Overview Document

"Bringing auto-tuning to HIP: Analysis of tuning impact and difficulty on AMD and Nvidia GPUs" by Milo Lurati, Stijn Heldens, Alessio Sclocco and Ben van Werkhoven

Getting started Guide

This artifact contains all the scripts to reproduce the analyses and plots presented in the paper based on the collected data. For completeness, all scripts that are used for data collection are also included. However, the data collection requires specialized hardware and multiple weeks of execution. Hence, the artifact focuses on reproducing the analyses and plots presented in the paper based on the already collected data.

The collected data is stored in so called 'cache files' that are used and produced by Kernel Tuner. The scripts included in this artifact will process the cache files for further analysis and to produce the plots included in the paper.

The scripts require a working Python installation of Python version 3.8, 3.9, 3.10 or 3.11, and the ability to install new packages using the 'pip' package manager. If you do not have a matching Python installed we recommend to install these locally using Miniconda. On Linux, one can use the following commands:

```
wget https://repo.anaconda.com/miniconda/Miniconda3-py311_24.3.0-0-Linux-x86_64.sh
bash Miniconda3-py311_24.3.0-0-Linux-x86_64.sh
```

For the full instructions on how to install Python using Miniconda, please see:
https://docs.anaconda.com/free/miniconda/#quick-command-line-install

After downloading the artifact's zip file, please unzip it and 'cd' into the main directory.

The artifact requires a new dependencies, which can be installed using:

```
pip install -r requirements.txt
```

Now you are ready to reproduce the analyses and plots presented in the paper with one command, you can do so with the script run_experiments.py:

```
python run_experiments.py
```

Step-by-Step Instructions

This section describes how to re-run all the experiments based on the already collected data which is part of this artifact. In the Appendix A of this document, we also include instructions on how to re-run post processing as well as the full data collection, these steps take much longer and require special hardware, and are only included here for completeness.

The script run_experiments.py will generate all plots and perform all data analyses needed to reproduce the results presented in the paper. All plots are saved in the directory 'plots', and all data is printed to the terminal.

If you prefer running the different experiments separately, you can follow the steps described in this section.

As described in the paper, we analyze the data collected using three main types of analyses:
1) We analyze the distribution of the auto-tuning search space to find the performance impact of using auto-tuning that we call tuning impact.

2) We analyze tuning difficulty using the proportion of centrality metric, which is calculated on the Fitness Flow Graph (FFG).

3) We analyze the performance portability of the auto-tuned kernels in case the optimal configuration on one device is used in a different setting.

Running these analyses has no special hardware or software requirements other than a Linux machine with a Python environment and the dependencies installed as explained in the Getting Started Section of this document.

Step 1a) Plot violins and calculate search space statistical values

To plot the violins and calculate the statistical values, please run:

```
python violins.py <kernel name>
```

Give kernel name as argument (convolution, hotspot, dedisp, gemm).

Step 1b) Calculate Top Configurations

When discussing the tuning results in the paper, we also include a table in the paper with the top 5 best performing configurations per kernel for each GPU. To reproduce these tables, please run:

```
python top_configurations.py <kernel name>
```

Give kernel name as argument (convolution, hotspot, dedisp, gemm).

Step 2) Plot proportion of centrality metric

To reproduce the plots for the tuning difficulty analysis, using the proportion of centrality metric, please run:

python plot_centralities.py

This script plots the result of the analysis of the Fitness Flow Graphs (FFGs) using the PageRank centrality metric. To also reproduce the FFGs and centralities, please see Optional Step 1.

Step 3) Plot Performance Portability

The command to reproduce the plots related to our analysis of the performance portability of kernels across AMD and Nvidia GPUs is:

python performance_portability.py

Appendix A

Optional step 1:

To also plot the Fitness Flow Graphs (FFGs) and rerun the centrality analysis using run_experiments.py, you will have to uncomment two lines that have been commented out in

run_experiments.py. The lines are line #10 and #11 in run_experiments.py. Line 10 is responsible for preprocessing the Kernel Tuner cache file into a cache file format that can be used with the Bloopy package. Line 11 performs further processing and uses Bloopy to reproduce the FFGs. Note that this takes hours of computation and requires a huge amount of RAM.

To reproduce the FFGs, run:

python compute_and_analyze_FFGs.py

By default, the script creates the FFG and computes the PageRank centralities (and saves them). By uncommenting line 198, the script will also draw the graph using networkX and save it as PDF. NOTE: Plotting FFGs is very expensive and may take a lot of RAM and time to plot.

Optional step 2:

If you want to reproduce data collection, that is the cache files produced by Kernel Tuner, this requires running Kernel Tuner to exhaustively explore the entire search space for all the tunable kernels. This step requires the following devices: AMD Radeon PRO W6600, AMD Instinct MI250X, NVIDIA RTX A4000, NVIDIA A100-PCIE-40GB. Note that running these scripts on the GPUs will take several days at least and in some cases may take close to two weeks.

For the experiments that we have performed for this paper, the system containing the W6600, A4000, and A100 used Rocky-8 Linux 4.18.0 with ROCM 6.0.2 with AMD clang 17.0.0, and CUDA 12.2 with GCC 9.4.0. For access to the MI250X GPU, we used the LUMI supercomputer, which uses SUSE Linux 5.14.21, ROCM 5.2.3 with AMD clang 14.0.0.

In terms of software you will need to install HIP for the AMD GPUs and CUDA for the Nvidia GPUs. You will also need to install Kernel Tuner. To install HIP, please consult the corresponding AMD resources: https://rocm.docs.amd.com/en/latest/. To install CUDA, please consult the documentation by Nvidia: https://docs.nvidia.com/cuda/cuda-installation-guide-linux/index.html

For AMD GPUs, once HIP is installed, proceed to install Kernel Tuner:

```
pip install kernel_tuner[hip]
```

For Nvidia, once CUDA is installed, please install Kernel Tuner using:

```
pip install kernel_tuner[cuda]
```

You can now run each Python script of the corresponding kernel. You can find these in the directory cache_scripts. For example, you can run:

```
python cache_scripts/convolution/convolution.py
```

NOTE: Remember that this process requires the correct GPU will take DAYS to execute!