samples/4_CUDA_Libraries/lineOfSight/lineOfSight.cu	#include <cuda_runtime.h>

120	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT/simpleCUFFT.cu	#include <hipfft.h>
121	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_2d_MGPU/simpleCUFFT_2d_MGPU.cu	#include <hipfftXt.h>
122	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_MGPU/simpleCUFFT_MGPU.cu	#include <hipfftXt.h>
123	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_callback/simpleCUFFT_callback.cu	#include <hipfft.h>
124	src/samples/Samples/5_Domain_Specific/Mandelbrot/Mandelbrot_cuda.cu	#include <cuda_gl_interop.h>
125	src/samples/Samples/5_Domain_Specific/MonteCarloMultiGPU/MonteCarlo_kernel.cu	#include <hiprand_kernel.h>
126	src/samples/Samples/5_Domain_Specific/SobelFilter/SobelFilter_kernels.cu	#include <cuda_gl_interop.h>
127	src/samples/Samples/5_Domain_Specific/VFlockingD3D10/VFlocking_kernel.cu	#include "cuda_runtime.h"
128	src/samples/Samples/5_Domain_Specific/bilateralFilter/bilateral_kernel.cu	#include "cuda_runtime.h"
129	src/samples/Samples/5_Domain_Specific/binomialOptions_nvrtc/binomialOptions_kernel.cu	#include <cuda.h>
130	src/samples/Samples/5_Domain_Specific/dxtc/dxtc.cu	#include "cuda_runtime.h"
131	src/samples/Samples/5_Domain_Specific/fluidsD3D9/fluidsD3D9_kernels.cu	#include <builtin_types.h>

The table lists the excluded samples along with header files which are missing.Other cases are marked with an asterisk (*) mark on them.

These samples either include header files not supported by HIP Architecture,contain references to assembly languages or have custom data types which are undefined and cannot be found.

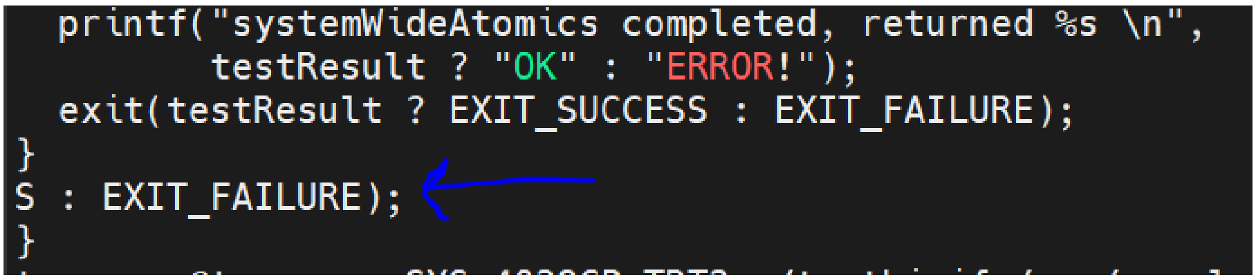
Samples are ignored for the following reasons:

•Inline PTX assembly
•CUDA intrinsics or Graphical Operations or calls to DirectX or Vulcan APIs.
•Hard-coded dependencies on warp size, shared memory size
•Code geared toward limited size of register file on NVIDIA hardware
•Functions implicitly inlined
•Unified Memory

3.5 Challenges

- HIPIFY-Perl provides a file which has regex-substituted HIP functions which can be manually rectified by the user while on the other hand,HIPIFY-Clang would return an empty file if it cannot detect a header file or the sample is unable to run on HIP Architecture.

- Notwithstanding what tool was used for Hipification of the samples,the output file (if generated) would sometimes give a chunk of untranslated code at the end of the hipified cuda sample which gives extraneous brackets error and incompatibility issues due to presence of functions form both CUDA and HIP codebases.



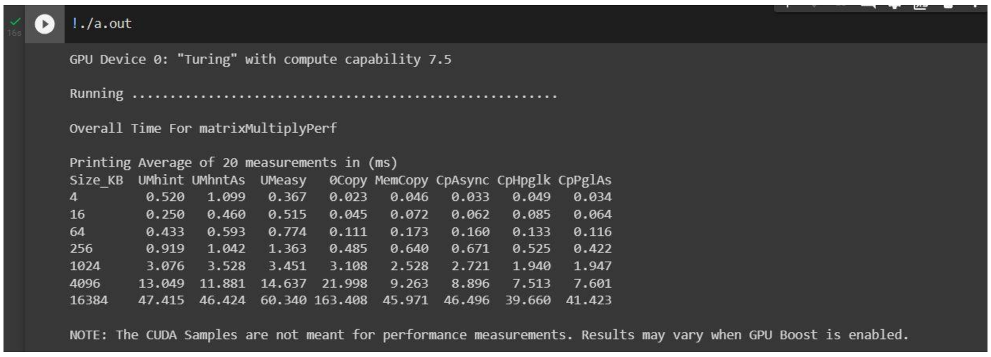
```
  printf("systemWideAtomics completed, returned %s \n",
         testResult ? "OK" : "ERROR!");
  exit(testResult ? EXIT_SUCCESS : EXIT_FAILURE);
}
S : EXIT_FAILURE);
}
```

- Absence of HIPCHECK.h header file which is requires to account for HIP-equivalent function for checkCudaErrors.This was resolved by adding the header file to the ‘Common’ Folder in the repository.

  HIPCHECK.h:

```
#define KNRM "\x1B[0m"
#define KRED "\x1B[31m"
#define KGRN "\x1B[32m"
#define KYEL "\x1B[33m"
#define KBLU "\x1B[34m"
#define KMAG "\x1B[35m"
#define KCYN "\x1B[36m"
#define KWHT "\x1B[37m"


#define failed(...)                                              \
  printf("%serror:                                                  ",
KRED);                                            \

  printf(__VA_ARGS__);                                             \
  printf("\n");                                   \
  printf("error:            TEST            FAILED\n%s",
KNRM);                              \
  exit(EXIT_FAILURE);


#define
HIPCHECK(error)                                   \

  if (error != hipSuccess) {                            \
    printf("%serror:   '%s'(%d)   at   %s:%d%s\n",   KRED,
hipGetErrorString(error), error, __FILE__,  \

        __LINE__, KNRM);                              \
    failed("API                 returned                error
code.");                            \
  }
```

- Entry Point of Samples:The project has shown that not only samples have to be hipified and compiled,the accompanying header files and C++ Programs which contains reference to main functions and additional files which needs to be accessed using the -I flag during compilation (during hipification if using HIPIFY-CLANG).
- Asynchronous Function Metric:In the UnitedMemoryPerf sample, the sample returns time (in ms) to perform matrix multiplication while calling on functions in synchronous and asynchronous flow.Even though the sample compiles successfully,it gets stuck after the asynchronous marker has been set to 1 which does not allow for further continuation of the executeable,By placing print statement markers and isolating certain parts,it is inferred that while synchronous function calling is completed,the sample cannot continue due to said marker set to 1.



```
!./a.out

GPU Device 0: "Turing" with compute capability 7.5

Running ......................................................

Overall Time For matrixMultiplyPerf

Printing Average of 20 measurements in (ms)
Size_KB  UMhint UMhntAs  UMeasy   0Copy MemCopy CpAsync CpHpglk CpPglAs
4         0.520   1.099   0.367   0.023   0.046   0.033   0.049   0.034
16        0.250   0.460   0.515   0.045   0.072   0.062   0.085   0.064
64        0.433   0.593   0.774   0.111   0.173   0.160   0.133   0.116
256       0.919   1.042   1.363   0.485   0.640   0.671   0.525   0.422
1024      3.076   3.528   3.451   3.108   2.528   2.721   1.940   1.947
4096     13.049  11.881  14.637  21.998   9.263   8.896   7.513   7.601
16384    47.415  46.424  60.340 163.408  45.971  46.496  39.660  41.423

NOTE: The CUDA Samples are not meant for performance measurements. Results may vary when GPU Boost is enabled.
```

- SystemWideAtomics Bug:This was a sample that was hipified and compiled successfully.Upon running that sample,the flow got stuck and upon deleting the process, it is found that all the files on the system is rendered as read-only, highlighted by green text in ls command.It also causes involuntary crashing and uninstallation of ROCm,highlighted by the code 103 error which shows ‘no HIP Device’ error.Other issues noted was on accessing rocm-info, the fans of the device had stopped working.The sample was excluded and removed from the execution queue in all processes.

- HipMalloc call to __host__ function from __global__ function is invalid and is needed to be substituted for malloc instead in sample-src/samples/Samples/3_CUDA_Features/cdpBezierTessellation/cdpBezierTessellation.cu

hipccmakes two compilation passes through source code. One to compile host code, and one to compile device code.

__global__ functions:

- **-**These are entry points to device code, called from the host
- **-**Code in these regions will execute on SIMD units
- __device__ functions:
- **-**Can be called from __global__and other __device__ functions.
- **-**Cannot be called from host code.
- **-**Not compiled into host code –essentially ignored during host compilation pass
- __host__ __device__ functions:
- **-**Can be called from __global__, __device__,and hostfunctions.

Barrier and atomic functions are unsupported under HIP Architecture as demonstrated by the following samples after providing the flag -I /usr/local/cuda-12.0/targets/x86_64-linux/include:

1. src/samples/Samples/3_CUDA_Features/binaryPartitionCG/binaryPartitionCG.cu
2. src/samples/Samples/2_Concepts_and_Techniques/reductionMultiBlockCG/reductionMultiBlockCG.cu

- ROCwmma does not act as proper substitute for mma.h header files demonstrated by the following samples after providing the flag -I /opt/rocm-5.5.0-11253/include/rocwmma/ :
  1. src/samples/Samples/3_CUDA_Features/immaTensorCoreGemm/immaTensorCoreGemm.cu
  2. src/samples/Samples/3_CUDA_Features/cudaTensorCoreGemm/cudaTensorCoreGemm.cu

3.6 Need for Patches/Troubleshooting

Git patches are text files that contain the differences between two sets of code or two versions of a file in a Git repository. Patches can be used to apply changes to a codebase in a controlled manner, or to distribute changes made by one person to another.

Operations with Git patches include:

- Creating patches: A patch file can be created by using the 'git diff' or 'git format-patch' commands.
- Applying patches: Patches can be applied using the 'git apply' command.
- Sharing patches: Patches can be shared via email, bug tracking systems, or code review tools.
- Reverting patches: Patches can be reverted by using the 'git apply --reverse' command.
- Managing patches: Git patches can be managed by creating branches, stashing changes, or resetting to a specific commit.

Overall, Git patches are a powerful tool for managing changes in a codebase, especially when working with a team or in a distributed development environment.

The patches edit the hipified samples and performs the following operations:

Replaces references to ‘helper_cuda.h’ and ‘helper_fucntions.h’ to a hipified version of the same files and replacing the original header file references:

src/samples/Samples/5_Domain_Specific/binomialOptions/binomialOptions_kernel.cu
src/samples/Samples/5_Domain_Specific/p2pBandwidthLatencyTest/p2pBandwidthLatencyTest.cu
src/samples/Samples/1_Utilities/topologyQuery/topologyQuery.cu
src/samples/Samples/1_Utilities/bandwidthTest/bandwidthTest.cu
src/samples/Samples/3_CUDA_Features/StreamPriorities/StreamPriorities.cu
src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocation/streamOrderedAllocation.cu
src/samples/Samples/0_Introduction/simpleP2P/simpleP2P.cu
src/samples/Samples/0_Introduction/vectorAdd/vectorAdd.cu
src/samples/Samples/0_Introduction/simpleOccupancy/simpleOccupancy.cu
src/samples/Samples/0_Introduction/simpleCooperativeGroups/simpleCooperativeGroups.cu
src/samples/Samples/0_Introduction/matrixMul/matrixMul.cu
src/samples/Samples/0_Introduction/fp16ScalarProduct/fp16ScalarProduct.cu
src/samples/Samples/0_Introduction/asyncAPI/asyncAPI.cu
src/samples/Samples/0_Introduction/clock/clock.cu
src/samples/Samples/0_Introduction/simpleIPC/simpleIPC.cu
src/samples/Samples/0_Introduction/simplePrintf/simplePrintf.cu
src/samples/Samples/5_Domain_Specific/BlackScholes/BlackScholes.cu
src/samples/Samples/2_Concepts_and_Techniques/histogram/histogram.cu
src/samples/Samples/2_Concepts_and_Techniques/convolutionSeparable/ConvolutionSeperable.cu
src/samples/Samples/0_Introduction/simpleStreams/simpleStreams.cu
src/samples/Samples/0_Introduction/cppIntegration/cppIntegration.cu
src/samples/Samples/0_Introduction/mergeSort/mergeSort.cu
src/samples/Samples/0_Introduction/simpleZeroCopy/simpleZeroCopy.cu
src/samples/Samples/0_Introduction/simpleStreams/simpleStreams.cu
src/samples/Samples/2_Concepts_and_Techniques/scan/scan.cu
src/samples/Samples/0_Introduction/simpleHyperQ/simpleHyperQ.cu
src/samples/Samples/3_CUDA_Features/newdelete/newdelete.cu
src/samples/Samples/2_Concepts_and_Techniques/sortingNetworks/[samples].cu
src/samples/Samples/5_Domain_Specific/dwtHaar1D//[samples].cu
src/samples/Samples/5_Domain_Specific/fastWalshTransform//[samples].cu
testHIPIFYsrc/samples/Samples/2_Concepts_and_Techniques/scalarProd//[samples].cu
src/samples/Samples/2_Concepts_and_Techniques/eigenvalues//[samples].cu
src/samples/Samples/0_Introduction/simpleCallback/simpleCallback.cu
src/samples/Samples/0_Introduction/simpleTexture/[SAMPLES].cu
src/samples/Samples/0_Introduction/simpleAtomicIntrinsics/simpleAtomicIntrinsics.cu

src/samples/Samples/0_Introduction/simpleTemplates/simpleTemplates.cu
src/samples/Samples/0_Introduction/template/template.cu
vsrc/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu
src/samples/Samples/0_Introduction/simpleCubemapTexture/simpleCubemapTexture.cu
src/samples/Samples/0_Introduction/simpleAssert/simpleAssert.cu
src/samples/Samples/0_Introduction/simpleMultiGPU/[samples].cu
src/samples/Samples/0_Introduction/systemWideAtomics/systemWideAtomics.cu
src/samples/Samples/0_Introduction/simpleSurfaceWrite/simpleSurfaceWrite.cu
src/samples/Samples/6_Performance/transpose/transpose.cu
src/samples/Samples/5_Domain_Specific/quasirandomGenerator_nvrtc/quasirandomGenerator_kernel.cu

The patches also replaces checkCudaErrror functions and includes the HIPCHECK.h header file:

src/samples/Samples/2_Concepts_and_Techniques/threadMigration/threadMigration_kernel.out
src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu
src/samples/Samples/3_CUDA_Features/jacobiCudaGraphs/jacobi.cu
src/samples/Samples/0_Introduction/concurrentKernels/[SAMPLES].cu
src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/nv12_to_bgr_planar.cu
src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/nv12_resize.cu
src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/utils.cu
src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/bgr_resize.cu
src/samples/Samples/6_Performance/transpose/transpose.cu
src/samples/Samples/0_Introduction/simpleCallback/simpleCallback.cu

The patches comment out retirementCount invocations:

src/samples/Samples/2_Concepts_and_Techniques/threadFenceReduction/*[SAMPLES].cu*

The patch manually rewrote and replaced all CUDA Functions for HIP-based functions:

src/samples/Samples/5_Domain_Specific/binomialOptions/binomialOptions_kernel.cu

src/samples/Samples/0_Introduction/mergeSort/mergeSort.cu

src/samples/Samples/0_Introduction/simpleStreams/simpleStreams.cu

The patch also removes refereces to profiler based functions:

src/samples/Samples/0_Introduction/matrixMul/matrixMul.cu

src/samples/Samples/0_Introduction/asyncAPI/asyncAPI.cu

The patch removes CUDART_CB* pointer references :

src/samples/Samples/0_Introduction/simpleCallback/simpleCallback.cu

The patch assigned arbitrary unsigned int flag value:

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu

The patch changed cudaStream_t to hipStream_t and its calling functions:

src/samples/Samples/0_Introduction/simpleMultiGPU/*[SAMPLES].cu*

The patch changed atomicInc_system to atomicInc and atomicDec_system to atomicDec in hipified sample:

src/samples/Samples/0_Introduction/systemWideAtomics/systemWideAtomics.cu

The patch removed one argument from surf2Dwrite function to match definition:

src/samples/Samples/0_Introduction/simpleSurfaceWrite/simpleSurfaceWrite.cu

The patch replaced .cuh files wherever called:

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu

The patch typecasted float pointers to void** pointers

src/samples/Samples/6_Performance/UnifiedMemoryPerf/UnifiedMemoryPerf.cu

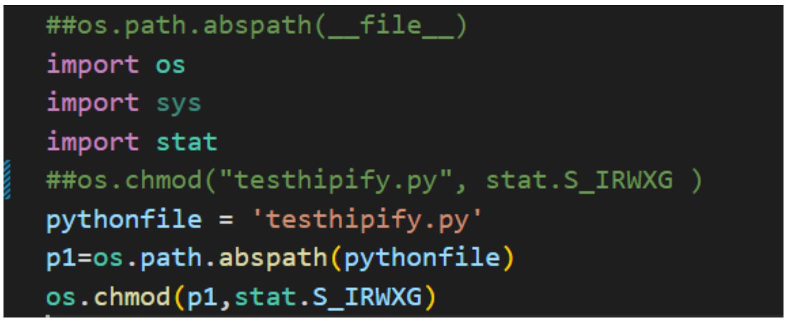
The patch had to remove variables DeviceOverlap and asyncEngineCount as they havent been defined in HipDeviceProp but are present in CudaDeviceProp;dummy variables can be provided but has to be taken up with the developmental team

src/samples/Samples/0_Introduction/simpleMultiCopy/*[SAMPLES].cu*

Note:[SAMPLES].cu refers to the multiple sample/s present inside a sample directory.

3.7 Other Features

- An alternate script which substitutes HIPIFY-Perl for HIPIFY-CLANG is included which carries out the same procedures as the main testHIPIFY.py script.
- A Script to grant admin privileges to the main python script testHIPIFY.py in case of permission based limitations during usage.



```
##os.path.abspath(__file__)
import os
import sys
import stat
##os.chmod("testhipify.py", stat.S_IRWXG )
pythonfile = 'testhipify.py'
p1=os.path.abspath(pythonfile)
os.chmod(p1,stat.S_IRWXG)
```

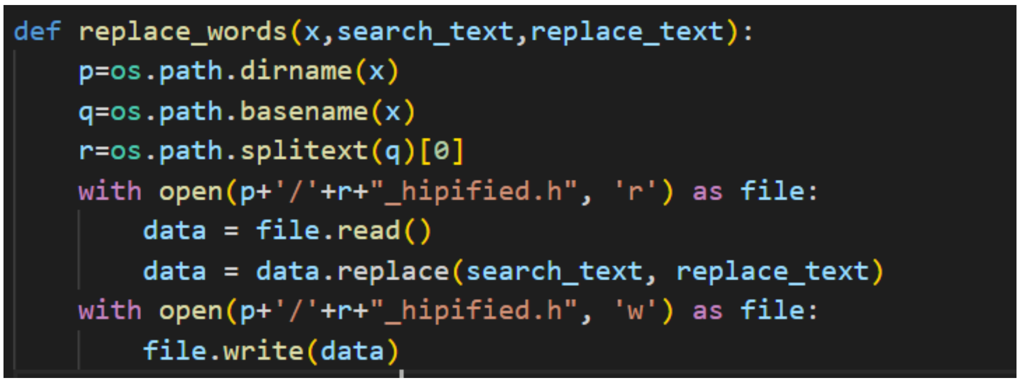
This code modifies the permissions of a python file named "testhipify.py".

1. The "os" module is imported to make use of its functionalities such as os.path and os.chmod.
2. The "sys" module and "stat" module are also imported.
3. The function os.path.abspath(file) gets the absolute path of the current file.
4. The line os.chmod("testhipify.py", stat.S_IRWXG) tries to modify the permissions of the file located at the specified path, but it has syntax errors and will fail because the path is not properly formatted.

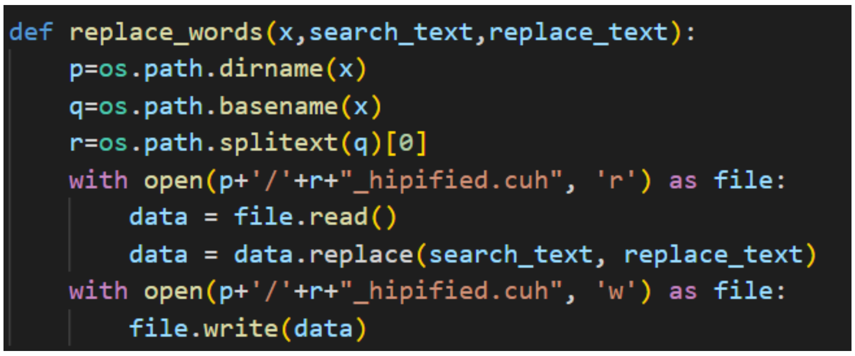
5. The variable "pythonfile" is defined with a string value 'testhipify.py'.
6. The variable "p1" is defined with the absolute path of the python file by calling os.path.abspath() function with "pythonfile" as argument.
7. The function os.chmod() is called with two arguments "p1" (the absolute path of the python file) and "stat.S_IRWXG" (representing the permissions to be granted) to modify the permissions of the "testhipify.py" file.

Note: The permissions specified with stat.S_IRWXG grants read, write and execute permissions to the group.

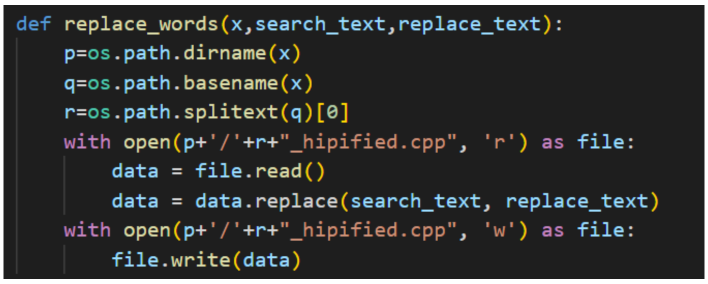
- 3 patch generator scripts to make HIPIFY and make chages to header files,C++ Programs and .cuh files in case the main script glosses over them during the process.These files call onto the same functions as defined above with the only novelty being the replace_word functions which seek for files ending with *_hipified.cpp,*_hipified.h,*_hipified.cuh suffixes.



```python
def replace_words(x,search_text,replace_text):
    p=os.path.dirname(x)
    q=os.path.basename(x)
    r=os.path.splitext(q)[0]
    with open(p+'/'+r+"_hipified.h", 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(p+'/'+r+"_hipified.h", 'w') as file:
        file.write(data)
```



```python
def replace_words(x,search_text,replace_text):
    p=os.path.dirname(x)
    q=os.path.basename(x)
    r=os.path.splitext(q)[0]
    with open(p+'/'+r+"_hipified.cuh", 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(p+'/'+r+"_hipified.cuh", 'w') as file:
        file.write(data)
```



```python
def replace_words(x,search_text,replace_text):
    p=os.path.dirname(x)
    q=os.path.basename(x)
    r=os.path.splitext(q)[0]
    with open(p+'/'+r+"_hipified.cpp", 'r') as file:
        data = file.read()
        data = data.replace(search_text, replace_text)
    with open(p+'/'+r+"_hipified.cpp", 'w') as file:
        file.write(data)
```

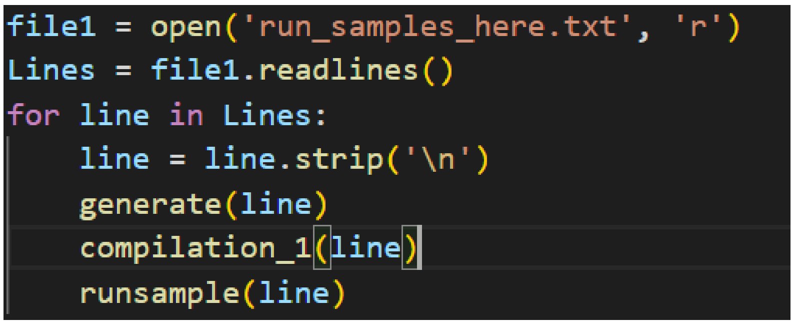
The code replace_words(x,search_text,replace_text) is a function that replaces all instances of search_text with replace_text in a file named x.

It performs the following operations:

1. p is set to the directory path of x.
2. q is set to the base name of x.

3. r is set to the root name of q by splitting the file name and extension using os.path.splitext.
4. The function opens the file p+'/'+r+"_hipified.h" in read mode, reads the contents of the file into data, and replaces all instances of search_text with replace_text.
5. The function opens the same file in write mode, and writes the modified data back to the file.

- Testing_experimental.py is an alternate solution to running samples where you can feed in paths in the text file run_samples_here.txt to carry out hipification, compilation and usage of samples whose paths have been provided by user.have been provided by user.have been provided by user.It calls onto the same functions as testhipify.py, with the only exception being the main function which calls the pre-defined functions.



```
file1 = open('run_samples_here.txt', 'r')
Lines = file1.readlines()
for line in Lines:
    line = line.strip('\n')
    generate(line)
    compilation_1(line)
    runsample(line)
```

This code reads the contents of a text file named "run_samples_here.txt" line by line and performs some actions on each line.

1. The file "run_samples_here.txt" is opened using the built-in function "open()" with mode 'r' which stands for read-only mode. The file handle is stored in the variable "file1".
2. The contents of the file are read using the method "readlines()" and stored in the variable "Lines".
3. A for loop is used to iterate over each line in the "Lines" list.
4. Within the for loop, each line is stripped of the newline character '\n' using the "strip()" method.
5. The function "generate()" is called with the line as an argument.
6. The function "compilation_1()" is called with the line as an argument.
7. The function "runsample()" is called with the line as an argument.

CHAPTER 4:VALIDATION/TESTING

The testhipify repository is a tool for converting C++ code from CUDA syntax to HIP syntax. As such, the main purpose of the test report for this repository is to ensure that the tool works correctly and can reliably convert C++ code from CUDA to HIP syntax.
The test report for this repository should include the following information:

- Test environment: The report should specify the environment in which the tests were conducted, including the operating system, the hardware used, and any other relevant details.
- Test coverage: The report should outline the areas of the code that were tested, including any functions or modules that were specifically targeted. The report should also indicate the level of coverage achieved, such as unit tests, integration tests, or system tests.
- Test results: The report should include the results of the tests, including any errors or failures that were encountered. The report should also include any warnings or issues that were identified during testing.
- Test summary: The report should provide a summary of the overall results of the tests, including any areas of the code that need improvement or further testing.
- Recommendations: Based on the test results, the report should provide recommendations for improvements to the code or testing process. This could include suggestions for additional test cases, code refactoring, or modifications to the testing environment.

Overall, the test report for the testhipify repository should demonstrate that the tool is reliable and effective at converting C++ code from CUDA syntax to HIP syntax. The report should also provide insights into any areas of the code that need improvement or further testing.
The tests have been carried out in baremetal sessions with same specifications as stated above and counts metrics for executeable as all samples passed can generate HIP files and be compiled.
The following is a summary of the test case results obtained for the reported test effort. Refer to subordinate sections of this document for detailed results and explanations of any reported variances.

Test Case Summary Results

- 194 Samples
- 112 Ignored Samples
- 82 Working Samples
- 69 Executeables
- 7 Accused Samples(Samples which are debarred but can made to work with modifications)

Legend

x-Failure

✓-Success

~-Ignored

#	Sample Name	V100	MI100	MI200	MI50	Comments
1	src/samples/Samples/0_Introduction/asyncAPI/asyncAPI.cu	X	X	X	X	API Error
2	src/samples/Samples/0_Introduction/clock/clock.cu	✓	✓	✓	✓	
3	src/samples/Samples/0_Introduction/clock_nvrtc/clock_kernel.cu	X	X	X	X	Kernel Program requires "cuda.h" header file
4	src/samples/Samples/0_Introduction/concurrentKernels/concurrentKernels.cu	✓	✓	✓	✓	
5	src/samples/Samples/0_Introduction/cppIntegration/cppIntegration.cu	✓	✓	✓	✓	
6	src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu	✓	✓	✓	✓	
7	src/samples/Samples/0_Introduction/fp16ScalarProduct/fp16ScalarProduct.cu	✓	✓	✓	✓	
8	src/samples/Samples/0_Introduction/matrixMul/matrixMul.cu	✓	✓	✓	✓	
9	src/samples/Samples/0_Introduction/mergeSort/bitonic.cu	✓	✓	✓	✓	
10	src/samples/Samples/0_Introduction/mergeSort/mergeSort.cu	✓	✓	✓	✓	
11	src/samples/Samples/0_Introduction/simpleCallback/simpleCallback.cu	✓	✓	✓	✓	
12	src/samples/Samples/0_Introduction/simpleCooperativeGroups/simpleCooperativeGroups.cu	✓	✓	✓	✓	
13	src/samples/Samples/0_Introduction/simpleCubemapTexture/simpleCubemapTexture.cu	X	X	X	X	API Error
14	src/samples/Samples/0_Introduction/simpleHyperQ/simpleHyperQ.cu	✓	✓	✓	✓	

15	src/samples/Samples/0_Introduction/simpleIPC/simpleIPC.cu	X	X	X	X	
16	src/samples/Samples/0_Introduction/simpleLayeredTexture/simpleLayeredTexture.cu	√	√	√	√	
17	src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu	√	√	√	√	Requires OpenMPI Installation
18	src/samples/Samples/0_Introduction/simpleMultiCopy/simpleMultiCopy.cu	√	√	√	√	
19	src/samples/Samples/0_Introduction/simpleMultiGPU/simpleMultiGPU.cu	√	√	√	√	
20	src/samples/Samples/0_Introduction/simpleOccupancy/simpleOccupancy.cu	√	√	√	√	
21	src/samples/Samples/0_Introduction/simpleP2P/simpleP2P.cu	X	√	X	X	If machine is PCI based then AMD_IOMMU =ON , iommu=pt were needed parameters in order for √ful execution of P2P-based samples.
22	src/samples/Samples/0_Introduction/simplePrintf/simplePrintf.cu	√	√	√	√	
23	src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleDeviceLibrary.cu	√	√	√	√	
24	src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleSeparateCompilation.cu	√	√	√	√	
25	src/samples/Samples/0_Introduction/simpleStreams/simpleStreams.cu	√	√	√	√	
26	src/samples/Samples/0_Introduction/simpleTemplates/simpleTemplates.cu	√	√	√	√	
27	src/samples/Samples/0_Introduction/simpleZeroCopy/simpleZeroCopy.cu	√	√	√	√	

28	src/samples/Samples/0_Introduction/template/template.cu	✓	✓	✓	✓	
29	src/samples/Samples/0_Introduction/UnifiedMemoryStreams/UnifiedMemoryStreams.cu	X	X	X	X	Requires hipblas header file;error while loading shared libraries: libomp.so: cannot open shared object file
30	src/samples/Samples/0_Introduction/vectorAdd/vectorAdd.cu	✓	✓	✓	✓	
31	src/samples/Samples/1_Utilities/bandwidthTest/bandwidthTest.cu	✓	✓	✓	✓	
32	src/samples/Samples/1_Utilities/topologyQuery/topologyQuery.cu	X	X	X	X	API Error
33	src/samples/Samples/2_Concepts_and_Techniques/convolutionSeparable/convolutionSeparable.cu	✓	✓	✓	✓	
34	src/samples/Samples/2_Concepts_and_Techniques/eigenvalues/bisect_large.cu	X	X	X	X	sdkFilePath Error
35	src/samples/Samples/2_Concepts_and_Techniques/eigenvalues/bisect_small.cu	X	X	X	X	sdkFilePath Error
36	src/samples/Samples/2_Concepts_and_Techniques/eigenvalues/bisect_util.cu	X	X	X	X	sdkFilePath Error
37	src/samples/Samples/2_Concepts_and_Techniques/eigenvalues/main.cu	X	X	X	X	sdkFilePath Error
38	src/samples/Samples/2_Concepts_and_Techniques/histogram/histogram256.cu	✓	✓	✓	✓	

39	src/samples/Samples/2_Concepts_and_Techniques/histogram/histogram64.cu	✓	✓	✓	✓	
40	src/samples/Samples/2_Concepts_and_Techniques/interval/interval.cu	✓	✓	✓	✓	
41	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiInlineQ/src/piestimator.cu	X	X	X	X	Requires header file : #include <hiprand.h>
42	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiP/src/piestimator.cu	X	X	X	X	Requires header file : #include <hiprand.h>
43	src/samples/Samples/2_Concepts_and_Techniques/MC_SingleAsianOptionP/src/priceingengine.cu	X	X	X	X	version `GLIBC_2.34' not found
44	src/samples/Samples/2_Concepts_and_Techniques/MC_SingleAsianOptionP/src/pricingengine.cu	X	X	X	X	version `GLIBC_2.34' not found
45	src/samples/Samples/2_Concepts_and_Techniques/MC_SingleAsianOptionP/src/src.cu	X	X	X	X	version `GLIBC_2.34' not found
46	src/samples/Samples/2_Concepts_and_Techniques/radixSortThrust/radixSortThrust.cu	X	X	X	X	Requires header file : #include <thrust/host_vector.h>
47	src/samples/Samples/2_Concepts_and_Techniques/reduction/reduction_kernel.cu	X	X	X	X	Requires header file : #include <cuda/std/type_traits>
48	src/samples/Samples/2_Concepts_and_Techniques/scalarProd/scalarProd.cu	✓	✓	✓	✓	
49	src/samples/Samples/2_Concepts_and_Techniques/scan/scan.cu	✓	✓	✓	✓	
50	src/samples/Samples/2_Concepts_and_Techniques/segmentationTreeThrust/segmentationTree.cu	X	X	X	X	Requires header file : #include <thrust/iterator/discard_iterator.h>
51	src/samples/Samples/2_Concepts_and_Techniques/sortingNetworks/bitonicSort.cu	✓	✓	✓	✓	

52	src/samples/Samples/2_Concepts_and_Techniques/sortingNetworks/oddEvenMergeSort.cu	✓	✓	✓	✓	
53	src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocation/streamOrderedAllocation.cu	X	X	X	X	Device does not support Memory Pool
54	src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu	X	X	X	X	No Two or more GPUs with same architecture capable of cuda Memory Pools found
55	src/samples/Samples/2_Concepts_and_Techniques/threadFenceReduction/threadFenceReduction.cu	✓	✓	✓	✓	
56	src/samples/Samples/3_CUDA_Features/cudaCompressibleMemory/saxpy.cu	X	X	X	X	API Error
57	src/samples/Samples/3_CUDA_Features/jacobiCudaGraphs/jacobi.cu	✓	✓	✓	✓	
58	src/samples/Samples/3_CUDA_Features/newdelete/newdelete.cu	✓	✓	✓	✓	
59	src/samples/Samples/3_CUDA_Features/simpleCudaGraphs/simpleCudaGraphs.cu	✓	✓	✓	✓	
60	src/samples/Samples/3_CUDA_Features/StreamPriorities/StreamPriorities.cu	✓	✓	✓	✓	
61	src/samples/Samples/5_Domain_Specific/binomialOptions/binomialOptions_kernel.cu	✓	✓	✓	✓	
62	src/samples/Samples/5_Domain_Specific/BlackScholes/BlackScholes.cu	✓	✓	✓	✓	
63	src/samples/Samples/5_Domain_Specific/convolutionFFT2D/convolutionFFT2D.cu	X	X	X	X	Requires header file : #include <hipfft/hipfft.h>
64	src/samples/Samples/5_Domain_Specific/dwtHaar1D/dwtHaar1D.cu	X	X	X	X	sdkFilePath Error

65	src/samples/Samples/5_Domain_Specific/fastWalshTransform/fastWalshTransform.cu	✓	✓	✓	✓	
66	src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu	X	X	X	X	sdkFilePath Error
67	src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/bgr_resize.cu	X	X	X	X	sdkFilePath Error
68	src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/nv12_resize.cu	X	X	X	X	sdkFilePath Error
69	src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/nv12_to_bgr_planar.cu	X	X	X	X	sdkFilePath Error
70	src/samples/Samples/5_Domain_Specific/NV12toBGRandResize/utils.cu	X	X	X	X	sdkFilePath Error
71	src/samples/Samples/5_Domain_Specific/p2pBandwidthLatencyTest/p2pBandwidthLatencyTest.cu	X	✓	X	X	If machine is PCI based then AMD_IOMMU =ON , iommu=pt were needed parameters in order for ✓ful execution of P2P-based samples.
72	src/samples/Samples/5_Domain_Specific/quasirandomGenerator/quasirandomGenerator_kernel.cu	✓	✓	✓	✓	
73	src/samples/Samples/5_Domain_Specific/SobolQRNG/sobol_gpu.cu	✓	✓	✓	✓	
74	src/samples/Samples/6_Performance/alignedTypes/alignedTypes.cu	✓	✓	✓	✓	
75	src/samples/Samples/6_Performance/transpose/transpose.cu	✓	✓	✓	✓	
76	src/samples/Samples/6_Performance/UnifiedMemoryPerf/commonKernels.cu	X	X	X	X	Sample stuck at asynchronous marker
77	src/samples/Samples/6_Performance/UnifiedMemoryPerf/matrixMultiplyPerf.cu	X	X	X	X	Sample stuck at asynchronous marker

78	src/samples/Samples/0_Introduction/c++11_cuda/c++11_cuda.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
79	src/samples/Samples/0_Introduction/cppOverload/cppOverload.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
80	src/samples/Samples/0_Introduction/matrixMulDrv/matrixMul_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
81	src/samples/Samples/0_Introduction/matrixMul_nvrtc/matrixMul_kernel.cu	~	~	~	~	Requires header file : #include <cuda/barrier>
82	src/samples/Samples/0_Introduction/simpleAWBarrier/simpleAWBarrier.cu	~	~	~	~	ROCm Deletion Bug
83	src/samples/Samples/0_Introduction/simpleAssert/simpleAssert.cu	X	X	X	X	HSA_STATUS_ERROR_EXCEPTION: An HSAIL operation resulted in a hardware exception.
84	src/samples/Samples/0_Introduction/simpleAssert_nvrtc/simpleAssert_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
85	src/samples/Samples/0_Introduction/simpleAtomicIntrinsics/simpleAtomicIntrinsics.cu	✓	✓	✓	✓	
86	src/samples/Samples/0_Introduction/simpleAttributes/simpleAttributes.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
87	src/samples/Samples/0_Introduction/simpleCUDA2GL/simpleCUDA2GL.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
88	src/samples/Samples/0_Introduction/simpleDrvRuntime/vectorAdd_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
89	src/samples/Samples/0_Introduction/simplePitchLinearTexture/simplePitchLinearTexture.cu	✓	✓	✓	✓	

90	src/samples/Samples/0_Introduction/simpleSurfaceWrite/simpleSurfaceWrite.cu	X	X	X	X	Accused;sdkFilePath
91	src/samples/Samples/0_Introduction/simpleTemplates_nvrtc/simpleTemplates_kernel.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
92	src/samples/Samples/0_Introduction/simpleTexture/simpleTexture.cu	X	X	X	X	Accused;sdkFilePath
93	src/samples/Samples/0_Introduction/simpleTexture3D/simpleTexture3D_kernel.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
94	src/samples/Samples/0_Introduction/simpleTextureDrv/simpleTexture_kernel.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
95	src/samples/Samples/0_Introduction/simpleVoteIntrinsics/simpleVoteIntrinsics.cu	~	~	~	~	use of undeclared identifier '__any_sync'
96	src/samples/Samples/0_Introduction/systemWideAtomics/systemWideAtomics.cu	~	~	~	~	ROCm Deletion Bug
97	src/samples/Samples/0_Introduction/vectorAddDrv/vectorAdd_kernel.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
98	src/samples/Samples/0_Introduction/vectorAddMMAP/vectorAdd_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
99	src/samples/Samples/0_Introduction/vectorAdd_nvrtc/vectorAdd_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
100	src/samples/Samples/0_Introduction/vectorAdd_nvrtc/vectorAdd_nvrtc.cu	~	~	~	~	Requires header file : #include <cuda.h>
101	src/samples/Samples/2_Concepts_and_Techniques/EGLStream_CUDA_CrossGPU/kernel.cu	~	~	~	~	Requires header file : #include "cudaEGL.h"
102	src/samples/Samples/2_Concepts_and_Techniques/EGLSync_CUDAEvent_Interop/EGLSync_CUDAEvent_Interop.cu	~	~	~	~	Requires header file : #include "cudaEGL.h"

103	src/samples/Samples/2_Concepts_and_Techniques/FunctionPointers/FunctionPointers_kernels.cu	~	~	~	~	<cuda_gl_interop.h>
104	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiInlineP/src/piestimator.cu	~	~	~	~	Requires header file : #include <hiprand_kernel.h>
105	src/samples/Samples/2_Concepts_and_Techniques/MC_EstimatePiQ/src/piestimator.cu	~	~	~	~	Requires header file : #include <hiprand.h>
106	src/samples/Samples/2_Concepts_and_Techniques/boxFilter/boxFilter_kernel.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
107	src/samples/Samples/2_Concepts_and_Techniques/convolutionTexture/convolutionTexture.cu	~	~	~	~	Requires header file : #include <cuda_runtime.h>
108	src/samples/Samples/2_Concepts_and_Techniques/dct8x8/dct8x8.cu	~	~	~	~	Requires header file : #include <cuda_runtime.h>
109	src/samples/Samples/2_Concepts_and_Techniques/imageDenoising/imageDenoising.cu	~	~	~	~	<cuda_gl_interop.h>
110	src/samples/Samples/2_Concepts_and_Techniques/inlinePTX/inlinePTX.cu	~	~	~	~	ASM Language
111	src/samples/Samples/2_Concepts_and_Techniques/inlinePTX_nvrtc/inlinePTX_kernel.cu	~	~	~	~	ASM Language
112	src/samples/Samples/2_Concepts_and_Techniques/particles/particleSystem_cuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
113	src/samples/Samples/2_Concepts_and_Techniques/reductionMultiBlockCG/reductionMultiBlockCG.cu	~	~	~	~	Requires header file : #include <cooperative_groups/reduce.h>
114	src/samples/Samples/2_Concepts_and_Techniques/shfl_scan/shfl_scan.cu	~	~	~	~	Unsupported shfl_sync functions.

115	src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationIPC/streamOrderedAllocationIPC.cu	~	~	~	~	unknown type name 'CUresult', unsupported identifier "CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_POSIX_FILE_DESCRIPTOR_SUPPORTED" unsupported identifier "CU_DEVICE_ATTRIBUTE_HANDLE_TYPE_WIN32_HANDLE_SUPPORTED"
116	src/samples/Samples/2_Concepts_and_Techniques/threadMigration/threadMigration_kernel.cu	✓	✓	✓	✓	
117	src/samples/Samples/3_CUDA_Features/bf16TensorCoreGemm/bf16TensorCoreGemm.cu	~	~	~	~	Requires header file : #include <cuda_bf16.h>
118	src/samples/Samples/3_CUDA_Features/binaryPartitionCG/binaryPartitionCG.cu	~	~	~	~	Requires header file : #include <cooperative_groups/reduce.h>
119	src/samples/Samples/3_CUDA_Features/bindlessTexture/bindlessTexture_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
120	src/samples/Samples/3_CUDA_Features/cdpAdvancedQuicksort/cdpAdvancedQuicksort.cu	~	~	~	~	call to __host__ function from __global__ function
121	src/samples/Samples/3_CUDA_Features/cdpAdvancedQuicksort/cdpBitonicSort.cu	~	~	~	~	call to __host__ function from __global__ function
122	src/samples/Samples/3_CUDA_Features/cdpBezierTessellation/BezierLineCDP.cu	~	~	~	~	no member named 'any' and 'ballot' in 'cooperative_groups
123	src/samples/Samples/3_CUDA_Features/cdpQuadtree/cdpQuadtree.cu	~	~	~	~	no member named 'any' and 'ballot' in 'cooperative_groups

124	src/samples/Samples/3_CUDA_Features/cdpSimplePrint/cdpSimplePrint.cu	~	~	~	~	reference to __global__ function 'cdp_kernel' in __global__ function, use of undeclared identifier 'checkCmdLineFlag'
125	src/samples/Samples/3_CUDA_Features/cdpSimpleQuicksort/cdpSimpleQuicksort.cu	~	~	~	~	call to __host__ function from __global__ function
126	src/samples/Samples/3_CUDA_Features/cudaTensorCoreGemm/cudaTensorCoreGemm.cu	~	~	~	~	Requires header file : #include <mma.h>
127	src/samples/Samples/3_CUDA_Features/dmmaTensorCoreGemm/dmmaTensorCoreGemm.cu	~	~	~	~	Requires header file : #include <mma.h>
128	src/samples/Samples/3_CUDA_Features/globalToShmemAsyncCopy/globalToShmemAsyncCopy.cu	~	~	~	~	Requires header file : #include <cuda/pipeline>
129	src/samples/Samples/3_CUDA_Features/graphMemoryFootprint/graphMemoryFootprint.cu	~	~	~	~	unsupported identifier "cudaGraphAddMemFreeNode","cudaMemAllocNodeParams"
130	src/samples/Samples/3_CUDA_Features/graphMemoryNodes/graphMemoryNodes.cu	~	~	~	~	use of undeclared identifier 'cudaGraphAddMemFreeNode, unknown type name 'cudaMemAllocNodeParams'
131	src/samples/Samples/3_CUDA_Features/immaTensorCoreGemm/immaTensorCoreGemm.cu	~	~	~	~	Requires header file : #include <mma.h>
132	src/samples/Samples/3_CUDA_Features/memMapIPCDrv/memMapIpc_kernel.cu	~	~	~	~	<builtin_types.h>
133	src/samples/Samples/3_CUDA_Features/ptxjit/ptxjit_kernel.cu	~	~	~	~	No equivalent function for findCudaDeviceDRV

134	src/samples/Samples/3_CUDA_Features/tf32TensorCoreGemm/tf32TensorCoreGemm.cu	~	~	~	~	Requires header file : #include <mma.h>
135	src/samples/Samples/3_CUDA_Features/warpAggregatedAtomicsCG/warpAggregatedAtomicsCG.cu	~	~	~	~	Requires header file : #include <cooperative_groups/reduce.h>
136	src/samples/Samples/4_CUDA_Libraries/conjugateGradientCudaGraphs/conjugateGradientCudaGraphs.cu	~	~	~	~	Requires header file : #include <hipblas.h>
137	src/samples/Samples/4_CUDA_Libraries/conjugateGradientMultiBlockCG/conjugateGradientMultiBlockCG.cu	~	~	~	~	Requires header file : #include <cooperative_groups/reduce.h>
138	src/samples/Samples/4_CUDA_Libraries/conjugateGradientMultiDeviceCG/conjugateGradientMultiDeviceCG.cu	~	~	~	~	Requires header file : #include <cooperative_groups/reduce.h>
139	src/samples/Samples/4_CUDA_Libraries/cuDLAErrorReporting/main.cu	~	~	~	~	Requires header file : #include "cudla.h"
140	src/samples/Samples/4_CUDA_Libraries/cuDLAHybridMode/main.cu	~	~	~	~	Requires header file : #include "cudla.h"
141	src/samples/Samples/4_CUDA_Libraries/cudaNvSci/imageKernels.cu	~	~	~	~	Requires header file : #include <cuda_runtime.h>
142	src/samples/Samples/4_CUDA_Libraries/cudaNvSciNvMedia/cuda_consumer.cu	~	~	~	~	Requires header file : #include <nvscisync.h>
143	src/samples/Samples/4_CUDA_Libraries/lineOfSight/lineOfSight.cu	~	~	~	~	Requires header file : #include <cuda_runtime.h>
144	src/samples/Samples/4_CUDA_Libraries/oceanFFT/oceanFFT_kernel.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
145	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT/simpleCUFFT.cu	~	~	~	~	Requires header file : #include <hipfft.h>

146	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_2d_MGPU/simpleCUFFT_2d_MGPU.cu	~	~	~	~	Requires header file : #include <hipfftXt.h>
147	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_MGPU/simpleCUFFT_MGPU.cu	~	~	~	~	Requires header file : #include <hipfftXt.h>
148	src/samples/Samples/4_CUDA_Libraries/simpleCUFFT_callback/simpleCUFFT_callback.cu	~	~	~	~	Requires header file : #include <hipfft.h>
149	src/samples/Samples/5_Domain_Specific/FDTD3d/src/FDTD3dGPU.cu	~	~	~	~	Requires header file : #include "FDTD3d.h"
150	src/samples/Samples/5_Domain_Specific/Mandelbrot/Mandelbrot_cuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
151	src/samples/Samples/5_Domain_Specific/MonteCarloMultiGPU/MonteCarlo_kernel.cu	~	~	~	~	Requires header file : #include <hiprand_kernel.h>
152	src/samples/Samples/5_Domain_Specific/SLID3D10Texture/texture_2d.cu	~	~	~	~	Requires header file : #include <windows.h>
153	src/samples/Samples/5_Domain_Specific/SobelFilter/SobelFilter_kernels.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
154	src/samples/Samples/5_Domain_Specific/VFlockingD3D10/VFlocking_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
155	src/samples/Samples/5_Domain_Specific/bicubicTexture/bicubicTexture_cuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
156	src/samples/Samples/5_Domain_Specific/bilateralFilter/bilateral_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
157	src/samples/Samples/5_Domain_Specific/binomialOptions_nvrtc/binomialOptions_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
158	src/samples/Samples/5_Domain_Specific/dxtc/dxtc.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"

159	src/samples/Samples/5_Domain_Specific/fluidsD3D9/fluidsD3D9_kernels.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
160	src/samples/Samples/5_Domain_Specific/fluidsGL/fluidsGL_kernels.cu	~	~	~	~	Requires header file : #include <hipfft.h>
161	src/samples/Samples/5_Domain_Specific/fluidsGLES/fluidsGLES_kernels.cu	~	~	~	~	Requires header file : #include <hipfft.h>
162	src/samples/Samples/5_Domain_Specific/marchingCubes/marchingCubes_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
163	src/samples/Samples/5_Domain_Specific/nbody/bodysystemcuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
164	src/samples/Samples/5_Domain_Specific/nbody_opengles/bodysystemcuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
165	src/samples/Samples/5_Domain_Specific/nbody_screen/bodysystemcuda.cu	~	~	~	~	Requires header file : #include <screen/screen.h>
166	src/samples/Samples/5_Domain_Specific/postProcessGL/postProcessGL.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
167	src/samples/Samples/5_Domain_Specific/quasirandomGenerator_nvrtc/quasirandomGenerator_kernel.cu	~	~	~	~	Requires header file : #include <cuda.h>
168	src/samples/Samples/5_Domain_Specific/recursiveGaussian/recursiveGaussian_cuda.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
169	src/samples/Samples/5_Domain_Specific/simpleD3D10/simpleD3D10_kernel.cu	~	~	~	~	Requires header file : #include <builtin_types.h>
170	src/samples/Samples/5_Domain_Specific/simpleD3D10RenderTarget/simpleD3D10RenderTarget_kernel.cu	~	~	~	~	Requires header file : #include <builtin_types.h>

171	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_2d.cu	~	~	~	~	Requires header file : #include <windows.h>
172	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_3d.cu	~	~	~	~	Requires header file : #include <windows.h>
173	src/samples/Samples/5_Domain_Specific/simpleD3D10Texture/texture_cube.cu	~	~	~	~	Requires header file : #include <windows.h>
174	src/samples/Samples/5_Domain_Specific/simpleD3D11/sinewave_cuda.cu	~	~	~	~	Requires header file : #include <windows.h>
175	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_2d.cu	~	~	~	~	Requires header file : #include <windows.h>
176	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_3d.cu	~	~	~	~	Requires header file : #include <windows.h>
177	src/samples/Samples/5_Domain_Specific/simpleD3D11Texture/texture_cube.cu	~	~	~	~	Requires header file : #include <windows.h>
178	src/samples/Samples/5_Domain_Specific/simpleD3D12/sinewave_cuda.cu	~	~	~	~	Requires header file : #include <windows.h>
179	src/samples/Samples/5_Domain_Specific/simpleD3D9/simpleD3D9_kernel.cu	~	~	~	~	Requires header file : #include <Windows.h>
180	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_2d.cu	~	~	~	~	Requires header file : #include <cuda_d3d9_interop.h>
181	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_cube.cu	~	~	~	~	Requires header file : #include <cuda_d3d9_interop.h>
182	src/samples/Samples/5_Domain_Specific/simpleD3D9Texture/texture_volume.cu	~	~	~	~	Requires header file : #include <cuda_d3d9_interop.h>
183	src/samples/Samples/5_Domain_Specific/simpleGL/simpleGL.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>

184	src/samples/Samples/5_Domain_Specific/simpleGLES/simpleGLES.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
185	src/samples/Samples/5_Domain_Specific/simpleGLES_EGLOutput/simpleGLES_EGLOutput.cu	~	~	~	~	Requires header file : #include <drm.h>
186	src/samples/Samples/5_Domain_Specific/simpleGLES_screen/simpleGLES_screen.cu	~	~	~	~	Requires header file : #include <screen/screen.h>
187	src/samples/Samples/5_Domain_Specific/simpleVulkan/SineWaveSimulation.cu	~	~	~	~	Requires header file : #include <cuda_runtime_api.h>
188	src/samples/Samples/5_Domain_Specific/simpleVulkanMMAP/MonteCarloPi.cu	~	~	~	~	Requires header file : #include <cuda_runtime_api.h>
189	src/samples/Samples/5_Domain_Specific/smokeParticles/ParticleSystem_cuda.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
190	src/samples/Samples/5_Domain_Specific/stereoDisparity/stereoDisparity.cu	~	~	~	~	ASM Language
191	src/samples/Samples/5_Domain_Specific/volumeFiltering/volumeFilter_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
192	src/samples/Samples/5_Domain_Specific/volumeFiltering/volumeRender_kernel.cu	~	~	~	~	Requires header file : #include "cuda_runtime.h"
193	src/samples/Samples/5_Domain_Specific/volumeRender/volumeRender_kernel.cu	~	~	~	~	Requires header file : #include <cuda_gl_interop.h>
194	src/samples/Samples/5_Domain_Specific/vulkanImageCUDA/vulkanImageCUDA.cu	~	~	~	~	Requires header file : #include <GLFW/glfw3.h>

4.1 Nvidia Machine Testing

Since Nvidia samples cannot be run on both HIP and CUDA codebases,the following steps need to be followed to make use of the project on Nvidia GPU-based devices:

Install ROCm for usage of HIP Runtime and toolchain: Provisions for ROCm Stack which includes the HIP based runtime and functions can be found on this webpage: https://docs.amd.com/bundle/ROCm-Installation-Guidev5.4.3/page/How_to_Install_ROCm.html

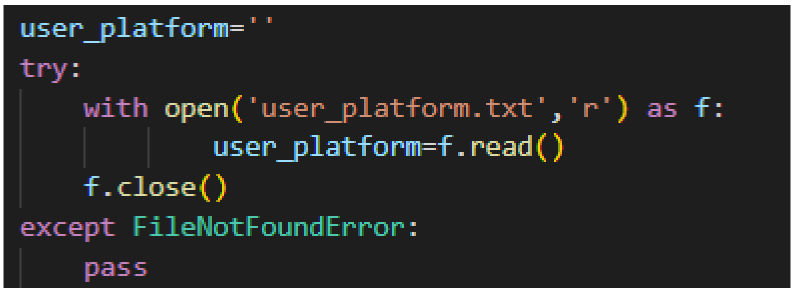
The user can follow the installation instructions for their respective Operating System and install the required packages.

Install Nvidia CUDA Toolkit:Download and install the Nvidia CUDA toolkit from the site: https://developer.nvidia.com/cuda-downloads. The user needs to make sure to install the compatible version of CUDA Toolkit (12.0,preferably) that works with HIP Architecture.

Incase the machine has multiple versions of CUDA installed, ensure that the installation folder of the toolkit is exported as an environmental variable and the path is correctly listen when using the setup provision (-s argument) when running the testHIPIFY project.

Note that not all features of the HIP architecture may be supported on Nvidia GPUs. You should check the compatibility of your code with Nvidia GPUs before running the program. Additionally, you may need to modify your code to take advantage of specific features of the Nvidia hardware.

The following enhancements to the main script were made to accommodate HIP-based samples on an Nvidia Device:

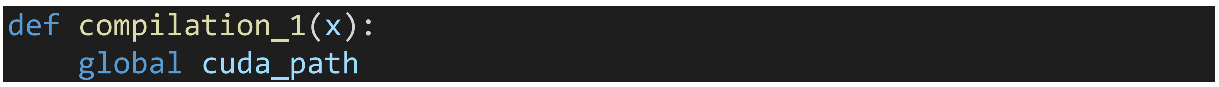


```python
user_platform=''
try:
    with open('user_platform.txt','r') as f:
            user_platform=f.read()
    f.close()
except FileNotFoundError:
    pass
```

Outside the scope of the functions,we initialize a variable called 'user_platform' to an empty string.It attempts to open a file called user_platform.txt in read mode using the with statement. If the file is found, it reads its contents into the user_platform variable.If the FileNotFoundError exception is raised (i.e., the file is not found), the code block does nothing and continues executing.

The purpose of this code block is to check if a user_platform.txt file exists, and if it does, read its contents into the user_platform variable. This allows the program to resume from where it left off the last time it was executed, without having to prompt the user again for their system specifications. If the file does not exist, the except block catches the FileNotFoundError exception and the code continues executing without any error.

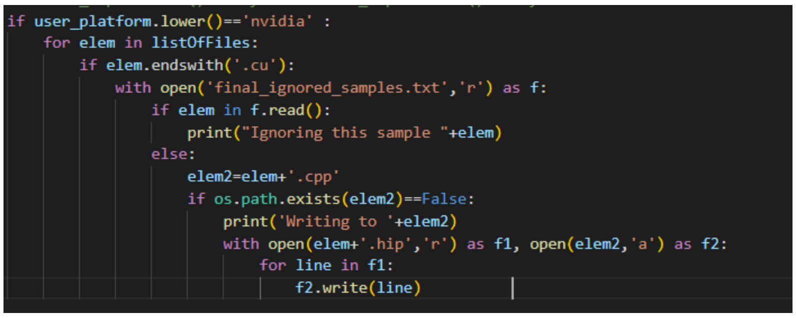
The following modifications were done to the functions compilation_1() and compilation_2() to process the newly generated hipified samples with the difference that unlike conversion to file extension of *.cu.hip, the script would generate files of extensions *.cu.cpp for NVIDIA compatibility purposes.



```python
def compilation_1(x):
    global cuda_path
    global user_platform
    cpp=[]
    print(user_platform)
    x=x.replace('"', '')
    p=os.path.dirname(x)
    p=p.replace("\\","/")
    if
x=='src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu' and
user_platform.lower() == 'amd':
        command='hipcc -I src/samples/Common
src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu.hip
src/samples/Samples/0_Introduction/simpleMPI/simpleMPI_hipified.cpp
-lmpi -o src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.out'
        print(command)
        os.system(command)
    elif
x=='src/samples/Samples/0_Introduction/simpleSeparateCompilation/sim
pleDeviceLibrary.cu' or
x=='/src/samples/Samples/0_Introduction/simpleSeparateCompilation/si
mpleSeparateCompilation.cu' and user_platform.lower() == 'amd':
        command='hipcc -I src/samples/Common -fgpu-rdc
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleD
eviceLibrary.cu.hip
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleS
eparateCompilation.cu.hip -o
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleS
eparateCompilation.out'
        print(command)
        os.system(command)
    elif
x=='src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu' and
user_platform.lower() == 'amd':
        command='hipcc -I src/samples/Common -fopenmp
src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu.hip -o
src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.out'
        print(command)
        os.system(command)
    if
x=='src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu' and
user_platform.lower() == 'nvidia':
        command='hipcc -I src/samples/Common
src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.cu.cpp
src/samples/Samples/0_Introduction/simpleMPI/simpleMPI_hipified.cpp
-lmpi -o src/samples/Samples/0_Introduction/simpleMPI/simpleMPI.out'
        print(command)
        os.system(command)
    elif
x=='src/samples/Samples/0_Introduction/simpleSeparateCompilation/sim
```

```
pleDeviceLibrary.cu' or
x=='/src/samples/Samples/0_Introduction/simpleSeparateCompilation/si
mpleSeparateCompilation.cu' and user_platform.lower() == 'nvidia':
        command='hipcc -I src/samples/Common -fgpu-rdc
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleD
eviceLibrary.cu.cpp
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleS
eparateCompilation.cu.cpp -o
src/samples/Samples/0_Introduction/simpleSeparateCompilation/simpleS
eparateCompilation.out'
        print(command)
        os.system(command)
    elif
x=='src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu' and
user_platform.lower() == 'nvidia':
        command='hipcc -I src/samples/Common -fopenmp
src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.cu.cpp -o
src/samples/Samples/0_Introduction/cudaOpenMP/cudaOpenMP.out'
        print(command)
        os.system(command)
    elif user_platform.lower()=='nvidia':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or
file.endswith(".cu.cpp"):
                cpp.append(file)
    elif user_platform.lower()=='amd':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or
file.endswith(".cu.hip"):
                cpp.append(file)




    cpp = [p+'/'+y for y in cpp]
    command='hipcc -I src/samples/Common -I '+cuda_path+' '+'
'.join(cpp)+' -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out'
    print(command)
    os.system(command)
```

The following code was added to the generate() function:

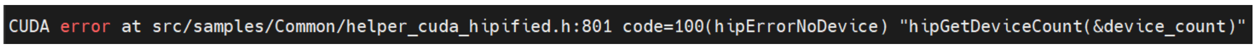


```
if user_platform.lower()=='nvidia' :
    for elem in listOfFiles:
        if elem.endswith('.cu'):
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    elem2=elem+'.cpp'
                    if os.path.exists(elem2)==False:
                        print('Writing to '+elem2)
                        with open(elem+'.hip','r') as f1, open(elem2,'a') as f2:
                            for line in f1:
                                f2.write(line)
```

This code checks if the user platform is "nvidia". If it is, it loops through a list of files named "listOfFiles". For each file in this list that ends with ".cu", it checks if the file is present in the "final_ignored_samples.txt" file. If it is, it prints a message indicating that the sample is being ignored. If it is not present in the "final_ignored_samples.txt" file, it creates a new file with the same name as the ".cu" file but with a ".cpp" extension and writes the contents of the ".hip" file to it. The code is essentially converting the ".hip" files to ".cpp" files for samples that are not being ignored.

4.2 ISSUES and TROUBLESHOOTING (NVIDIA PLATFORM)

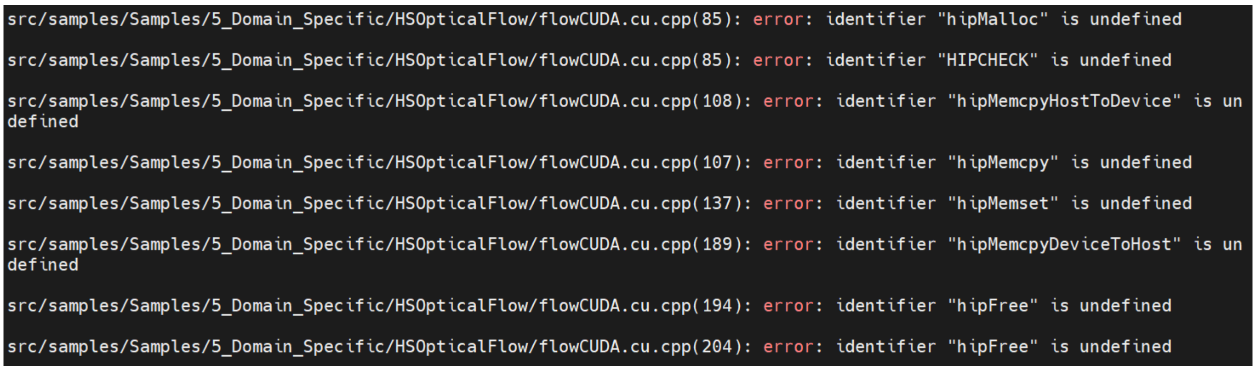
On beginning phase of testing,the following error can be seen while compilation and execution phases:



```
CUDA error at src/samples/Common/helper_cuda_hipified.h:801 code=100(hipErrorNoDevice) "hipGetDeviceCount(&device_count)"
```

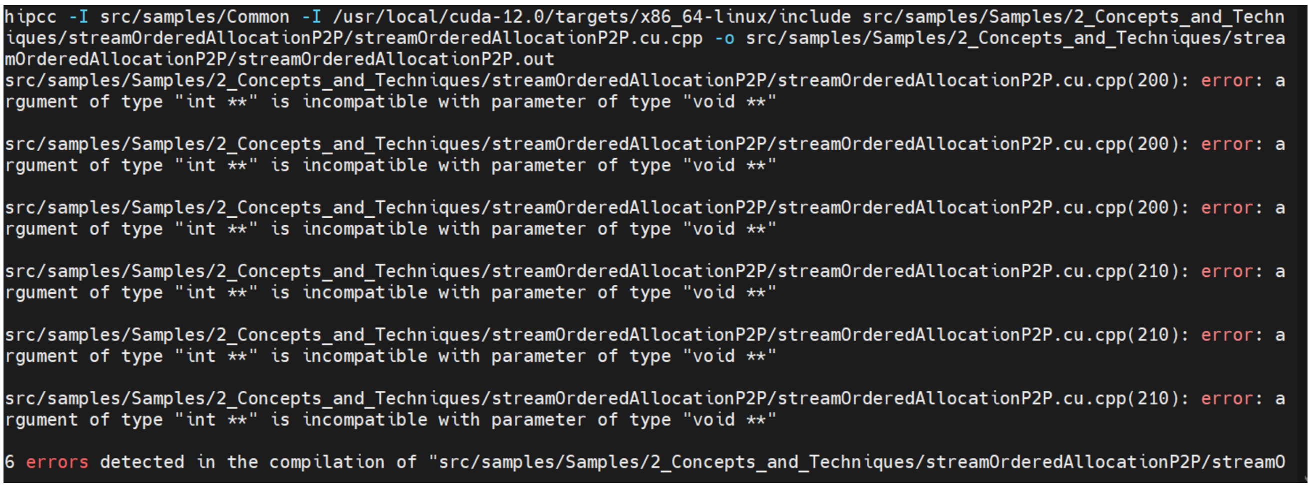
The issue was resolved by exporting the environmental paths to CUDA Toolkit.

The additional patches which were created for proper running of the samples on an Nvidia devices solved,to some extent,the following problems:

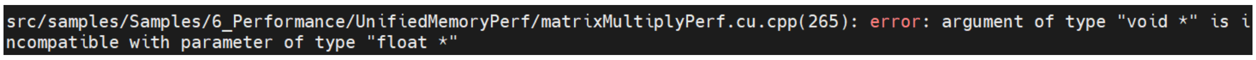


```
src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(85): error: identifier "hipMalloc" is undefined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(85): error: identifier "HIPCHECK" is undefined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(108): error: identifier "hipMemcpyHostToDevice" is un
defined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(107): error: identifier "hipMemcpy" is undefined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(137): error: identifier "hipMemset" is undefined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(189): error: identifier "hipMemcpyDeviceToHost" is un
defined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(194): error: identifier "hipFree" is undefined

src/samples/Samples/5_Domain_Specific/HSOpticalFlow/flowCUDA.cu.cpp(204): error: identifier "hipFree" is undefined
```

HIP Runtime had to be redefined in the new files as they were getting omitted.

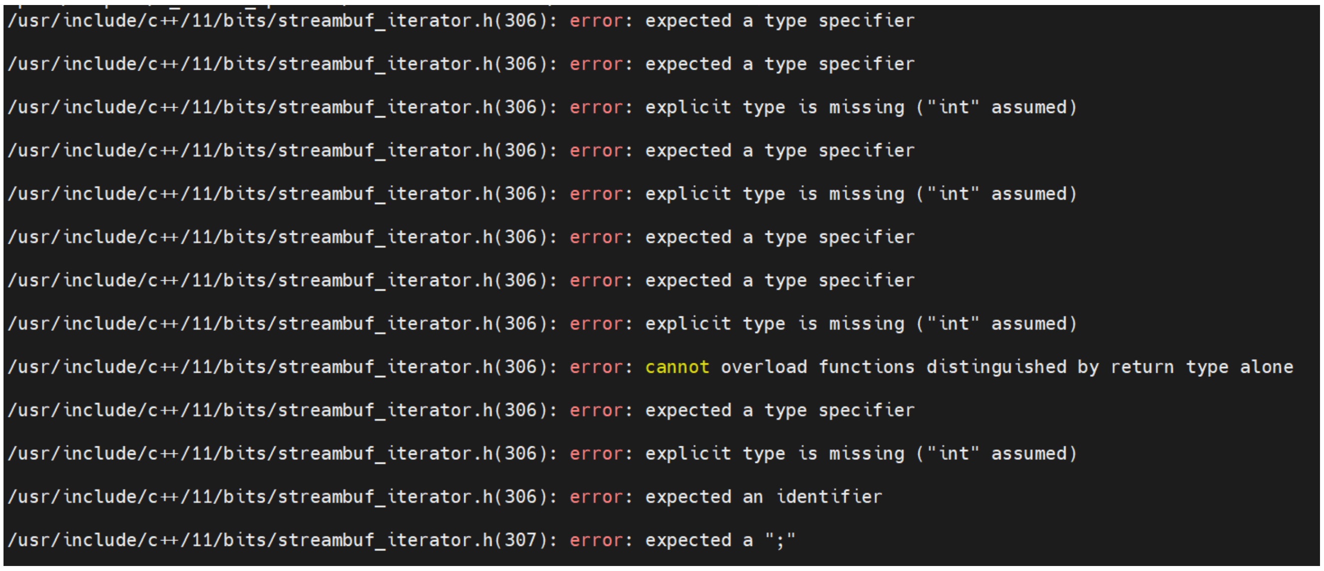


```
hipcc -I src/samples/Common -I /usr/local/cuda-12.0/targets/x86_64-linux/include src/samples/Samples/2_Concepts_and_Techn
iques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp -o src/samples/Samples/2_Concepts_and_Techniques/strea
mOrderedAllocationP2P/streamOrderedAllocationP2P.out
src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(200): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(200): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(200): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(210): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(210): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamOrderedAllocationP2P.cu.cpp(210): error: a
rgument of type "int **" is incompatible with parameter of type "void **"

6 errors detected in the compilation of "src/samples/Samples/2_Concepts_and_Techniques/streamOrderedAllocationP2P/streamO
```



```
src/samples/Samples/6_Performance/UnifiedMemoryPerf/matrixMultiplyPerf.cu.cpp(265): error: argument of type "void *" is i
ncompatible with parameter of type "float *"
```

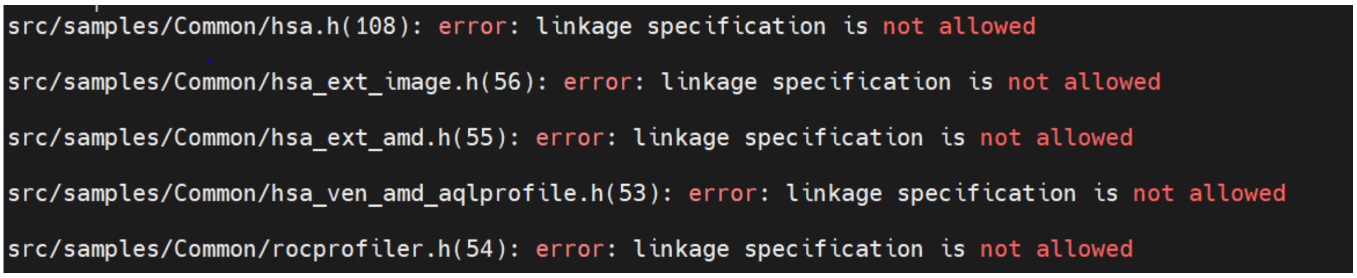
The above samples were resolved by explicit type-casting.



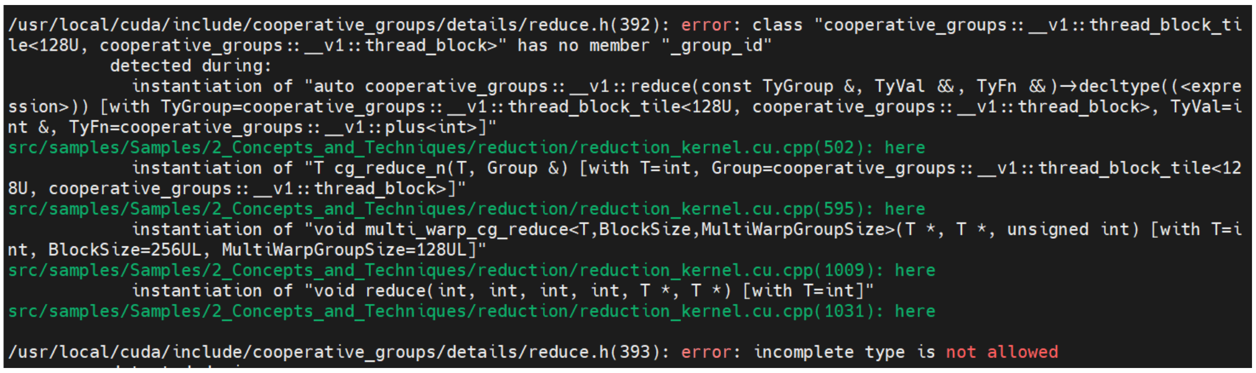
```
/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: explicit type is missing ("int" assumed)

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: explicit type is missing ("int" assumed)

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: explicit type is missing ("int" assumed)

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: cannot overload functions distinguished by return type alone

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected a type specifier

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: explicit type is missing ("int" assumed)

/usr/include/c++/11/bits/streambuf_iterator.h(306): error: expected an identifier

/usr/include/c++/11/bits/streambuf_iterator.h(307): error: expected a ";"
```

The issue was resolved by ensuring that the defined header files followed the conventional way of defining header files by first declaring the system header files, followed by HIP Runtime header files and finally the header files detailing the project codes and utility functions.

The following errors were unresolved but their corresponding executables can be run which would give different results.



```
src/samples/Common/hsa.h(108): error: linkage specification is not allowed

src/samples/Common/hsa_ext_image.h(56): error: linkage specification is not allowed

src/samples/Common/hsa_ext_amd.h(55): error: linkage specification is not allowed

src/samples/Common/hsa_ven_amd_aqlprofile.h(53): error: linkage specification is not allowed

src/samples/Common/rocprofiler.h(54): error: linkage specification is not allowed
```



```
/usr/local/cuda/include/cooperative_groups/details/reduce.h(392): error: class "cooperative_groups::__v1::thread_block_ti
le<128U, cooperative_groups::__v1::thread_block>" has no member "_group_id"
          detected during:
            instantiation of "auto cooperative_groups::__v1::reduce(const TyGroup &, TyVal &&, TyFn &&)->decltype((<expre
ssion>)) [with TyGroup=cooperative_groups::__v1::thread_block_tile<128U, cooperative_groups::__v1::thread_block>, TyVal=i
nt &, TyFn=cooperative_groups::__v1::plus<int>]"
src/samples/Samples/2_Concepts_and_Techniques/reduction/reduction_kernel.cu.cpp(502): here
            instantiation of "T cg_reduce_n(T, Group &) [with T=int, Group=cooperative_groups::__v1::thread_block_tile<12
8U, cooperative_groups::__v1::thread_block>]"
src/samples/Samples/2_Concepts_and_Techniques/reduction/reduction_kernel.cu.cpp(595): here
            instantiation of "void multi_warp_cg_reduce<T,BlockSize,MultiWarpGroupSize>(T *, T *, unsigned int) [with T=i
nt, BlockSize=256UL, MultiWarpGroupSize=128UL]"
src/samples/Samples/2_Concepts_and_Techniques/reduction/reduction_kernel.cu.cpp(1009): here
            instantiation of "void reduce(int, int, int, int, T *, T *) [with T=int]"
src/samples/Samples/2_Concepts_and_Techniques/reduction/reduction_kernel.cu.cpp(1031): here

/usr/local/cuda/include/cooperative_groups/details/reduce.h(393): error: incomplete type is not allowed
```

In conclusion, the project gave expected results in terms of usage and results.

4.3 ISSUES and TROUBLESHOOTING (MI100 and MI200)

While generation and compilation had been carried out without any issues, a bugged nature in executions regarding peer-to-peer samples was noticed where the execution would get stuck at a particular part.The tests were not present in coverage.On previous devices, the execution would simply waive the test as no connection could be established.

Using the roc-mem.sh bash script, we could see an increase in the memory being used but the sample was not executing after a marked point.On deliberation,it was found that a new change had to implemented in the **grub** file.

To troubleshoot the bug,the following steps needed to be carried out:

- The grub file at /etc/default/grub needed to be modified.The file was present in all linus distros.

For example :
If you want to add in grub "amd_iommu=on iommu=pt"
edit " sudo vi /etc/default/grub" and save it, the field "GRUB_CMDLINE_LINUX_DEFAULT"

```
GRUB_CMDLINE_LINUX_DEFAULT="amd_iommu=on iommu=pt"
```

For RHEL/CentOS based, you may see field name as "GRUB_CMDLINE_LINUX".
For Ubuntu/SLES, you may see field name as "GRUB_CMDLINE_LINUX_DEFAULT"

- Update grub
  Ubuntu : sudo update-grub
  CentOS/RHEL/SLES : sudo grub2-mkconfig -o /boot/grub2/grub.cfg
- >Update your RAMFs image :
  Ubuntu **: sudo update-initramfs -u -k `uname -r`**
  CentOS/RHEL/SLES : **sudo dracut -f**
- Reboot your machine
- Check the update and new change is done or not

```
$ cat /proc/cmdline
  BOOT_IMAGE=/vmlinuz-5.13.0-40-generic root=/dev/mapper/ubuntu--vg-ubuntu--lv
ro amd_iommu=on iommu=pt
```

If machine is PCI based then AMD_IOMMU =ON , iommu=pt were needed parameters in order for successful execution of P2P-based samples.

CHAPTER 5:RESULTS

5.1 Conclusion

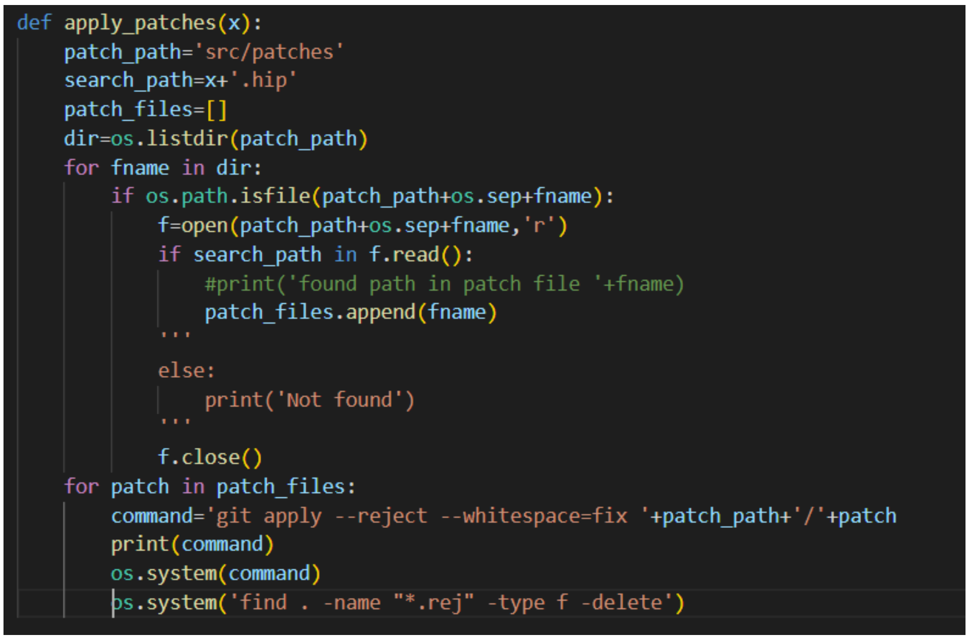
Testhipify adheres to not be dependent on the makefiles of any sample.It has provisions for processing single or multiple samples in a folder.The -h command would give the methods of usage of every function.The repository also contains the executable files of samples which were successfully processed 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.

5.2 Future Enhancements

Some provisions that can be added in this project are as follows:

1.Automate patch applying by seeking path to CUDA samples in every patches available and apply them on detection as patches includes references to other samples also.

For this purpose, another script based on the testing_experimental.py is created where we modify the apply_patches function to take a string argument x.The function searches for patch files in a specified directory and applies any patches that contains the path to the sample.



```python
def apply_patches(x):
    patch_path='src/patches'
    search_path=x+'.hip'
    patch_files=[]
    dir=os.listdir(patch_path)
    for fname in dir:
        if os.path.isfile(patch_path+os.sep+fname):
            f=open(patch_path+os.sep+fname,'r')
            if search_path in f.read():
                #print('found path in patch file '+fname)
                patch_files.append(fname)
            '''
            else:
                print('Not found')
            '''
            f.close()
    for patch in patch_files:
        command='git apply --reject --whitespace=fix '+patch_path+'/'+patch
        print(command)
        os.system(command)
        os.system('find . -name "*.rej" -type f -delete')
```

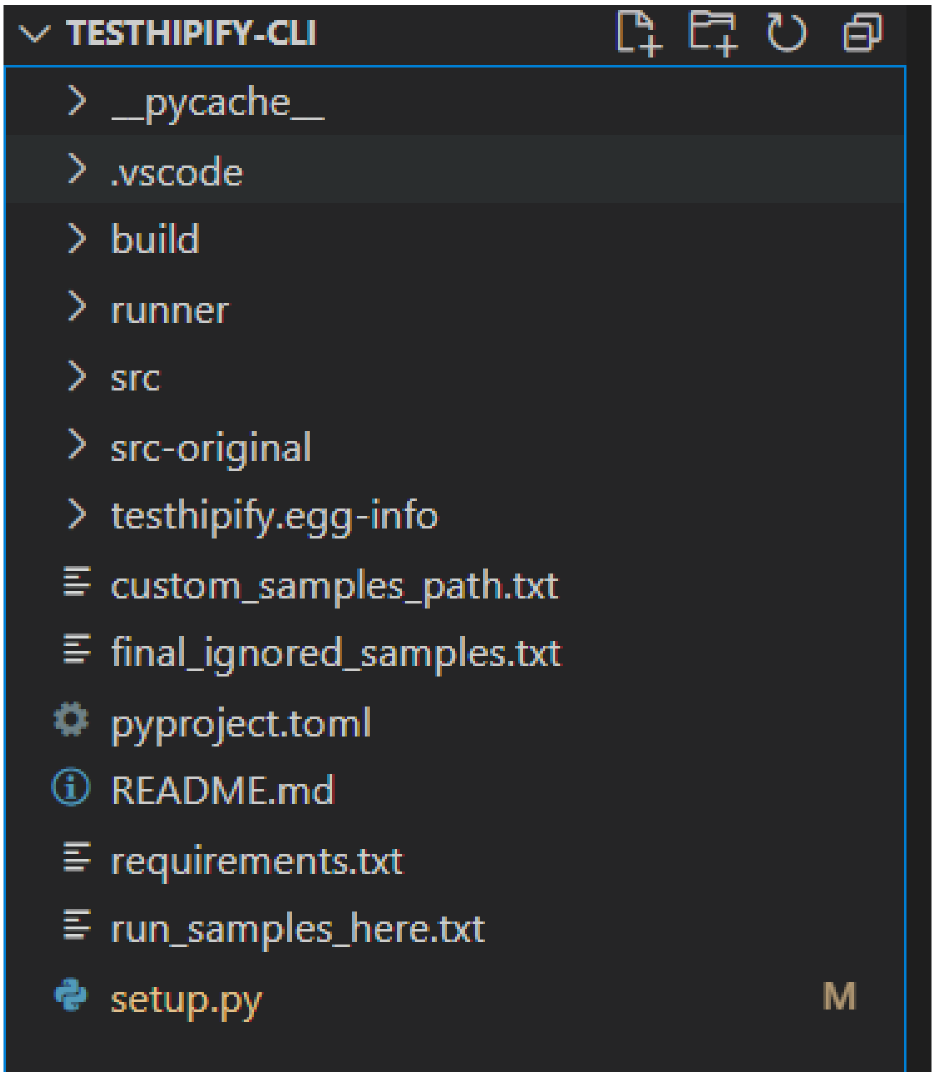
The function begins by defining a string variable patch_path which specifies the directory where the patch files are located. The function then defines another string variable search_path which is constructed by concatenating the argument x with the file extension .hip. The function creates an empty list called patch_files which will be used to store the names of the patch files that match the search path. The function uses the os.listdir function to get a list of allFor each file in the directory, the function checks if it is a regular file using the os.path.isfile function. If the file is a regular file, it is opened and searched for the search path using the read method. If the search path is found in the file, the name of the file is appended to the patch_files list. the files in the patch directory. It then iterates over this list using a for loop. For each file in the directory, the function checks if it is a regular file using the os.path.isfile function. If the file is a regular file, it is opened and searched for the search path using the read method. If the search path is found in the file, the name of the file is appended to the patch_files list.

The function then iterates over the list of patch files found in step 5. For each patch file, it constructs a Git command to apply the patch using the git apply command with the --reject and --whitespace=fix options. The command is printed to the console using the print function, and then executed using the os.system function.After applying the patch, the function deletes any reject files that may have been created using the find command and the os.system function.For non-disruption purposes,this function is not included in the main testhipify script.

2.Create a CLI Tool making it much easier for the user to make use of testHIPIFY.

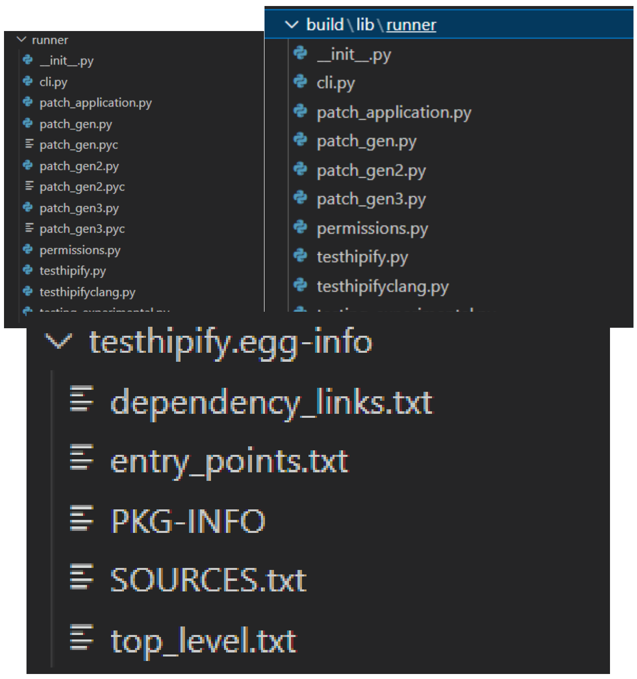
To make it easier for users to use testhipify without need of executing the main script.The directory layout is manipulated such as testHIPFY can be built and run on AMD systems as in other tools like HIPIFY-Perl or HIPIFY-clang.

We change the file structuring of the repository as follows:



```
TESTHIPIFY-CLI
  > __pycache__
  > .vscode
  > build
  > runner
  > src
  > src-original
  > testhipify.egg-info
    custom_samples_path.txt
    final_ignored_samples.txt
    pyproject.toml
    README.md
    requirements.txt
    run_samples_here.txt
    setup.py                         M
```

Screenshot:CLI Tool Layout

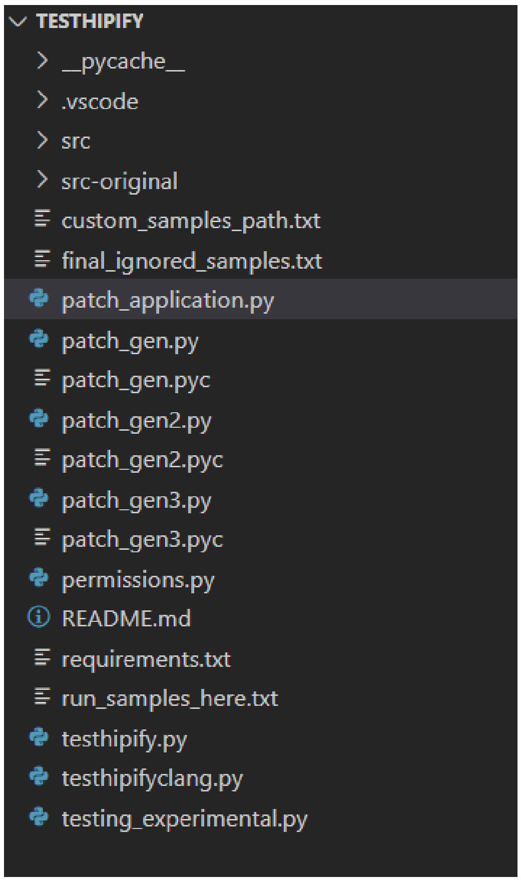


```
runner
  __init__.py
  cli.py
  patch_application.py
  patch_gen.py
  patch_gen.pyc
  patch_gen2.py
  patch_gen2.pyc
  patch_gen3.py
  patch_gen3.pyc
  permissions.py
  testhipify.py
  testhipifyclang.py
```

```
build \ lib \ runner
  __init__.py
  cli.py
  patch_application.py
  patch_gen.py
  patch_gen2.py
  patch_gen3.py
  permissions.py
  testhipify.py
  testhipifyclang.py
```

```
testhipify.egg-info
  dependency_links.txt
  entry_points.txt
  PKG-INFO
  SOURCES.txt
  top_level.txt
```

Screenshot:src and src-original contains the samples while runner,build,and info file contains the scripts and informations on dependancies



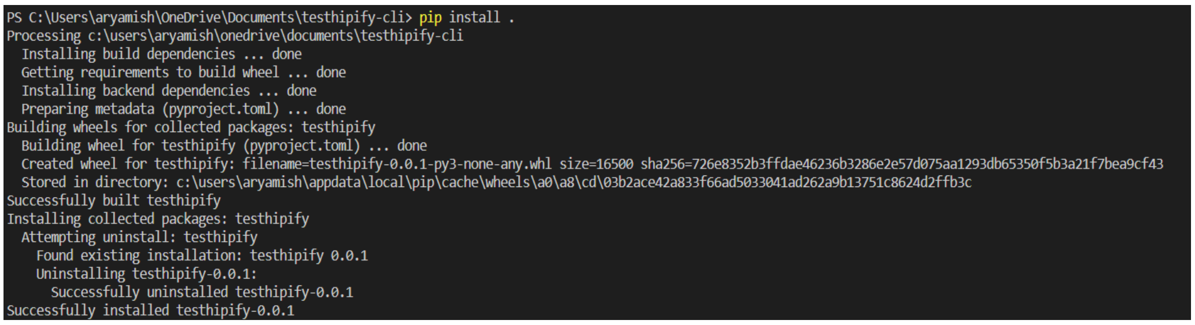
```
TESTHIPIFY
  > __pycache__
  > .vscode
  > src
  > src-original
  custom_samples_path.txt
  final_ignored_samples.txt
  patch_application.py
  patch_gen.py
  patch_gen.pyc
  patch_gen2.py
  patch_gen2.pyc
  patch_gen3.py
  patch_gen3.pyc
  permissions.py
  README.md
  requirements.txt
  run_samples_here.txt
  testhipify.py
  testhipifyclang.py
  testing_experimental.py
```

Screenshot:Structure for testHIPIFY tool as per guidelines

We create a setup.py file and a pyproject.toml file in the root directory which tells Python's setup tools some basic information about the package and how to install it.We extract the argparse functions and convert them to cli.py script while importing all the functions from testHIPIFY.By using the setuptools module,we define the name,version,packages required and entry points.After completion,the user would only need to clone the repository from this link: https://github.com/AryamanAMD/testhipify-cli ,navigate to the directory and install the package by the command ‘pip install .’

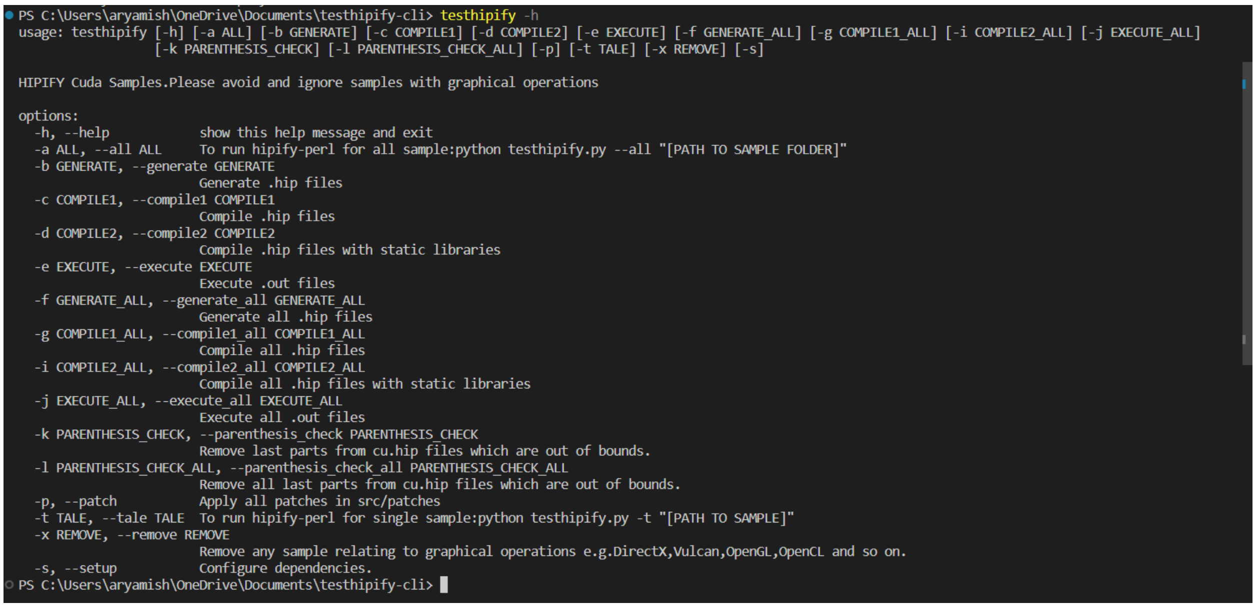
The testHIPIFY modules would then be accessible for usage for any directory without needing to invoke python repeatedly.

Installation:



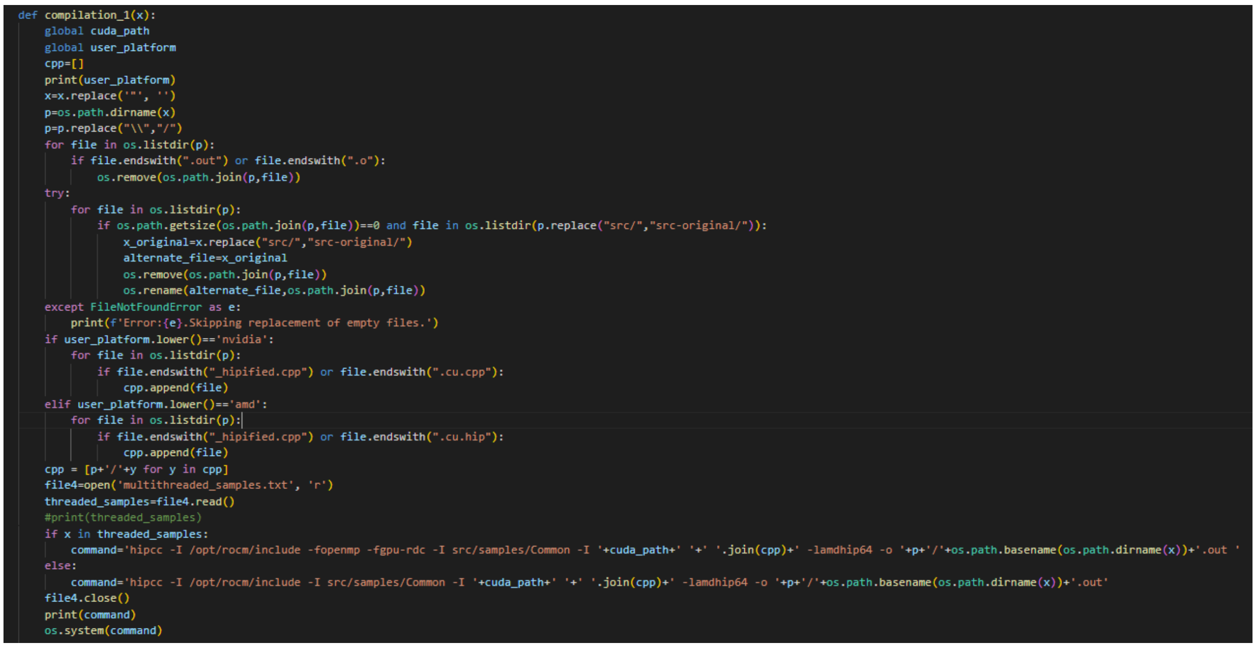
```
PS C:\Users\aryamish\OneDrive\Documents\testhipify-cli> pip install .
Processing c:\users\aryamish\onedrive\documents\testhipify-cli
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Installing backend dependencies ... done
  Preparing metadata (pyproject.toml) ... done
Building wheels for collected packages: testhipify
  Building wheel for testhipify (pyproject.toml) ... done
  Created wheel for testhipify: filename=testhipify-0.0.1-py3-none-any.whl size=16500 sha256=726e8352b3ffdae46236b3286e2e57d075aa1293db65350f5b3a21f7bea9cf43
  Stored in directory: c:\users\aryamish\appdata\local\pip\cache\wheels\a0\a8\cd\03b2ace42a833f66ad5033041ad262a9b13751c8624d2ffb3c
Successfully built testhipify
Installing collected packages: testhipify
  Attempting uninstall: testhipify
    Found existing installation: testhipify 0.0.1
    Uninstalling testhipify-0.0.1:
      Successfully uninstalled testhipify-0.0.1
Successfully installed testhipify-0.0.1
```

Output:



```
PS C:\Users\aryamish\OneDrive\Documents\testhipify-cli> testhipify -h
usage: testhipify [-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.
PS C:\Users\aryamish\OneDrive\Documents\testhipify-cli>
```

3.Detect OpenMP/MPI presence in samples and add flags for their compilation.As there are multiple samples requiring parallel computing, a measure can be taken for inclusion of flags using a list to be added into the compilation command upon processing of samples not covered in NVIDIA’s repository.This issue is rectified by using an optimized version of compilation_1() and compilation_2() functions.



```python
def compilation_1(x):
    global cuda_path
    global user_platform
    cpp=[]
    print(user_platform)
    x=x.replace('"', '')
    p=os.path.dirname(x)
    p=p.replace("\\","/")
    for file in os.listdir(p):
        if file.endswith(".out") or file.endswith(".o"):
            os.remove(os.path.join(p,file))
    try:
        for file in os.listdir(p):
            if os.path.getsize(os.path.join(p,file))==0 and file in os.listdir(p.replace("src/","src-original/")):
                x_original=x.replace("src/","src-original/")
                alternate_file=x_original
                os.remove(os.path.join(p,file))
                os.rename(alternate_file,os.path.join(p,file))
    except FileNotFoundError as e:
        print(f'Error:{e}.Skipping replacement of empty files.')
    if user_platform.lower()=='nvidia':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.cpp"):
                cpp.append(file)
    elif user_platform.lower()=='amd':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.hip"):
                cpp.append(file)
    cpp = [p+'/'+y for y in cpp]
    file4=open('multithreaded_samples.txt', 'r')
    threaded_samples=file4.read()
    #print(threaded_samples)
    if x in threaded_samples:
        command='hipcc -I /opt/rocm/include -fopenmp -fgpu-rdc -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out '
    else:
        command='hipcc -I /opt/rocm/include -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out'
    file4.close()
    print(command)
    os.system(command)
```

The function compiles a set of input files for execution on either an NVIDIA or AMD platform, depending on the value of the user_platform global variable. It first cleans up any previously compiled files in the directory p, which is the directory of the input file x. Then, it replaces any empty files in p with non-empty files from a corresponding directory. Next, it appends any files that end with _hipified.cpp, .cu.cpp, or .cu.hip in p to a list named cpp, based on the value of user_platform. It then reads the contents of a file named multithreaded_samples.txt and stores it in a variable named threaded_samples. If the input file x is present in threaded_samples, the command variable is set to compile the input files with OpenMP and with reduced GPU code, while otherwise, the command variable is set to compile the input files without these flags. Finally, the function executes the command using the os.system function.The same steps are carried out for compilation_2() but it’s not required as the -use-staticlib flag has become deprecated in latest ROCm versions.

4.Automated Dependencies Installer

To make it easier for users to ensure that all packages are accounted for in a single run without user input, the base setup() function is replaced by 2 functions to accommodate installation automatically and manually.

This code is a setup script for installing and configuring the necessary software and libraries for a specific project or program. It is divided into several parts, each of which checks for the existence of specific software and libraries, installs missing dependencies, and sets up environment variables.

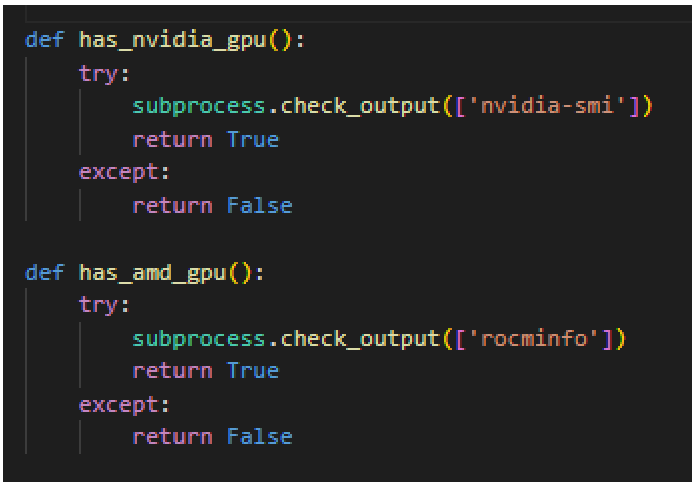
First, the script checks for the presence of an Nvidia or AMD GPU by calling the "has_nvidia_gpu()" and "has_amd_gpu()" functions. If a GPU is detected, the user platform is set to Nvidia or AMD, respectively. Next, the script writes the user platform and other configuration variables to a file named "config.txt".

Then, the script checks for the presence of the GCC compiler, installs it if necessary, and then installs the necessary Python packages specified in the requirements.txt file. Next, the script installs the OpenMP library and sets the number of threads to be used by the library.

Finally, the script checks for the presence of MPI (Message Passing Interface) and installs it if necessary. It downloads the OpenMPI source code, compiles it, and installs it. It then sets the PATH and LD_LIBRARY_PATH environment variables and tests the MPI installation by running the "mpirun --version" command.

The base setup() function is renamed as setup2() instead and allows option to download the latest version of CUDA Samples by giving a user input along with generating hipified versions of all header and C++ files, allowing for linking of the programs to finally generate an executeable.

5.Detection of Platform.



```python
def has_nvidia_gpu():
    try:
        subprocess.check_output(['nvidia-smi'])
        return True
    except:
        return False

def has_amd_gpu():
    try:
        subprocess.check_output(['rocminfo'])
        return True
    except:
        return False
```

The code consists of two Python functions, has_nvidia_gpu() and has_amd_gpu(), that check if the system has an NVIDIA or AMD GPU, respectively.

The has_nvidia_gpu() function uses the subprocess.check_output() function to execute the command nvidia-smi. If the command executes successfully, it means that NVIDIA GPU is present and the function returns True. If the command fails, it means that the system does not have an NVIDIA GPU, and the function returns False.

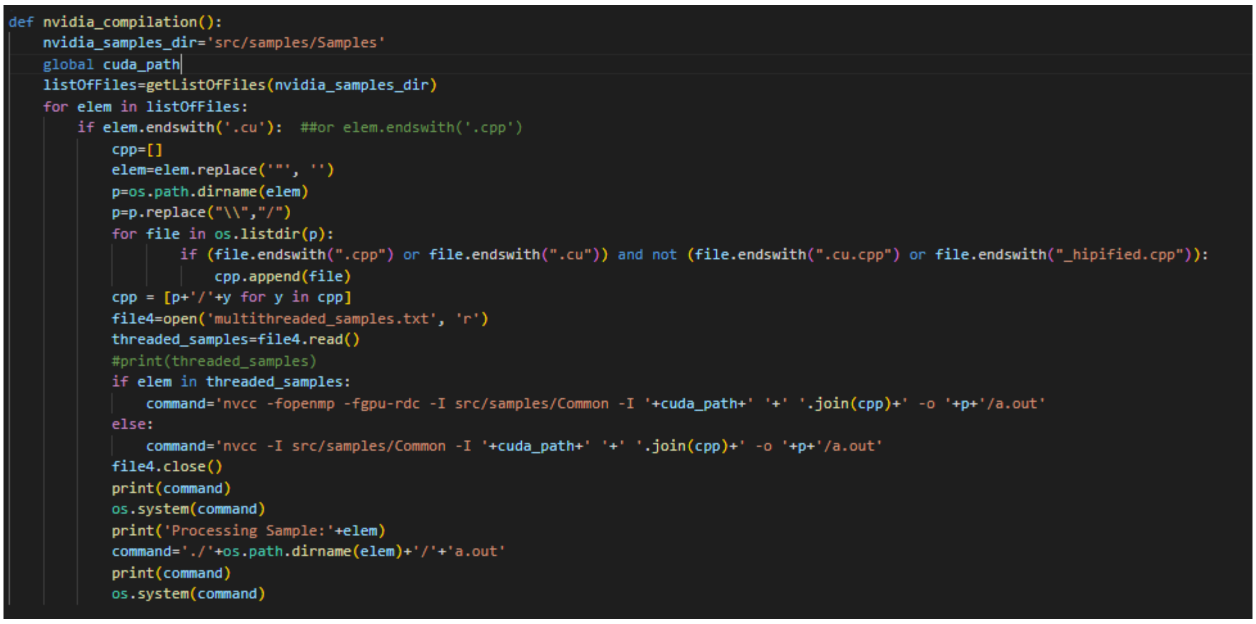
The has_amd_gpu() function uses the subprocess.check_output() function to execute the command rocminfo. If the command executes successfully, it means that AMD GPU is present and the function returns True. If the command fails, it means that the system does not have an AMD GPU, and the function returns False.

Both functions utilize exception handling to gracefully handle any errors that may occur during the execution of the commands. If the check_output() function throws an exception, it means that the command failed to execute, which is considered as an indication that the corresponding GPU is not present in the system.In both cases, the user has to install and verify CUDA manually and the program would only ask to verify the installation path.

6.Editing of Configuration file and Global Variable

This code attempts to open a file named 'config.txt' in read-only mode, and then reads each line in the file. For each line, it splits the string by the '=' character and creates a dictionary with the first part of the split string as the key and the second part as the value. It then assigns the values of the keys "user_platform" and "cuda_path" to the variables "user_platform" and "cuda_path" respectively. If the file "config.txt" is not found, it raises a FileNotFoundError exception.This makes the program much easier to use as it can be manually defined by the user or let the program decide the 2 parameters.

7.Nvidia CUDA-based Compilation



```python
def nvidia_compilation():
    nvidia_samples_dir='src/samples/Samples'
    global cuda_path
    listOfFiles=getListOfFiles(nvidia_samples_dir)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            cpp=[]
            elem=elem.replace('"', '')
            p=os.path.dirname(elem)
            p=p.replace("\\","/")
            for file in os.listdir(p):
                    if (file.endswith(".cpp") or file.endswith(".cu")) and not (file.endswith(".cu.cpp") or file.endswith("_hipified.cpp")):
                        cpp.append(file)
            cpp = [p+'/'+y for y in cpp]
            file4=open('multithreaded_samples.txt', 'r')
            threaded_samples=file4.read()
            #print(threaded_samples)
            if elem in threaded_samples:
                command='nvcc -fopenmp -fgpu-rdc -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -o '+p+'/a.out'
            else:
                command='nvcc -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -o '+p+'/a.out'
            file4.close()
            print(command)
            os.system(command)
            print('Processing Sample:'+elem)
            command='./'+os.path.dirname(elem)+'/'+'a.out'
            print(command)
            os.system(command)
```

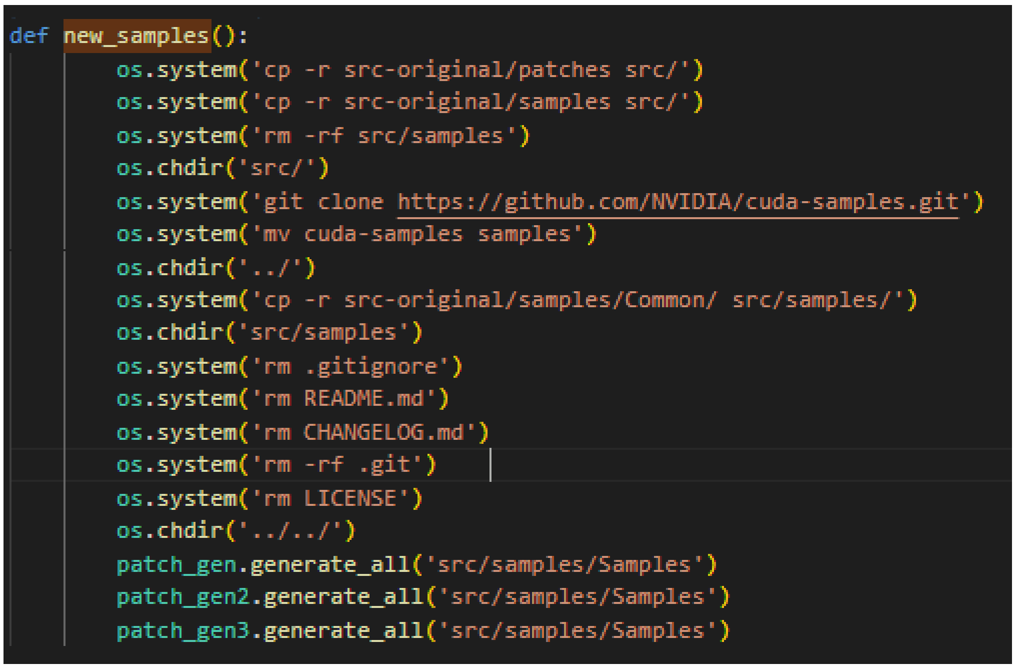
It sets the directory of NVIDIA CUDA samples to src/samples/Samples and makes the cuda_path variable global.It uses the getListOfFiles function to get a list of all the files in the NVIDIA CUDA samples directory.For each file in the list, the function checks if it has the .cu extension (the extension used for CUDA code files). If it does, the function looks for other .cpp and .cu files in the same directory and adds them to a list called cpp.

The function opens a file called multithreaded_samples.txt (which should contain a list of the names of CUDA samples that use multithreading) and reads its contents into a variable called threaded_samples.If the current file is in threaded_samples, the function adds the flags -fopenmp and -fgpu-rdc to the nvcc command. These flags enable OpenMP and GPU RDC (relocatable device code), respectively.

The function then constructs a nvcc command by concatenating the cuda_path variable, the -I flag (to specify the directory of common headers), the cpp list, and the output filename.It prints the nvcc command to the console and executes it using the os.system function.Finally, the function constructs a command to run the compiled code (by concatenating the directory name, the executable filename, and the ./ prefix), prints it to the console, and executes it using os.system.

Note that this function assumes that you have the NVIDIA CUDA toolkit installed and configured correctly, and that you have the sample codes installed in the default directory (/usr/local/cuda/samples/ on Linux). If your setup is different, you may need to modify the function to reflect your configuration.

7.Function to only download CUDA Samples without taking care of any dependencies.



```
def new_samples():
        os.system('cp -r src-original/patches src/')
        os.system('cp -r src-original/samples src/')
        os.system('rm -rf src/samples')
        os.chdir('src/')
        os.system('git clone https://github.com/NVIDIA/cuda-samples.git')
        os.system('mv cuda-samples samples')
        os.chdir('../')
        os.system('cp -r src-original/samples/Common/ src/samples/')
        os.chdir('src/samples')
        os.system('rm .gitignore')
        os.system('rm README.md')
        os.system('rm CHANGELOG.md')
        os.system('rm -rf .git')
        os.system('rm LICENSE')
        os.chdir('../../')
        patch_gen.generate_all('src/samples/Samples')
        patch_gen2.generate_all('src/samples/Samples')
        patch_gen3.generate_all('src/samples/Samples')
```

The new_samples() function is used to download and copy new CUDA samples from NVIDIA's GitHub repository to the local project directory. First, it copies the original patches and samples from the src-original directory to the src directory. It then removes the src/samples directory and changes the working directory to src/. It uses git clone to download the CUDA samples repository and renames the downloaded folder to samples. The function then copies the Common folder from the original samples to the new samples directory. Finally, it removes unnecessary files such as .gitignore, README.md, CHANGELOG.md, .git and LICENSE. Once done, it changes the working directory back to the project root directory.The program then calls on the patch_gen functions to hipify all the .cuh,.h and .cpp extension files to allow the user to only work with the sample files.

8.A program can be made such that user can pass parameters required on the CUDA Installation page and a web scraper can be used to take the scan page elements which contains the commands to install CUDA and the os module to execute the scraped commands.

9.A program can be made such that all patches work consistently on every distinct devices and ensure that no hipified file has their contents purged by any of the patches contained.

10.Setup measures to install and build ROCm,HIPCC and Hipify-perl or Hipify-Clang can be added in the setup functions.

11.TestHIPIFY is an ever evolving project,such that any updation to ROCm or HIP Runtime can be automatically adjusted in the main script by moduling or automation,wit.

5.3 FINAL ADDRESS

After 9 months of deliberation, I hereby leave behind a powerful tool for ROCm users to run a wide range of Nvidia samples.I believe in the capable hands of AMD employees that this tool will evolve and will run all kinds of translated CUDA samples on ROCm supported hardware.At this point,the developmental stage brought forth some underlying problems which have been discussed in the 'Issues' section of this documentation.TestHIPIFY can also be used as a CLI tool, simplifying the usage of the project even more.

Even after documenting every single aspect,more features got added over time,as this list goes:

- "-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.

Thank you AMD, and all the best for the users of TestHIPIFY,hope you enjoy and reach your destination target with this tool.

5.4 APPENDIX

1-DEVELOPMENT CYCLE LOGS

Task Summary	Date Started	Status	Remarks
Sample Correction and Patch Creation	21/9/22	Work in Progress:Macros references needed	
Stage 2 Docker Image Pull for Hipify-Clang	20/9/22	Completed by pulling 66GB of stage 2 docker for building and testing hipify-clang;Command line:**sudo docker run -d -it --privileged --shm-size=1G --name=stage_2_docker --network=host --device=/dev/kfd --device=/dev/dri --group-add video --cap-add=SYS_PTRACE --security-opt seccomp=unconfined --ipc=host -v $HOME/dockerx:/dockerx compute-artifactory.amd.com:5000/rocm-plus-docker/compute-rocm-dkms-no-npi-hipclang:10890-ubuntu-22.04-stg2**	
Base Script for Hipifying using Perl Script in Python Environment	16/8/22	Completed with Maximum Optimization	
Continue iterations of samples and avoid samples containing OpenGL and OpenCL related header files	28/9/22	ROCm Stack Hipify doesnt support OPenGL and OpenCL functions.Alternate method required for graphical operations and multiple processor computation(likely) or left as it is. **UPDATE:Found unrelated Graphic operation headers referencing DirectX and Vulcan Runtime API along with DRM-based APIs.Added function to skip such files.**	
Substitute Cuda Error Check macros with HIPCHECK during runtime of program	28/9/22	Finding substitutes to replace value of macro without dsisrupting basic function of the testhipify script **UPDATE:Functionality added.**	
Building HIPIFY-Clang by attaching docker image	28/9/22	To be conducted in office premises to execute succesfully in Stage 2 Docker Image.	

Pull Docker Image using TMUX	23/9/22	First attempt was carried out in session 1(interrupted due to IT/Server issues),Second attempt to pulling succesful in session 2.Command to attach to session:**tmux attach-session -t 2**	
Removal of Hard Coding to account for compiler to find custom paths containing required header files.	26/9/22	Not required as samples needed to be converted can be kept in the src folder of the repository for conversion.Script is work in progress(if required) but would likely find wrong subdirectories,increase runtime and execute wrong header files instead.	Done.
All Hands Meet at Whitefeld	27/9/22	Understood upcoming product plans,revenue streams and communicated about product limitations in GPU Encoders.	
UIF Meeting	19/9/22	Hipify Activity-priority objective	

Continue iterations of samples and avoid samples containing OpenGL and OpenCL related header files	8/10/2022	ROCm Stack Hipify doesnt support OPenGL and OpenCL functions.Alternate method required for graphical operations and multiple processor computation(likely) or left as it is. **UPDATE:Found unrelated Graphic operation headers referencing DirectX and Vulcan Runtime API along with DRM-based APIs.Added function to skip such files.**
Substitute Cuda Error Check macros with HIPCHECK during runtime of program	6/10/2022	Finding substitutes to replace value of macro without dsisrupting basic function of the testhipify script **UPDATE:Functionality added.**
Building HIPIFY-Clang by attaching docker image	28/9/2022	To be conducted in office premises to execute succesfully in Stage 2 Docker Image.

Continue iterations of samples and avoid samples containing OpenGL and OpenCL related header files	3/10/2022	ROCm Stack Hipify doesnt support OPenGL and OpenCL functions.Alternate method required for graphical operations and multiple processor computation(likely) or left as it is. **UPDATE:Found unrelated Graphic operation headers referencing DirectX and Vulcan Runtime API along with DRM-based APIs.Added function to skip such files.**	
Figure out why patches encodings are failing and remove .rej files	28/10/22	Done	
Add more functions to script to repeat particular procedures step-by-step	23/10/22	Done	
Provisions for sample exclusion	22/10/22	Done	
Build HIPIFY-clang succesfully	29/10/22	Done	
Rectify CUDA-kit dependency issue	25/10/22	Done	
Use Perl script template to accomodate HIPIFY-clang functions	3/11/2022	Done	
Create new documentation for every problems and solutions tried	14/11/22	Done	
Remove and update RocXT/ROCTracer functions	6/11/2022	Permanent Solution Achieved	
Debug for lastCudaError and findCudaDevice	16/11/22	WIP	findCudaDevice debugged
Faulty Sample causing ROCm Bug Detection	7/12/2022	Access to Device Lost;Ticket lodged	Sample is rendering all user permissions as read-only highlighted by green text in ls command
	9/12/2022	Access granted to alternate shareable device	Never run systemwideAtomics sample
	12/12/2022	Pulled Stage 2 Docker	
Recompilation required	14/12/22	Done	
Add provisions in script to include hipified cpp programs also	15/12/22	Done	

2-Main Driver Code

Testhipify.py

```
import os
import argparse
import fileinput
import os.path
from sys import platform
import patch_gen
import patch_gen2
import patch_gen3
import subprocess
try:
    with open('config.txt','r') as f:
            config_variables={variable.split("=")[0]:variable.split("=")[1].strip() for variable in f.readlines()}
    f.close()
    user_platform=config_variables["user_platform"]
    cuda_path=config_variables["cuda_path"]
except FileNotFoundError:
    pass
def getListOfFiles(dirName):
    listOfFile=os.listdir(dirName)
    allFiles=list()
    for entry in listOfFile:
        fullPath=os.path.join(dirName, entry)
        if os.path.isdir(fullPath):
            allFiles=allFiles+getListOfFiles(fullPath)
        else:
            allFiles.append(fullPath)

    return allFiles
def sorting(filename):
  infile = open(filename)
  words = []
  for line in infile:
    temp = line.split()
    for i in temp:
      words.append(i)
  infile.close()
  words.sort()
  outfile = open("final_ignored_samples1.txt", "w")
  for i in words:
    outfile.writelines(i)
    outfile.writelines("\n")
  outfile.close()
  with open('final_ignored_samples1.txt','r') as f:lines=f.readlines()
  os.remove('final_ignored_samples1.txt')
  with open('accused_samples.txt','r') as f:lines_to_remove=f.readlines()
  new_lines=[line for line in lines if line not in lines_to_remove]
  with open('final_ignored_samples.txt','w') as f:
        f.writelines(new_lines)
def prepend_line(file_name, line):
    result=check_for_word(file_name,line)
    if result==-1:
        p=os.path.dirname(file_name)
        file=open(file_name,'r')
        lines = file.readlines()
        for elem in lines:
            if elem == '#include <stdio.h>\n':
                index=lines.index(elem)
                lines.insert(index+1,line)
            else:
                continue
        with open(p+'/'+'a.cu.hip','w') as fp:
            for item in lines:
                fp.write(item)
        file.close()
        os.remove(file_name)
        os.rename(p+'/a.cu.hip', file_name)

def check_for_word(file_name,word):
    file = open(file_name, 'r')
    linelist = file.read()
    index=linelist.find(word)
    file.close()
    return index
def has_nvidia_gpu():
    try:
        subprocess.check_output(['nvidia-smi'])
        return True
    except:
        return False
def has_amd_gpu():
    try:
        subprocess.check_output(['rocminfo'])
        return True
    except:
        return False
def setup1():
    global cuda_path
    global user_platform
    global config_variables
    if has_nvidia_gpu():
        config_variables['user_platform']='Nvidia'
    elif has_amd_gpu():
        config_variables['user_platform']='AMD'
    print('If CUDA is not installed,refer to this link and follow the steps:https://developer.nvidia.com/cuda-
downloads')
```

```
    print('CUDA Path:'+cuda_path)
    with open('config.txt', 'w') as f:
        # f.write(str(user_platform))
        for variable, value in config_variables.items():
            f.write(f"{variable}={value}\n")
    f.close()
    os.system('gcc --version')
    user_input = 'gcc' # set to 'gcc' to install gcc compiler
    if user_input.lower() == 'gcc':
        os.system('sudo apt install gcc')
    user_input = 'requirements'
    if user_input.lower() == 'requirements':
        os.system('pip install -r requirements.txt')
    user_input = 'omp'
    if user_input.lower() == 'omp':
        os.system('sudo apt install libomp-dev')
        os.system('echo |cpp -fopenmp -dM |grep -i open')
        print('Enter number of threads ')
        x = 4
        os.system(f'export OMP_NUM_THREADS={x}')
        print("Always add -fopenmp flag on compilation.")
    user_input = 'mpi'
    if user_input.lower() == 'mpi':
        print('cd ~')
        os.chdir(os.path.expanduser("~"))
        print('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        os.system('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        print('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('mv -r ')
        print('cd openmpi-3.1.3')
        os.system('cd openmpi-3.1.3')
        os.chdir('openmpi-3.1.3')
        print('pwd')
        os.system('pwd')
        print('./configure --prefix=/usr/local/')
        os.system('./configure --prefix=/usr/local/')
        print('./configure --prefix=/usr/local/openmpi-3.1.3/')
        os.system('./configure --prefix=/usr/local/openmpi-3.1.3/')
        print('sudo make all install')
        os.system('sudo make all install')
        print('After make install is completed, mpirun or orterun executable should be at /usr/local/bin/.')
        print('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        os.system('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        print('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" >
$HOME/.bashrc')
        os.system('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" >
$HOME/.bashrc')
```

```
        print('export PATH=$PATH:/usr/local/bin')
        os.system('export PATH=$PATH:/usr/local/bin')
        print('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        os.system('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        print('source $HOME/.bashrc')
        os.system('source $HOME/.bashrc')
        print('mpirun --version')
        os.system('mpirun --version')
def new_samples():
        os.system('cp -r src-original/patches src/')
        os.system('cp -r src-original/samples src/')
        os.system('rm -rf src/samples')
        os.chdir('src/')
        os.system('git clone https://github.com/NVIDIA/cuda-samples.git')
        os.system('mv cuda-samples samples')
        os.chdir('../')
        os.system('cp -r src-original/samples/Common/ src/samples/')
        os.chdir('src/samples')
        os.system('rm .gitignore')
        os.system('rm README.md')
        os.system('rm CHANGELOG.md')
        os.system('rm -rf .git')
        os.system('rm LICENSE')
        os.chdir('../../')
        patch_gen.generate_all('src/samples/Samples')
        patch_gen2.generate_all('src/samples/Samples')
        patch_gen3.generate_all('src/samples/Samples')
def setup2():
    global cuda_path
    global user_platform
    global config_variables
    #cuda_path = '/usr/local/cuda-12.0/targets/x86_64-linux/include'
    print("Enter Nvidia or AMD as per your system specifications.")
    user_input=input()
    if user_input != '':
        config_variables['user_platform']=user_input
    print('Confirm the following CUDA Installation path for compilation:')
    print('CUDA Path:'+cuda_path)
    print('If Path is incorrect,please provide current path by typing CUDA or press any key to continue')
    user_input=input()
    if user_input.lower() == 'cuda':
        print('Enter path of your CUDA installation')
        config_variables['cuda_path']=input()
    with open('config.txt','w') as f:
        #f.write(str(user_platform))
        for variable, value in config_variables.items():
            f.write(f"{variable}={value}\n")
    f.close()
```

```
   os.system('gcc --version')
   print('Enter gcc to install gcc compiler, or any other button to continue.')
   user_input=input()
   if user_input.lower() == 'gcc':
      os.system('sudo apt install gcc')
   print("Enter 'requirements' to install python packages dependencies")
   user_input=input()
   if user_input.lower() == 'requirements':
      os.system('pip install -r requirements.txt')
   print("Enter 'samples' to install latest version of CUDA Samples")
   user_input=input()
   if user_input.lower() == 'samples':
      os.system('cp -r src-original/patches src/')
      os.system('cp -r src-original/samples src/')
      os.system('rm -rf src/samples')
      os.chdir('src/')
      os.system('git clone https://github.com/NVIDIA/cuda-samples.git')
      os.system('mv cuda-samples samples')
      os.chdir('../')
      os.system('cp -r src-original/samples/Common/ src/samples/')
      os.chdir('src/samples')
      os.system('rm .gitignore')
      os.system('rm README.md')
      os.system('rm CHANGELOG.md')
      os.system('rm -rf .git')
      os.system('rm LICENSE')
      os.chdir('../../')
   print("Enter 'generate' to hipify additional files.")
   user_input=input()
   if user_input.lower() == 'generate':
      patch_gen.generate_all('src/samples/Samples')
      patch_gen2.generate_all('src/samples/Samples')
      patch_gen3.generate_all('src/samples/Samples')
   print("Enter 'omp' to install OpenMP in your system, or any other button to continue.")
   user_input=input()
   if user_input.lower() == 'omp':
      os.system('sudo apt install libomp-dev')
      os.system('echo |cpp -fopenmp -dM |grep -i open')
      print('Enter number of threads ')
      x=int(input())
      os.system('export OMP_NUM_THREADS='+str(x))
      print("Always add -fopenmp flag on compilation.")
   print("Enter 'mpi' to install OpenMPI, or any other button to continue.It's better to install latest version
from this link manually:https://sites.google.com/site/rangsiman1993/comp-env/program-install/install-
openmpi")
   user_input=input()
   if user_input.lower()=='mpi':
      print('cd ~')
```

```
        os.chdir(os.path.expanduser("~"))
        print('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        os.system('wget https://download.open-mpi.org/release/open-mpi/v3.1/openmpi-3.1.3.tar.gz')
        print('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('tar -xzvf openmpi-3.1.3.tar.gz')
        os.system('mv -r ')
        print('cd openmpi-3.1.3')
        os.system('cd openmpi-3.1.3')
        os.chdir('openmpi-3.1.3')
        print('pwd')
        os.system('pwd')
        print('./configure --prefix=/usr/local/')
        os.system('./configure --prefix=/usr/local/')
        print('./configure --prefix=/usr/local/openmpi-3.1.3/')
        os.system('./configure --prefix=/usr/local/openmpi-3.1.3/')
        print('sudo make all install')
        os.system('sudo make all install')
        print('After make install is completed, mpirun or orterun executable should be at /usr/local/bin/.')
        print('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        os.system('echo "export PATH=$PATH:/usr/local/bin" >> $HOME/.bashrc')
        print('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" >
$HOME/.bashrc')
        os.system('echo "export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib" >
$HOME/.bashrc')
        print('export PATH=$PATH:/usr/local/bin')
        os.system('export PATH=$PATH:/usr/local/bin')
        print('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        os.system('export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/local/lib')
        print('source $HOME/.bashrc')
        os.system('source $HOME/.bashrc')
        print('mpirun --version')
        os.system('mpirun --version')
def parenthesis_check(file_name):
    string=''
    p=os.path.dirname(file_name)
    file=open(file_name,'r')
    file2=open(file_name,'r')
    while 1:
        char=file.read(1)
        if not char:
            break
        if char=='{' or char=='}':
            string+=char
    result=check(string)
    if result==1:
        lines = file2.readlines()
        lines = lines[:-1]
        for elem in reversed(lines):
            if '}\n' not in elem:
                lines = lines[:-1]
            else:
                break
        with open(p+'/a.cu.hip','w') as fp:
            for item in lines:
                fp.write(item)
        file.close()
        file2.close()
        os.remove(file_name)
        os.rename(p+'/a.cu.hip', file_name)
def parenthesis_check_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu.hip'):  ##or elem.endswith('.cpp')
            parenthesis_check(elem)
open_list = ["{"]
close_list = ["}"]
def check(myStr):
    stack = []
    for i in myStr:
        if i in open_list:
            stack.append(i)
        elif i in close_list:
            pos = close_list.index(i)
            if ((len(stack) > 0) and
                (open_list[pos] == stack[len(stack)-1])):
                stack.pop()
            else:
                return 1
    if len(stack) == 0:
        return 0
    else:
        return 1
def ftale(x):
    generate(x)
    apply_patches_individually(x)
    compilation_1(x)
    #compilation_2(x)
    runsample(x)
def generate_all(y):
    global user_platform
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
                #generate(elem)
                with open('final_ignored_samples.txt','r') as f:
                    if elem in f.read():
                        print("Ignoring this sample "+elem)
                    else:
                        generate(elem)
    apply_patches()
    #find . -type f -name '*.cu.hip' -print -delete
    #print("Do you also want to generate files of extension cu.cpp for compilation on Nvidia devices?")
    #user_input=input()
    #if user_input.lower() == 'yes' or user_input.lower() == 'y':
    if user_platform.lower()=='nvidia' :
        for elem in listOfFiles:
            if elem.endswith('.cu'):
                with open('final_ignored_samples.txt','r') as f:
                    if elem in f.read():
                        print("Ignoring this sample "+elem)
                    else:
                        elem2=elem+'.cpp'
                        if os.path.exists(elem2)==False:
                            print('Writing to '+elem2)
                            with open(elem+'.hip','r') as f1, open(elem2,'a') as f2:
                                for line in f1:
                                    f2.write(line)
def compilation_1_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    compilation_1(elem)
def compilation_2_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    compilation_2(elem)
def runsample_all(y):
    y=y.replace('"', '')
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
```

```
            with open('final_ignored_samples.txt','r') as f:
                if elem in f.read():
                    print("Ignoring this sample "+elem)
                else:
                    runsample(elem)
    os.chdir('src/')
    print("In your src folder:")
    print("Number of converted samples:")
    os.system('find . -name "*.cu.hip" | wc -l')
    print("Number of executables .out / .o:")
    os.system('find . -name "*.out" | wc -l')
    os.system('find . -name "*.o" | wc -l')
    print("Number of Ignored Samples:")
    os.system('cat ../final_ignored_samples.txt | wc -l')
    os.chdir('../src-original')
    print("In src-original folder:")
    print("Number of converted samples:")
    os.system('find . -name "*.cu.hip" | wc -l')
    print("Number of executables .out / .o:")
    os.system('find . -name "*.out" | wc -l')
    os.system('find . -name "*.o" | wc -l')
    print("Number of Ignored Samples:")
    os.system('cat final_ignored_samples.txt | wc l')
def generate(x):
    x=x.replace('"', '')
    p=os.path.dirname(x)
    q=os.path.basename(x)
    p=p.replace("\\","/")
    os.system("cd "+p)
    command="hipify-perl "+x+" > "+x+".hip"
    print(command)
    os.system(command)
    prepend_line(x+".hip",'#include "HIPCHECK.h"\n')
    prepend_line(x+".hip",'#include "rocprofiler.h"\n')
    textToSearch="checkCudaErrors"
    textToReplace="HIPCHECK"
    fileToSearch=p+"/"+q+".hip"
    textToSearch1="#include <helper_cuda.h>\n"
    textToReplace1='#include "helper_cuda_hipified.h"\n'
    textToSearch2="#include <helper_functions.h>\n"
    textToReplace2='#include "helper_functions.h"\n'
    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch,textToReplace))
    tempFile.close()

    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch1,textToReplace1))
    tempFile.close()
    tempFile=open(fileToSearch,'r+')
    for line in fileinput.input(fileToSearch):
        tempFile.write(line.replace(textToSearch2,textToReplace2))
    tempFile.close()
    parenthesis_check(x+".hip")
def apply_patches():
    y="src/patches"
    listOfFiles=getListOfFiles(y)
    for elem in listOfFiles:
        if elem.endswith('.patch'):
            command='git apply --reject --whitespace=fix '+elem
            print(command)
            os.system(command)
            os.system('find . -name "*.rej" -type f -delete')

def apply_patches_individually(x):
    patch_path='src/patches'
    search_path=x+'.hip'
    patch_files=[]
    dir=os.listdir(patch_path)
    for fname in dir:
        if os.path.isfile(patch_path+os.sep+fname):
            #f=open(patch_path+os.sep+fname,'r')
            with open(patch_path+os.sep+fname,encoding="utf8",errors='ignore') as f:
            #with open(patch_path+os.sep+fname) as f:
                contents = f.read()
                if search_path in contents:
                    #print('found path in patch file '+fname)
                    #lines = [x.decode('utf8').strip() for x in f.readlines()]
                    patch_files.append(fname)
                '''
                else:
                    print('Not found')
                '''
                f.close()
    for patch in patch_files:
        command='git apply --reject --whitespace=fix '+patch_path+'/'+patch
        print(command)
        os.system(command)
        os.system('find . -name "*.rej" -type f -delete')

def compilation_1(x):
    global cuda_path
    global user_platform
    cpp=[]
    print(user_platform)
    x=x.replace('"', '')
    p=os.path.dirname(x)
    p=p.replace("\\","/")
    for file in os.listdir(p):
        if file.endswith(".out") or file.endswith(".o"):
            os.remove(os.path.join(p,file))
    try:
        for file in os.listdir(p):
            if os.path.getsize(os.path.join(p,file))==0 and file in os.listdir(p.replace("src/","src-original/")):
                x_original=x.replace("src/","src-original/")
                alternate_file=x_original
                os.remove(os.path.join(p,file))
                os.rename(alternate_file,os.path.join(p,file))
    except FileNotFoundError as e:
        print(f'Error:{e}.Skipping replacement of empty files.')
    if user_platform.lower()=='nvidia':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.cpp"):
                cpp.append(file)
    elif user_platform.lower()=='amd':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.hip"):
                cpp.append(file)
    cpp = [p+'/'+y for y in cpp]
    file4=open('multithreaded_samples.txt', 'r')
    threaded_samples=file4.read()
    #print(threaded_samples)
    if x in threaded_samples:
        command='hipcc -I /opt/rocm/include -fopenmp -fgpu-rdc -I src/samples/Common -I '+cuda_path+'
'+' '.join(cpp)+' -lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out '
    else:
        command='hipcc -I /opt/rocm/include -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -
lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out'
    file4.close()
    print(command)
    os.system(command)
def compilation_2(x):
    global cuda_path
    global user_platform
    cpp=[]
    print(user_platform)
    x=x.replace('"', '')
    p=os.path.dirname(x)
    p=p.replace("\\","/")
    if user_platform.lower()=='nvidia':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.cpp"):
                cpp.append(file)
    elif user_platform.lower()=='amd':
        for file in os.listdir(p):
            if file.endswith("_hipified.cpp") or file.endswith(".cu.hip"):
                cpp.append(file)
    cpp = [p+'/'+y for y in cpp]
    file4=open('multithreaded_samples.txt', 'r')
    threaded_samples=file4.read()
    #print(threaded_samples)
    if x in threaded_samples:
        command='hipcc -I /opt/rocm/include -use-staticlib -fopenmp -fgpu-rdc -I src/samples/Common -I
'+cuda_path+' '+' '.join(cpp)+' -lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out'
    else:
        command='hipcc -I /opt/rocm/include -use-staticlib -I src/samples/Common -I '+cuda_path+' '+'
'.join(cpp)+' -lamdhip64 -o '+p+'/'+os.path.basename(os.path.dirname(x))+'.out'
    file4.close()
    print(command)
    os.system(command)

def runsample(x):
    print('Processing Sample:'+x)
    command='./'+os.path.dirname(x)+'/'+os.path.basename(os.path.dirname(x))+'.out'
    print(command)
    os.system(command)
def fall(y):
    generate_all(y)
    compilation_1_all(y)
    runsample_all(y)
def rem(z):
    print("This script automates sample exclusion.Please backup any paths provided by you to avoid loss or
overwriting.")
    input("Press Enter to continue...")
    a=open("samples_to_be_ignored.txt","w")
    a.truncate(0)
    b=open("final_ignored_samples.txt","w")
    b.close()
    z=z.replace("'",")
    ignore_list = ['<GL/','<screen/screen.h>', '<drm.h>','"FDTD3dGPU.h"','<d3d12.h>',
    ' <GLES3/gl31.h>','<EGL/egl.h>','<GLFW/glfw3.h>','"cudla.h"','#include <d312.h>',
    ' #include <GLES3/gl31.h>','#include <windows.h>','#include "nvmedia_image_nvscibuf.h"',
    '#include "graphics_interface.c"','#include <DirectXMath.h>','#include "cuda_gl_interop.h"',
    '#include "cudaEGL.h"','#include "cudaEGL.h"','"cuda_gl_interop.h"','#include "cuda_runtime.h"',
    '#include "cudla.h"','#include "nvscisync.h"','#include "FDTD3d.h"','#include "windows.h"',
    '#include "builtin_types.h"','#include "hipfft.h"','#include "screen/screen.h"','#include "Windows.h"',
    '#include "cuda_d3d9_interop.h"','#include "drm.h"','#include "cuda_runtime_api.h"','#include
"GLFW/glfw3.h"',
    '#include "cuda/barrier"','#include "cuda_runtime.h"','#include "cooperative_groups/reduce.h"',
    '#include "cuda_bf16.h"','#include "mma.h"','#include "cuda/pipeline"','"builtin_types.h"']
```

```
listofFiles=getListOfFiles(z)
for elem in listofFiles:
    if elem.endswith('.cu'):
        with open(elem) as f:
            for line in f:
                if any(word in line for word in ignore_list):
                    #a.write(elem+"\n"+line)
                    a.write(elem+"\n")
    elif elem.endswith('_hipified.cpp'):
        with open(elem) as f:
            for line in f:
                if any(word in line for word in ignore_list):
                    for file in os.listdir(os.path.dirname(elem)):
                        if file.endswith('.cu'):
                            a.write(os.path.dirname(elem)+'/'+file+"\n")
a.close()
with open('samples_to_be_ignored.txt') as fp:
    data1 = fp.read()
with open('custom_samples_path.txt') as fp:
    data2 = fp.read()
data1 += data2
with open ('final_ignored_samples.txt', 'a') as fp:
    fp.write(data1)
uniqlines = set(open('final_ignored_samples.txt').readlines())
bar = open('final_ignored_samples.txt', 'w')
bar.writelines(uniqlines)
bar.close()
fin = open('final_ignored_samples.txt', "rt")
data = fin.read()
data = data.replace('\\', '/')
fin.close()
fin = open("final_ignored_samples.txt", "wt")
fin.write(data)
fin.close()
lines_seen = set() # holds lines already seen
outfile = open('final_ignored_samples1.txt', "w")
for line in open('final_ignored_samples.txt', "r"):
    #outfile.writelines(sorted(lines_seen))
    if line not in lines_seen:
        outfile.write(line)
        lines_seen.add(line)
outfile.close()
os.remove('final_ignored_samples.txt')
if platform=="linux" or platform == "linux2":
    os.system('sort final_ignored_samples1.txt > final_ignored_samples.txt')
    os.remove('final_ignored_samples1.txt')
else:
    sorting('final_ignored_samples1.txt')
    os.remove('samples_to_be_ignored.txt')
    paths = []
    for root, dirs, files in os.walk(z):
        for file in files:
            if file.endswith('.cu'):
                path = os.path.join(root, file)
                paths.append(path)
    output_file = "sample_list.txt"
    with open(output_file, "w") as f:
        for path in paths:
            path=path.replace("\\","/")
            f.write(path + "\n")
    file1="sample_list.txt"
    file2="final_ignored_samples.txt"
    with open(file1, "r") as f:
        content1 = f.readlines()
    with open(file2, "r") as f:
        content2 = f.readlines()
    result = [line for line in content1 if line not in content2]
    output_file = "working_samples.txt"
    with open(output_file, "w") as f:
        for line in result:
            f.write(line)
def nvidia_compilation():
    nvidia_samples_dir='src/samples/Samples'
    global cuda_path
    listOfFiles=getListOfFiles(nvidia_samples_dir)
    for elem in listOfFiles:
        if elem.endswith('.cu'):  ##or elem.endswith('.cpp')
            cpp=[]
            elem=elem.replace('"', '')
            p=os.path.dirname(elem)
            p=p.replace("\\","/")
            for file in os.listdir(p):
                    if (file.endswith(".cpp") or file.endswith(".cu")) and not (file.endswith(".cu.cpp") or
file.endswith("_hipified.cpp")):
                        cpp.append(file)
            cpp = [p+'/'+y for y in cpp]
            file4=open('multithreaded_samples.txt', 'r')
            threaded_samples=file4.read()
            #print(threaded_samples)
            if elem in threaded_samples:
                command='nvcc -fopenmp -fgpu-rdc -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -o
'+p+'/a.out'
            else:
                command='nvcc -I src/samples/Common -I '+cuda_path+' '+' '.join(cpp)+' -o '+p+'/a.out'
            file4.close()
            print(command)
            os.system(command)
            print('Processing Sample:'+elem)
            command='./'+os.path.dirname(elem)+'/'+'a.out'
            print(command)
            os.system(command)
parser=argparse.ArgumentParser(description ='HIPIFY Cuda Samples.Please avoid and ignore samples with graphical operations')
#group = parser.add_mutually_exclusive_group()
parser.add_argument("-a", "--all", help='To run hipify-perl for all sample:python testhipify.py --all "[PATH TO SAMPLE FOLDER]"')
parser.add_argument("-b", "--generate", help='Generate .hip files')
parser.add_argument("-c", "--compile1", help='Compile .hip files')
parser.add_argument("-d", "--compile2", help='Compile .hip files with static libraries')
parser.add_argument("-e", "--execute", help='Execute .out files')
parser.add_argument("-f", "--generate_all", help='Generate all .hip files')
parser.add_argument("-g", "--compile1_all", help='Compile all .hip files')
parser.add_argument("-i", "--compile2_all", help='Compile all .hip files with static libraries')
parser.add_argument("-j", "--execute_all", help='Execute all .out files')
parser.add_argument("-k", "--parenthesis_check", help='Remove last parts from cu.hip files which are out of bounds.')
parser.add_argument("-l", "--parenthesis_check_all", help='Remove all last parts from cu.hip files which are out of bounds.')
parser.add_argument("-n", "--nvidia_compile", help='Compile and execute via nvcc.',action='store_true')
parser.add_argument("-p", "--patch", help='Apply all patches in src/patches',action='store_true')
parser.add_argument("-t", "--tale", help='To run hipify-perl for single sample:python testhipify.py -t "[PATH TO SAMPLE]"')
parser.add_argument("-x", "--remove", help='Remove any sample relating to graphical operations e.g.DirectX,Vulcan,OpenGL,OpenCL and so on.')
parser.add_argument("-s", "--setup1", help='Configure dependencies automatically.',action='store_true')
parser.add_argument("-v", "--setup2", help='Configure dependencies manually.',action='store_true')
parser.add_argument("-u", "--new_samples", help='Download latest samples from Repository.',action='store_true')
args=parser.parse_args()
if args.tale:
    x=args.tale
    ##print(x)
    ftale(x)
if args.all:
    y=args.all
    fall(y)
if args.remove:
    z=args.remove
    rem(z)
if args.generate:
    a=args.generate
    generate(a)
if args.patch:
    apply_patches()
```

```
if args.nvidia_compile:
    nvidia_compilation()
if args.compile1:
    b=args.compile1
    compilation_1(b)
if args.compile2:
    c=args.compile2
    compilation_2(c)
if args.execute:
    d=args.execute
    runsample(d)
if args.generate_all:
    a=args.generate_all
    generate_all(a)
if args.compile1_all:
    b=args.compile1_all
    compilation_1_all(b)
if args.compile2_all:
    c=args.compile2_all
    compilation_2_all(c)
if args.execute_all:
    d=args.execute_all
    runsample_all(d)
if args.parenthesis_check:
    e=args.parenthesis_check
    parenthesis_check(e)
if args.parenthesis_check_all:
    f=args.parenthesis_check_all
    parenthesis_check_all(f)
if args.setup1:
    setup1()
if args.setup2:
    setup2()
if args.new_samples:
    new_samples()
```

5.5 REFERENCES

Links:

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