FP16 Tests Experiment Summary

Lixun Zhang

03 August 2022

Contents

List of Tables

1 Experiment Setup

1.1 Sources of fp16 configs

All fp16 configs are extracted from `/mlir/test/mlir-miopen-driver` and each config is moved into an individual file named `$filename_CHECK_$prefix.mlir`, where `$filename` is the original filename that the config resieds in and `$prefix` is the prefix used by `FileCheck` for the config.

Here is a list of all files that contain at least one f16 config:

- `auto_e2e/padding_kernel_gemmK.mlir` (3)
- `auto_e2e/padding_kernel_gemmN.mlir` (2)
- `auto_e2e/conv2d_host_validation_f16_fwd.mlir` (52, 1 is FIXME)
- `auto_e2e/conv2d_host_validation_f16_bwd.mlir` (18)
- `auto_e2e/padding_kernel_all_fwd.mlir` (2)
- `auto_e2e/padding_kernel_all_wrw.mlir` (2)
- `auto_e2e/conv2d_host_validation_f32_fwd.mlir` (1, f32 file has a f16 test?)
- `e2e_for_pr/Resnet50_validation_nchw.mlir` (4)
- `e2e_for_pr/Resnet50_validation_nhwc.mlir` (4)

- `misc_e2e/conv2d_harness_cpu_verifier_kyxc_nhwc_nhwk_f16.mlir` (1)
- `misc_e2e/conv2d_host_validation_f16.mlir` (10, 38 are FIXME)
- `misc_e2e/conv2d_host_validation_random.mlir` (8)

The number in the parenthesis indicates the number of fp16 configs in the file. There are totally 107 fp16 configs (FIXME configs are not counted).

1.2 Settings of fp16 tests

When invoking `miopen-gen` on each of the config, the following options can be set to control the behavior of the test:

- `-rand 1` or `-rand 0`: choose the seed of the random number generator. 1 means a fixed seed and 0 means a seed that is a function of the time.
- `-rand_min min` and `-rand_max max`: choose the range of the random numbers.
- `-pv` or `-pv_with_gpu`: choose the validation function method.
- `-x2` or : choose whether xdlops is enabled for the GPU kernel.

1.3 Metrics

To verify the results of the GPU kernel, the verification function uses the following metrics to measure the difference between the results of the GPU kernel (`kern`) and the validation function (`val`).

1.3.1 Element wise difference

The element wise difference metric computes the maximum difference among all elements that can be written as $\text{diff} = \max_i(d(\text{val}_i - \text{kern}_i, \text{val}_i))$ When different diff functions (`d()`) are used, we can have the following element wise difference metrics.

- Absolute difference:
$$\text{maxAbsDiff} = \max_i(|\text{val}_i - \text{kern}_i|)$$
- Relative difference:
$$\text{maxRelDiff} = \max_i \left(\frac{|\text{val}_i - \text{kern}_i|}{|\text{val}_i|} \right), |\text{val}_i| > 0$$
- Relative difference (ignore small denominators):
$$\text{maxRelDiff}_{\text{old}} = \max_i \left(\frac{|\text{val}_i - \text{kern}_i|}{|\text{val}_i|} \right), |\text{val}_i| > 1 \times 10^{-3}$$
- Epsilon difference:
$$\text{maxEpsilonDiff} = \max_i \left(\frac{|\text{val}_i - \text{kern}_i|}{\epsilon_{|\text{val}_i|}} \right)$$
where $\epsilon_{|\text{val}_i|}$ is the precision of fp16 numbers around $|\text{val}_i|$. For more information, check the precision limitations section of Half-precision floating-point format[1].

For debugging purpose, the histograms of the above element wise differences can be printed for an individual test. We will show the histograms of different element wise metrics for the worst test on both MI200 and MI100 in this section.

1.3.2 Root Mean Square (RMS)

The root mean square difference between two data sets A and B is defined as

$$\frac{\sqrt{\sum_{i=1}^{N}(a_i - b_i)^2}}{\sqrt{N} \cdot \max(\max(|a_i|), \max(|b_i|))}$$

[1] https://en.wikipedia.org/wiki/Half-precision_floating-point_format

2 Experiment Results

The random numbers used to initialize the inputs are uniformly sampled between the range `[rand_min, rand_max]`, which are provided as options when invoking `miopen-gen`. Four ranges are chosen:

- `r0`: [-1, 1]
- `r1`: [-10, 10]
- `r4`: [1, 5]
- `r5`: [5, 10]

2.1 Effects of Input Number Magnitude

The following table shows the aggregated values among all fp16 tests for all metrics on MI200. The error with r1 ([-10, 10]) is larger than the values with r0 ([-1, 1]) except for the maxRelDiff and RMS metrics. This is because with r0, the denominator of the relative difference is much closer to zero than with r1. When small denominators are ignored, which leads to the maxRelDiff old metric, the error with r1 is larger than that with r0.

	r0	r1
maxEpsilonDiff ave	314.24	1,627.64
maxEpsilonDiff max	2,030.33	23,316.33
maxAbsDiff ave	0.15	3
maxAbsDiff max	5.67	26.67
maxRelDiff ave	4.42	2.79
maxRelDiff max	111.33	105.67
maxRelDiff old ave	0.02	0.1
maxRelDiff old max	0.1	2.2
RMS ave	2.12E-06	1.31E-06
RMS max	1.55E-05	6.57E-06

Observation: The larger the magnitude, the larger the error. (This is true because r0 and r1 are around 0)

2.2 Effects of Subnormal Numbers in Inputs

The following table shows the aggregated values for r0 ([-1, 1]), r4 ([1, 5]), and r5 ([5, 10]) on MI200. The errors are much larger with r0 than r4/r5 except for the maxAbsDiff and RMS metrics. The maxAbsDiff metric is expected to be larger with r4 and r5 since larger input numbers are used. The RMS metric is relatively stable with different ranges.

	r0	r4	r5
maxEpsilonDiff ave	314.24	0.73	0.72
maxEpsilonDiff max	2,030.33	1	1
maxAbsDiff ave	0.15	2.67	4.99
maxAbsDiff max	5.67	32	32
maxRelDiff ave	4.42	5.64E-04	4.69E-04
maxRelDiff max	111.33	9.80E-04	9.80E-04
maxRelDiff old ave	2.06E-02	5.64E-04	4.69E-04
maxRelDiff old max	9.50E-02	9.80E-04	9.80E-04
RMS ave	2.12E-06	5.34E-06	3.21E-06
RMS max	1.55E-05	2.47E-05	1.25E-05

Observation:

1. When subnormal numbers are not involved, much smaller errors are expected.
2. The RMS metric is insensitive of subnormal numbers.

2.3 Effects of validation methods

The following table shows the results for all metrics with different combinations of kernel and validation methods. (The table does not show results for nonxdlops+gpu because the error is always 0 under all metrics.) Note that gpu validation function does not enable xdlops.

The cpu validation function implements the convolution operation in a sequential manner and uses fp64 variable as the accumulator. The nonxdlops gpu kernel converts the convolution to gemm and implements the gemm without using the `mfma` instructions. The accumulator is fp32 ?. The xdlops gpu kernel converts the convolution to gemm and implements the gemm with `mfma` instructions. The fundamental hardware is the DOT4_F32_F16 unit inside the miSIMD Matrix Core, which performs a fused multiply and add of 4 pairs of elements and accumulates the result with FP90.

	xdlops+cpu	xdlops+gpu	nonxdlops+cpu
maxEpsilonDiff ave	352.98	818.53	285.98
maxEpsilonDiff max	3,959.25	9,306.00	5,746.25
maxAbsDiff ave	3.13	3.76	1.21
maxAbsDiff max	24.13	28.00	20.13
maxRelDiff ave	2.01	2.82	0.57
maxRelDiff max	68.5	81.50	12.75
maxRelDiff old ave	2.45E-02	4.96E-02	1.72E-02
maxRelDiff old max	3.20E-01	8.43E-01	5.60E-01
RMS ave	2.75E-06	4.57E-06	1.67E-06
RMS max	1.26E-05	2.03E-05	1.16E-05

Observation: xdlops+gpu has the largest error and nonxdlops+cpu has the smallest error.

Explanation:

1. CPU implementation and nonxdlops GPU kernel are close to each other because they have similar truncation behavior and computation order.
2. CPU implementation and xdlops GPU kernel have medium difference because they have similar truncation behavior but different computation order.
3. xdlops and nonxdlops GPU kernels have largest error because they have different truncation behaviors and computation order.

2.4 Difference between MI100 and MI200

The relevant hardware difference between MI100 and MI200 is that the `mfma` instructions on MI200 flush denormalized numbers to zero in both input and output. (Refer to table 25 of the MI200 Manual[2] for descriptions of the instructions.) The following table shows the result on MI100 and MI200 for all metrics with different random number ranges.

	MI200(r0)	MI100(r0)	MI200(r1)	MI100(r1)	MI200(r4)	MI100(r4)	MI200(r5)	MI100(r5)
maxEpsilonDiff ave	314.24	90.50	1,627.64	1,757.24	0.73	0.72	0.72	0.72
maxEpsilonDiff max	2,030.33	1,739.00	23,316.33	40,028.00	1.00	1.00	1.00	1.00
maxAbsDiff ave	0.15	0.15	3.00	2.95	2.67	2.67	4.99	4.99
maxAbsDiff max	5.67	5.67	26.67	26.67	32.00	32.00	32.00	32.00
maxRelDiff ave	4.42	1.70	2.79	2.68	5.64E-04	5.63E-04	4.69E-04	4.68E-04
maxRelDiff max	111.33	34.00	105.67	92.67	9.80E-04	9.80E-04	9.80E-04	9.80E-04

[2] https://developer.amd.com/wp-content/resources/CDNA2_Shader_ISA_18November2021.pdf

	MI200(r0)	MI100(r0)	MI200(r1)	MI100(r1)	MI200(r4)	MI100(r4)	MI200(r5)	MI100(r5)
maxRelDiff old ave	0.02	0.01	0.10	0.10	5.64E-04	5.63E-04	4.69E-04	4.68E-04
maxRelDiff old max	0.10	0.09	2.20	2.22	9.80E-04	9.80E-04	9.80E-04	9.80E-04
RMS ave	2.12E-06	1.80E-06	1.31E-06	1.31E-06	5.34E-06	5.33E-06	3.21E-06	3.21E-06
RMS max	1.55E-05	1.55E-05	6.57E-06	6.77E-06	2.47E-05	2.47E-05	1.25E-05	1.25E-05

Observations:

1. The results with r4 and r5 are almost the same on MI100 and MI200.
2. With r0, errors on MI100 is smaller than that on MI200.
3. With r1, errors on MI100 and MI200 are almost the same except for the maxEpsilonDiff metric, in which the error on MI200 is smaller.

2.5 The "Worst" Test on MI200

We focus on the worst test on MI200, i.e. `mlir-miopen-driver/auto-e2e/padding_kernel_gemmK.mlir` (`CHECK_RESNET50_F16_CONFIG1`), which is a forward convolution operation with a total of 205520896 elements in the output.

The following table shows the histogram of relDiff old among all elements in the output with r0. The max relDiff old is 0.055, which is generated from `val=0.00101471` and `kern=0.001070023`.

relDiff_old value interval	number of elements
0	205235692(99.861229%)
(0,1e-6)	0(0.00%)
[1e-6, 1e-5)	0(0.00%)
[1e-5, 1e-4)	0(0.00%)
[1e-4, 1e-3)	244810(0.119117%)
[1e-3, 1e-2)	15725(0.007651%)
[1e-2, 0.1)	925(0.000450%)
[0.1, 1)	0(0.00%)
>= 1	0(0.00%)
max	0.055
val	0.00101471
kern	0.001070023

The following table shows the histogram of epsilonDiff among all elements in the output with r0. The max value is 850, which is generated from `val=3.50475E-05` and `kern=8.57115E-05`.

epsilonDiff value interval	number of elements
0	205235692(99.861229%)
1	259106(0.126073%)
2	11473(0.005582%)
[3, 10]	12039(0.005858%)
[11, 100]	2331(0.001134%)
> 100	255(0.000124%)
max	850
val	3.50475E-05
kern	8.57115E-05

3 Exploration of a "Better" Validation Function

The following two tables show the effects of some improvements of the validation function on the histograms of the error.

- `accumulate 4` refers to the changes that the cpu validation function accumulates 4 multiplication results with fp64 and truncates the sum to fp32 before moving forward.
- `flush subnormals` refers to the changes that the cpu validation function flushes subnormal numbers ($< 2^{-14}$) in the input to zero.

relDiff old	original	accumulate 4	flush subnormals
0	801676(99.858000%)	802015(99.900226%)	802369(99.944321%)
(0,1e-6)	0(0.000000%)	0(0.000000%)	0(0.000000%)
[1e-6, 1e-5)	0(0.000000%)	0(0.000000%)	0(0.000000%)
[1e-5, 1e-4)	0(0.000000%)	0(0.000000%)	0(0.000000%)
[1e-4, 1e-3)	970(0.120825%)	704(0.087691%)	364(0.045340%)
[1e-3, 1e-2)	60(0.007474%)	60(0.007474%)	0(0.000000%)
[1e-2, 0.1)	3(0.000374%)	3(0.000374%)	0(0.000000%)
[0.1, 1)	0(0.000000%)	0(0.000000%)	0(0.000000%)
>= 1	0(0.000000%)	0(0.000000%)	0(0.000000%)
cpuVal == 0	0(0.000000%)	0(0.000000%)	0(0.000000%)
max	0.027	0.028	0.00097
cpuVal	-0.001073837	-0.001074791	0.125244141
gpuVal	-0.001045227	-0.001045227	0.125366211

epsilonDiff	original	accumulate 4	flush subnormals
0	801676(99.858000%)	802015(99.900226%)	802369(99.944321%)
1	1047(0.130416%)	722(0.089933%)	413(0.051444%)
2	46(0.005730%)	40(0.004982%)	19(0.002367%)
[3, 10]	39(0.004858%)	32(0.003986%)	15(0.001868%)
[11, 100]	8(0.000996%)	7(0.000872%)	0(0.000000%)
> 100	0(0.000000%)	0(0.000000%)	0(0.000000%)
max	35	35	8
cpuVal	-0.001625061	-0.001625061	9.07E-05
gpuVal	-0.00165844	-0.00165844	9.02E-05

Observations:

1. `accumulate 4` slightly improves the results under both metrics.
2. `flush subnormals` greatly improves the results under both metrics.

4 Summary

From the experiment results, we can see that the errors for fp16 tests are caused by three sources:

1. Subnormal numbers flushed to zero
2. Computation order difference due to partition of gemm onto the grid
3. Truncation behavior difference during accumulation of intermediate results

The first cause can be solved by avoiding subnormal numbers in the computation. A good candidate is to use [1, 5] or [-5, -1] as the random number range to initialize the inputs. Another candidate is to use the RMS metric, which filters out differences between small numbers.

The computational order for each element in the output is determined by the partition mechanism used by the gemm algorithm. It is not fair, if not impossible, for the validation function to take into account the partition mechanism to keep the same computational order as the GPU kernel.

The truncation behavior is another thing that the validation function cannot emulate. To make things worse, the influence of the truncation behavior to the final result is deeply coupled with the computational order of the elements.

Fortunately, the last two causes can be filtered out with element wise metric with a random number range away from 0 or the RMS metric, which is naturally more robust than element wise metrics. Since the element wise metrics and the RMS metric have their pros and cons, the best verification function should apply both metrics and output flexible diagnostic messages for fp16 tests.

All the experiment results can be found at fp16_experiment_results[3].

5 Proposal of A New E2E Testing Mechanism

5.1 The Current Mechanism

The current E2E tests are in `/mlir/test/mlir-miopen-driver/` and follow a workflow as:

config → `miopen-gen` → `mlir-miopen-driver` → `mlir-rocm-runner` → `FileCheck`

- config: basic configuration of the convolution operations, which includes
  - direction: fwd, bwd_data, and bwd_weight
  - dimension of input, output, and filter
  - layout of input, output, and filter
  - number of channels and batches
  - paddings, dilations, and strides
- `miopen-gen`: generation of the convolution kernel and host harness code. It can also generate validation and verification functions as requested. The following information are further added to the config to create a complete test:
  - `-t`: data type of all elements
  - `-rand`: pattern of random numbers used for initialization
    - `-rand fixed`: a pattern of three hard-coded numbers is used to initialize the inputs.
    - `-rand N`: generate random numbers using `std::rand()` within range [-1, 1] for floats and [-5, 5] for integers
      - `N > 0`: random numbers are generated using `srand(N)`
      - `N = 0`: random numbers are generated using `srand(time(0))`
    - `-rand_type`: data type of the random numbers
  - `-x2`: enable xdlops in the GPU kernel
  - `-pv` or `-pv_with_gpu`: validation function. If either is specified, a verification function is generated to compare the results between the GPU kernel and the validation function. The metric used is the $\text{maxRelDiff}_{\text{old}}$ and the thresholds are hard-coded as
    - 0.25 for fp16 data type
    - 0.000001 (1e-6) for other data type
- `mlir-miopen-driver` + `mlir-rocm-runner`: JIT execute the generated code
- `FileCheck`: checks the output of the verification function: [0] is fail and [1] is pass

The current CI setup for E2E tests is as follows:

- In PR CI, tests in `e2e_for_pr` are enabled as fixed tests (`-rand fixed`) with GPU validation (`-pv_with_gpu`)
- In nightly CI, both fixed and random E2E tests are enabled
  - tests in `auto_e2e` are enabled as random tests (`-rand 1`) with CPU validation (`-pv`)
  - tests in `auto_e2e` and `misc_e2e` are enabled as fixed tests (`-rand fixed`) with GPU validation (`-pv_with_gpu`)

5.2 The Drawbacks of the Current Mechanism

1. The old validation/verification method

[3] https://amdcloud-my.sharepoint.com/:x:/g/personal/lixzhang_amd_com/EaERwVsy40RGksM0xS-RgKUBkw2wSfXNgrHIK8retNke3A

- The element wise metric $\text{maxRelDiff}_{\text{old}}$ is too sensitive to the absolute value of the element.
- The range of random numbers is hard-coded
- The hard-coded thresholds make it inflexible to use different threshold for different tests.
- The verification function is coupled with the data type of the inputs.

2. The CI testing mechanism
   - The inputs of fixed tests (`-rand fixed`) are too "good" to catch any bugs
   - The PR CI contains too few E2E tests.
   - Many E2E tests are organized according to when they were reported (MLIR tickets). This may lead to repetition of testing and makes it confusing to add more tests.

5.3 Proposals

1. Use RMS and element wise metrics together in the verification function.
   - The verification function computes [RMS, maxAbsDiff, maxRelDiff] between gpu kernel and the validation function.
   - The verification function compares the metrics with their corresponding thresholds and outputs 0 or 1 for each metric indicating whether the metric is below its threshold.
   - The verification function can also output more detailed information about the difference, such as the histogram of element wise metric, elements values corresponding to the max error.
2. Move the decision of how to do verification out of `miopen-gen` so that test writers can determine how to verify the results, such as random number patterns and ranges, verification metrics, and thresholds. This also enables the verification function to focus on comparing the results without worrying about numerical errors from different data types.
   - Make random number range and metric thresholds command line options for `miopen-gen`.
   - Each test (CHECK command) can determine which metric(s) to check and the corresponding threshold(s) for the metric(s).
3. Reorganize the current E2E tests