GEMM User Tuning for AI Workloads

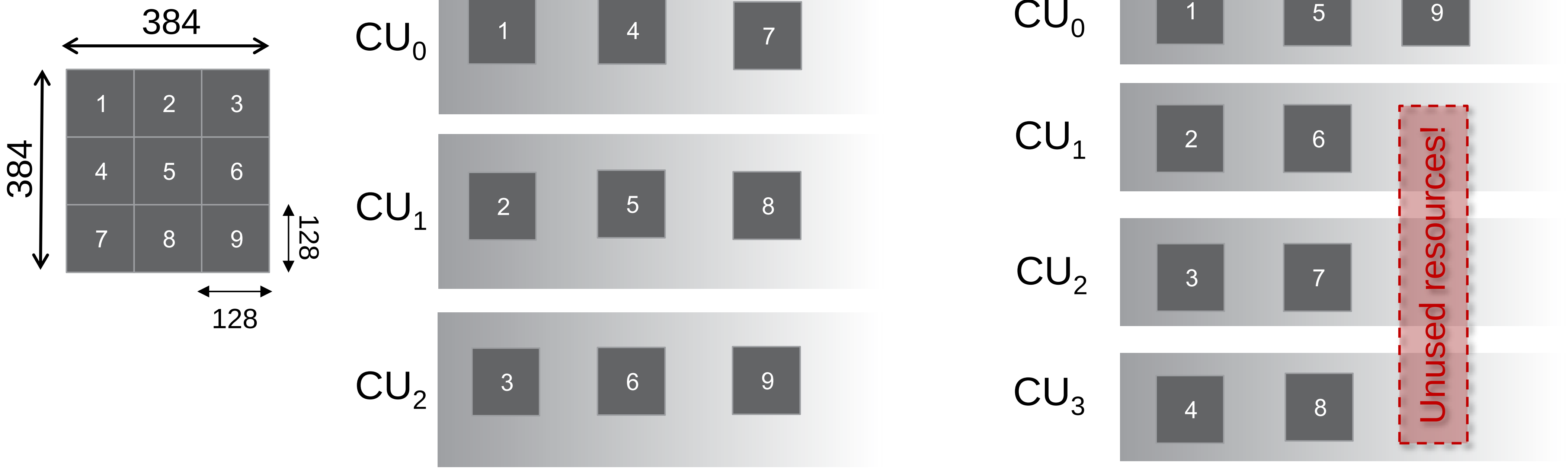
OLCF AI Training Series Part 3:
2025 Best Practices for AI on Frontier
March 28th 2025

Presenter: Alessandro Fanfarillo

AMD
together we advance_

Why do we need to tune for GEMMs?

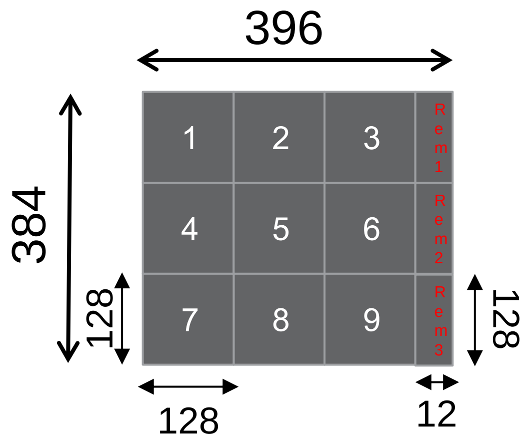
- GPU exploit parallelism to get high performance. Current approach is data-centric
- The output matrix is divided into tiles that get assigned to a workgroup. Each workgroup is assigned to a Compute Unit
- If the number of tiles are not a close multiple of the number of CUs, the utilization will be low
- Other parameters also influence the efficiency of the memory subsystem



AMD
together we advance_

Why do we need to tune for GEMMs?

- Using a tile size that is not a perfect multiple of the number of rows/columns causes underutilization
- GEMM libraries usually have a fix set of statically defined kernels and an algorithm selects the “optimal” kernel
- The selection algorithm is not perfect and it may pick a suboptimal kernel for a given GEMM
- Parallelization does not usually happen over k, the inner dimension
- Having a large K but small M and N does not (usually) help performance
- The situation is even worse with convolutions, involving different algorithms that perform better/worse based on the sizes



What can go wrong with GEMM performance?

1. The solution selection algorithm picks a suboptimal kernel for a particular size

   Solution: use rocblas-gemm-tune, `rocblas_gemm_ex_get_solutions_by_type()` API, TunableOp, hipblaslt-tuning, MIOpen tuning, etc.

2. An architecture (MI200) or a particular GEMM has not been optimized (using suboptimal kernels)

   Solution: Ask OLCF/COE to tune the library for that particular case

3. Math library is not able to generate optimized kernels for a particular size (small/large, skinny matrices)

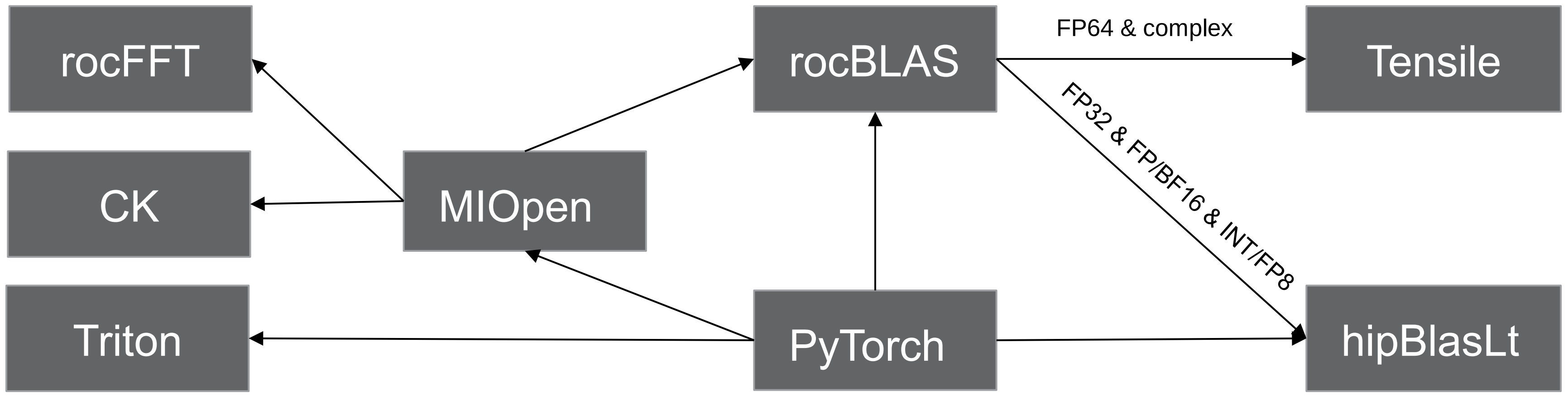
   Solution: Library developers need to be involved

As users, you can only address #1

Math libraries involved in AI workloads

The majority of operations needed by a deep learning model are GEMM-like

The overall model performance depends on how efficiently these operations are implemented



AMD
together we advance_

TunableOp in PyTorch

TunableOp is a PyTorch feature designed to optimize the GEMMs performance, by selecting the most efficient implementation available

Tuning is performed on a particular user workload. Results are stored in a csv file

```
PYTORCH_TUNABLEOP_ENABLED=1 python my_model.py
```

This will generate a file named `tunableop_results0.csv` in the same directory. The file contains the mapping between GEMM sizes and best performing kernel available in hipBlasLt

It is possible to build on top of previous tuning files

Once the tuning is done, the following command will use the tuning information:

```
PYTORCH_TUNABLEOP_ENABLED=1 PYTORCH_TUNABLEOP_FILENAME=tunableop_result0.csv python my_model.py
```

Depending on the model it is possible to achieve between 5% to 20% performance improvement

Problem: it is based on files, what happens if you run your model over 4K+ nodes?

Introduction to MIOpen

MIOpen is the ROCm™ library in charge of convolutions, normalizations, pooling, recurrent layers

It may or may not rely on other libraries to perform these operations

It selects the best algorithm/kernel size/library to perform a certain operation

We will only focus on convolutions, this is a list of solvers:

1. FFT-based
2. Winograd
3. GEMMs Convolution
4. Implicit GEMM Convolution
5. Direct Convolution

Tuning MIOpen

MIOpen uses the following to decide upon the best solver to be used for a requested convolution:

- User DB: stores the results of previous tunings and by default is found in "~/.config/miopen", although this can be changed
- System DB: part of the MIOpen install files and has specific shapes that the MIOpen team has saved as tuning choices
- Heuristics: model trained on the shapes and solvers so that it picks the best solver and parameters. The goal is that this is within 90% of what can be achieved by specific tuning for the given shape

Two possibilities for tuning:

1. Incremental tuning: tuning when MIOpen encounters a new shape that does not exist in the user or system DBs. Tuning is conducted once for each missing shape and the results are saved to the user DB. Use this mode to add missing tuning information to the user database
2. Exhaustive tuning: two possible methods:
   1. Method 1: similar to incremental but ignoring system DB. To be preferred
   2. Method 2: full tuning unrelated from workload

Incremental Tuning - MIOpen

This mode adds missing tuning information to the user database

When in this mode, your application’s first run may take substantially longer than expected

```
export MIOPEN_FIND_MODE=3

export MIOPEN_FIND_ENFORCE=3

export MIOPEN_USER_DB_PATH="/user/specified/directory“
```

FIND_MODE=3 checks User DB for an entry. If there is a hit, it uses that entry. If there is a miss, it uses the existing find machinery

FIND_ENFORCE=3 check if DB already contains optimized values, in that case tuning is not performed

Exhaustive Tuning Method 1 - MIOpen

Exhaustive tuning can be executed similarly to incremental tuning by ignoring the system DB

Redirecting the system DB path causes MIOpen to miss system entries and heuristic models, forcing a user DB entry to be generated

Each unique configuration is tuned exactly once, with results aggregated in the user DB files

```
export MIOPEN_FIND_MODE=3

export MIOPEN_FIND_ENFORCE=3

export MIOPEN_USER_DB_PATH="/user/specified/directory"

export MIOPEN_SYSTEM_DB_PATH="$MIOPEN_USER_DB_PATH“
```

The default location for the MIOpen user database is "$HOME/.config/miopen"

Extract GEMM sizes from PyTorch

Understanding what GEMM sizes are being executed can be helpful to communicate with OLCF/AMD staff on potential inefficiencies in the libraries

Modern PyTorch versions will almost always invoke hipBlasLt directly but sometimes it may invoke a mix of rocBLAS and hipBlasLt

For this reason, the following environment variables need to be set to make sure that logs from both libraries are being saved:

```
export ROCBLAS_LAYER=2

export ROCBLAS_LOG_PATH=rocblas-bench.log

export HIPBLASLT_LOG_MASK=32

export HIPBLASLT_LOG_FILE=hipblaslt-bench.log
```

For MIOpen:

```
export MIOPEN_ENABLE_LOGGING_CMD=1
```

Conclusions

- GEMMs are fundamental to AI workloads, serving as the backbone for neural network computations. Optimizing these operations is crucial for achieving high performance and efficiency in AI applications
- TunableOp allows for dynamic tuning of GEMM operations within PyTorch. Developers can achieve significant performance improvements without extensive code modifications
- MIOpen relies on heuristics and system databases to select the best algorithm and kernel to perform convolutions
- MIOpen perform operations other than convolutions (e.g., normalizations) that cannot be tuned
- Implementing these tuning strategies can lead to significant performance gains in AI workloads on AMD™ GPUs

Disclaimers and Trademark

AMD
together we advance_

AMD