Towards a Better Verification function

Lixun Zhang

August 17, 2022

Outline

Verification Method in MLIR

Verification: compare outputs from GPU kernel (kern) and validation function (val)

Current verification method used by MLIR

$$\text{maxRelDiff}_{\text{old}} = \max_i\left(\frac{|\text{val}_i - \text{kern}_i|}{|\text{val}_i|}\right),\ |\text{val}_i| > 1 \times 10^{-3}$$

- Element wise metric
- Relative error between two values
- Ignore the error if the denominator is too small ($< 1 \times 10^{-3}$)
- Pass if the maximum relative difference among all elements is smaller than a tolerance δ_{rel}

Problem: $\delta_{rel} = 15\%$ for fp16 tests with random inputs ([-1, 1])

- MLIR#408, PR#694, PR#696
- δ_{rel} is too large to distinguish bugs from numerical errors
- For other data types, $\delta_{rel} = 1 \times 10^{-6}$

Verification Method in MIOpen

Current verification method used by MIOpen: RMS

- global metric
- Pass if RMS is smaller than a tolerance δ_{RMS}
- Adjust δ_{RMS} for different directions, algorithms, data types, and issues

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

		int32	int8	fp32	fp16	bf16
fwd bwd	other	1.50E-06	1.50E-06	1.50E-06	8.20E-03	6.56E-02
	igemm	1.50E-05	1.50E-05	1.50E-05	8.20E-02	0.656
wrw	other	3.00E-06	3.00E-06	3.00E-06	1.64E-02	0.13
	winograd	3.00E-06	3.00E-06	3.00E-05	1.64E-02	0.13
	igemm	3.00E-06	3.00E-06	3.00E-05 0.01 (issue2176)	8.20E-02	0.13

Pro/con of RMS as a measure of difference

- Insensitive to element wise differences

Element wise vs. RMS

Element wise metric

- compare each element
- pick the maximum difference
- sensitive to large differences for individual elements
- false alarm caused by numerical errors

Aggregated global metric

- compare each element
- aggregate differences from all elements
- insensitive to large differences for individual elements
- false negative and miss a real bug

We need a better verification function that can detect bugs and ignore numerical errors

- which metric to use? Element wise or RMS?
- how to choose δ?
- how to deal with special tests?

Other Element Wise Metrics

- Absolute difference $\mathrm{maxAbsDiff} = \max_i(|\mathrm{val}_i - \mathrm{kern}_i|)$
- Relative difference (ignore zero denominators)

$$\mathrm{maxRelDiff} = \max_i(\frac{|\mathrm{val}_i - \mathrm{kern}_i|}{|\mathrm{val}_i|}),\ \ |\mathrm{val}_i| > 0$$

- Relative difference w.r.t ϵ

$$\mathrm{maxEpsilonDiff} = \max_i(\frac{|\mathrm{val}_i - \mathrm{kern}_i|}{\epsilon_{|\mathrm{val}_i|}})$$

where $\epsilon_{|\mathrm{val}_i|}$ is the precision of fp16 numbers around val_i.

Experiment Setup

Sources of configs: auto_e2e/ e2e_for_pr/ misc_e2e/

	fixed	random	PR CI	Nightly CI
e2e_for_pr	pv_with_gpu			
auto_e2e	pv_with_gpu	pv		
misc_e2e	pv_with_gpu			

Hardware

- MI100 gfx908 ixt-rack-112
- MI200 gfx90a lockhart5

Settings of the test

- `-rand 1`: Generate random numbers as inputs
- `-rand_min` and `-rand_max`: choose the range of the random numbers
  - r0 [-1, 1], r1 [-10, 10], r4 [1, 5], r5 [5, 10]
- `-pv` and `-pv_with_gpu`: choose the validation method
- `-x2`: enable xdlops

Methodology: Effects of different settings to the error w.r.t all metrics

Flush-denorms-to-zero

xdlops/cpu	MI200			MI100		
	r0	r1	r4	r0	r1	r4
maxEpsilon ave	423.43	986.77	0.87	74.89	1,084.96	0.86
maxEpsilon max	2446	13389	1	2012	18788	1
maxAbsDiff ave	0.04	3.31	3.08	0.04	3.16	3.08
maxAbsDiff max	0.5	32	32	0.5	32	32
maxRelDiff ave	6.33	1.71	6.85E-04	1.63	1.53	6.82E-04
maxRelDiff max	230	44	9.80E-04	20	25	9.80E-04
maxRelDiff old ave	0.03	0.07	6.85E-04	4.30E-03	0.07	6.82E-04
maxRelDiff old max	0.077	1.2	9.80E-04	0.051	0.97	9.80E-04
RMS ave	2.08E-06	1.12E-06	4.79E-06	1.41E-06	1.11E-06	4.75E-06
RMS max	1.40E-05	3.80E-06	2.30E-05	1.40E-05	4.40E-06	2.30E-05

`mfma` instructions flush subnormal numbers ($< 2^{-14}$) to zero on MI200

r0 [-1, 1]

r1 [-10, 10]

r4 [1, 5]

- flush-denorms-to-zero behavior leads to
  - $E_{r1} > E_{r0}$ on both MI200 and MI100 (except for maxRelDiff and RMS)
  - $E_{MI200} > E_{MI100}$ with r0
  - $E_{MI200} \approx E_{MI100}$ with r4

Underflow and Overflow

nonxdlops/ cpu	MI200		
	r0	r4	r5
maxEpsilon ave	48.13	0.36	0.42
maxEpsilon max	1367	1	1
pass rate	68.22%	100.00%	99.07%
maxAbsDiff ave	0.012	1.322	2.755
maxAbsDiff max	0.5	32	32
maxRelDiff ave	0.75	2.84E-04	2.57E-04
maxRelDiff max	18	9.80E-04	9.80E-04
maxRelDiff old ave	2.31E-03	2.84E-04	2.57E-04
maxRelDiff old max	3.80E-02	9.80E-04	9.80E-04
RMS ave	6.20E-07	3.39E-06	2.08E-06
RMS max	3.50E-06	2.60E-05	1.40E-05

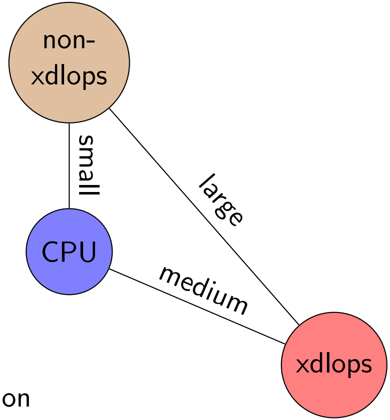
r0 [-1, 1] vs. r4 [1, 5]/r5 [5, 10]

A test passes if maxEpsilonDiff ≤ 1

- Errors with r0 is larger than that of r4/r5 $\Leftarrow$ Underflow
- Except for maxAbsDiff and RMS $\Leftarrow$ larger inputs with r4/r5
- All tests with r4 pass since the maximum maxEpsilonDiff is 1
- However, some tests with r5 fail even the maximum maxEpsilonDiff is 1 $\Leftarrow$ Overflow ($>$ 65504)

Validation Methods

kernel/ validation	MI200					
	r0			r4		
	nonxdlops/ cpu	xdlops/ cpu	xdlops/ nonxdlops	nonxdlops/ cpu	xdlops/ cpu	xdlops/ nonxdlops
maxEpsilon ave	48.13	423.43	471.15	0.36	0.87	0.95
maxEpsilon max	1367	2446	2278	1	1	1
maxAbsDiff ave	0.012	0.04	0.405	1.32	3.08	3.61
maxAbsDiff max	0.5	0.5	16	32	32	32
maxRelDiff ave	0.75	6.33	6.17	2.84E-04	6.85E-04	7.24E-04
maxRelDiff max	18	230	86	9.80E-04	9.80E-04	9.80E-04
maxRelDiff old ave	2.31E-03	2.63E-02	3.33E-02	2.84E-04	6.85E-04	7.24E-04
maxRelDiff old max	0.038	0.077	0.17	9.80E-04	9.80E-04	9.80E-04
RMS ave	6.20E-07	2.08E-06	3.67E-06	3.39E-06	4.79E-06	7.84E-06
RMS max	3.50E-06	1.40E-05	2.90E-05	2.60E-05	2.30E-05	2.50E-05



r0 [-1, 1] vs. r4 [1, 5]

- cpu: sequential implementation of the convolution operation
- non-xdlops: convert convolution to gemm and compute gemm without `mfma`
- xdlops: convert convolution to gemm and compute gemm with `mfma`

Exploration of a "Better" Validation Function

MI200 xdlops/cpu

`auto_e2e/padding_kernel_gemmK.mlir [CHECK_RESNET50_F16_CONFIG1]`

relDiff old	original	acc 4	flush+acc 4
0	801676(99.86%)	802015(99.90%)	802369(99.94%)
(0,1e-6)	0(0.0%)	0(0.0%)	0(0.0%)
[1e-6, 1e-5)	0(0.0%)	0(0.0%)	0(0.0%)
[1e-5, 1e-4)	0(0.0%)	0(0.0%)	0(0.0%)
[1e-4, 1e-3)	970(0.12%)	704(0.09%)	364(0.04%)
[1e-3, 1e-2)	60(0.007474%)	60(0.007474%)	0(0.0%)
[1e-2, 0.1)	3(0.000374%)	3(0.000374%)	0(0.0%)
[0.1, 1)	0(0.0%)	0(0.0%)	0(0.0%)
>= 1	0(0.0%)	0(0.0%)	0(0.0%)
val == 0	0(0.0%)	0(0.0%)	0(0.0%)
max	0.027	0.028	0.00097
val	-0.001073837	-0.001074791	0.125244141
kern	-0.001045227	-0.001045227	0.125366211

Push CPU to emulate the behavior of the xdlops pipeline

- `acc 4`: the `mfma` instructions are performed by the `DOT4_F32_F16` unit, which performs a fused multiply and add of 4 pairs of fp16 numbers
  $\Rightarrow$ slightly improves the results
- `flush`: `mfma` instructions on MI200
  $\Rightarrow$ greatly improves the results

`acc 4`: truncate the sum of every 4 multiplications from fp64 to fp32

`flush`: flush small numbers ($< 2^{-14}$) to zero

Summary

Causes of numerical errors for fp16 tests:

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

Element wise vs. RMS

- Element wise metrics are more sensitive to numerical errors
  $\Rightarrow$ not suitable for small inputs
- RMS is more sensitive to the magnitude of the elements
  $\Rightarrow$ not suitable for large inputs

Upcoming Testing Framework

1. Combine RMS and element wise metrics together (MLIR#620)
    - For random inputs with [-1, 1], use RMS
    - For random inputs with [-3, -1] and [1, 3], use maxRelDiff
2. Let each test decide how to verify the results (MLIR#621)
    - choose the range of random numbers
    - choose which metric to use
    - choose the tolerance for each metric
        - RMS: 3×10^{-5}
        - maxRelDiff: 1×10^{-3}