GPU-FPX and FTTN

Xinyi Li* , Ganesh Gopalakrishnan**

* Postdoctoral Fellow, Pacific Northwest National Laboratory, Richland, WA
xinyinicole.com
** Professor, Kahlert School of Computing, University of Utah, Salt Lake City, UT
https://www.cs.utah.edu/~ganesh

Presenters

Prof. Ganesh Gopalakrishnan

Professor,
Kahlert School of Computing,
University of Utah, Salt Lake City, UT

https://www.cs.utah.edu/~ganesh

Dr. Xinyi Li

Postdoctoral Associate,
Pacific Northwest National Laboratory,
Richland, WA

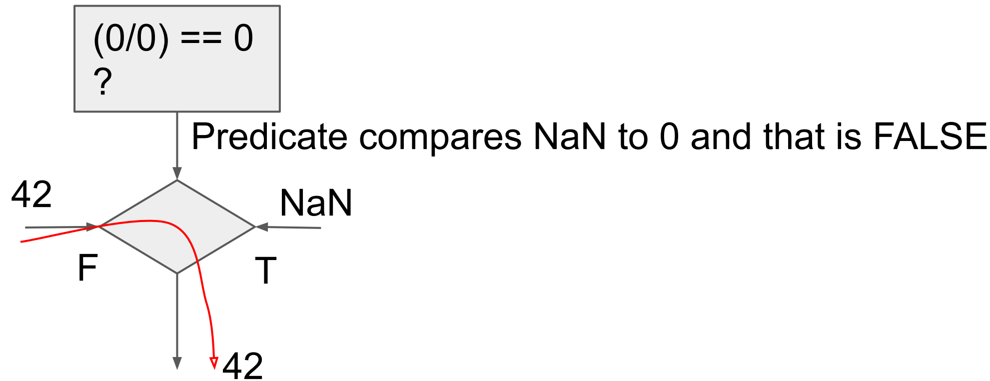
https://xinyinicole.com

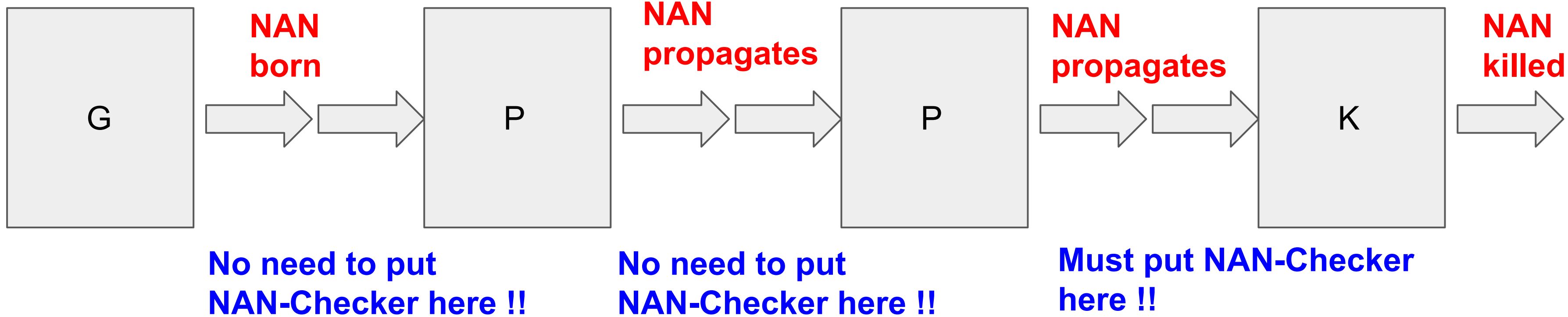
TOOLS PRESENTED :
GPU-FPX: For detecting Floating-Point Exceptions
FTTN: To discover the numerical behavior of Tensor Cores

- FP Exceptions
  - Cause non-determinism - even serious accidents
  - May be masked, resulting in users not seeing the effects
  - Numerical solutions with NaNs are useless
  - Exceptions can change with platforms
  - Exception-checking support is limited in Heterogeneous Hardware
  - Closed-source libraries make the problem worse
  - We provide the first usable binary-instrumentation framework for NVIDIA GPUs
  - Long term: Manufacturer Help is essential

- FTTN
  - Important to know the "HW substrate" which increasingly uses Tensor-Cores
  - These are poorly documented
  - In the short-term, we must experimentally discover what the behaviors are
  - We provide a set of tests to reveal many of the important Tensor-Core behaviors
  - Long term: Manufacturer Help is Essential

NaN … how perhaps born, how it flows, how likely killed





GPU-FPX Tool Efficiently Detects Exceptions in NVIDIA GPU Binaries

X. Li, I. Laguna, B. Fang, K. Swirydowicz, A. Li and G. Gopalakrishnan, ***"Design and Evaluation of GPU-FPX: A Low-Overhead tool for Floating-Point Exception Detection in NVIDIA GPUs,"*** HPDC '23: Proceedings of the 32nd International Symposium on High-Performance Parallel and Distributed Computing, August 2023, Pages 59–71, https://doi.org/10.1145/3588195.3592991

- HPDC 2023 paper published on **new tool GPU-FPX** released at https://github.com/LLNL/GPU-FPX
- Found 27 previously unknown exceptions detected across 151 programs on their own data sets
    - Some repairs also identified based on tool feedback

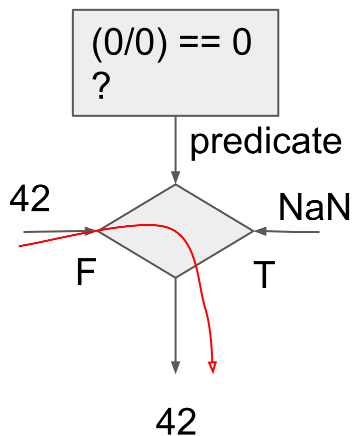


Table 7. Overview of Exception Diagnoses and Repairs using *Analyzer* for Programs with Severe Exceptions

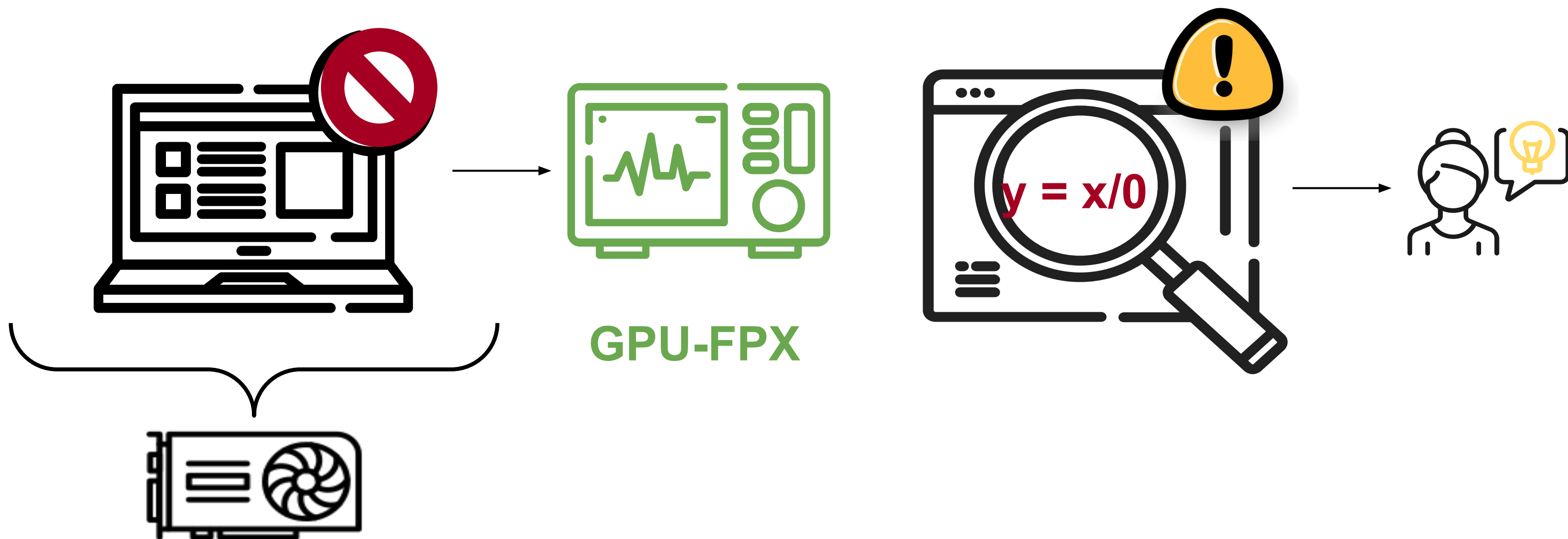
Program	Source available?	Diagnose?	Exceptions Matter?	Fixed?	How Fixed?
GRAMSCHM	yes	yes	yes	yes	Remove 0 from input
LU	yes	yes	yes	yes	Remove 0 from input
myocyte	yes	no	N.A.	N.A.	N.A.
S3D	yes	yes	no	N.A.	N.A.
Interval	yes	yes	no	N.A.	N.A.
Laghos	yes	no	N.A.	N.A.	N.A.
Sw4lite	yes	no	N.A.	N.A.	N.A.
HPCG	no	no	N.A.	N.A.	N.A.
CuMF-Movielens	yes	yes	yes	yes	Enforce variable consistency
cuML-HousePrice	partial	yes	yes	partial	N.A.
CUDA GMRES	partial	yes	yes	partial	Diagonal boosting
SRU-Example	yes	yes	yes	yes	Change input generator

Non-Deterministic Behavior: This is incorrect in FP

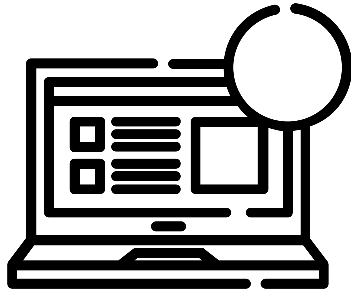
$$\text{\#define MAX}(x, y)\,((x) \geq (y) ? (x) : (y))$$

GPU-FPX

A Low-Overhead tool for Floating-Point Exception Detection in NVIDIA GPUs



Programs may not run correctly

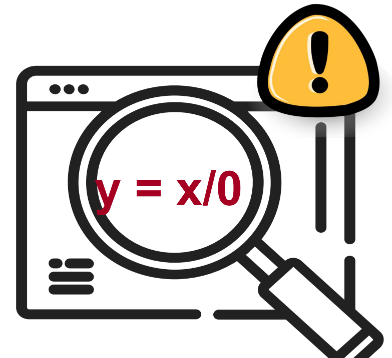


output

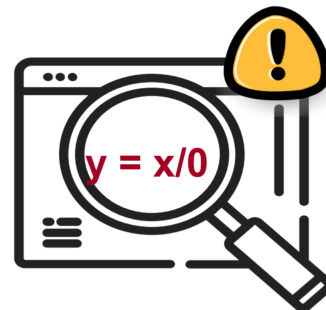
NaN, INF result

Just not correct answer

All these may stem from a simple floating-point exception

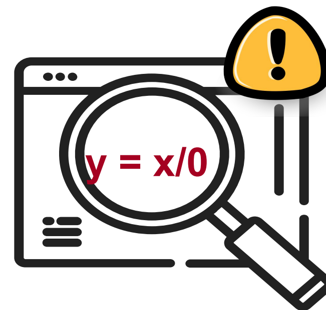


Floating point exceptions



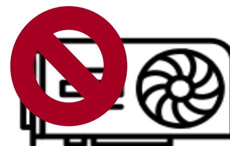
Invalid Operations	Division by Zero	Overflow	Underflow
Resulting in NaN	Resulting in NaN, INF	Resulting in INF	Resulting in Subnormal
sqrt(-1)	0/0, 3/0	power(2,2000)	1.0e-308 / 1.0e308

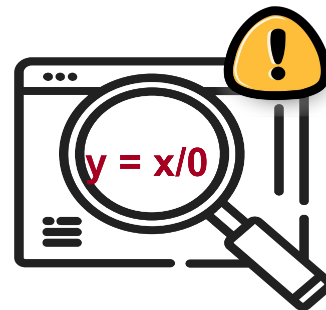
Hardware exception traps can be enabled on CPUs



Invalid Operations	Division by Zero	Overflow	Underflow
Resulting in NaN	Resulting in NaN, INF	Resulting in INF	Resulting in Subnormal
sqrt(-1)	0/0, 3/0	power(2,2000)	1.0e-308 / 1.0e308

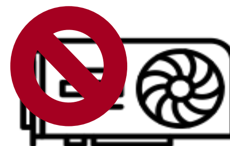
... but not on NVIDIA GPUs!!

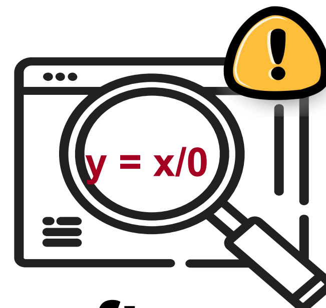




Invalid Operations	Division by Zero	Overflow	Underflow
Resulting in NaN	Resulting in NaN, INF	Resulting in INF	Resulting in Subnormal
sqrt(-1)	0/0, 3/0	power(2,2000)	1.0e-308 / 1.0e308

… but not on NVIDIA GPUs!!

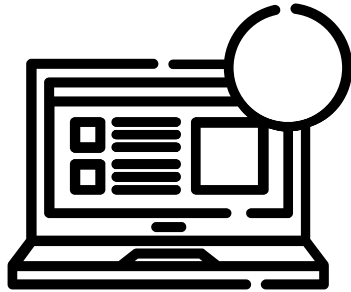




requiring exceptions to be detected by examining the results in **software**

Invalid Operations	Division by Zero	Overflow	Underflow
Resulting in NaN	Resulting in NaN, INF	Resulting in INF	Resulting in Subnormal
sqrt(-1)	0/0, 3/0	power(2,2000)	1.0e-308 / 1.0e308

Programs may not run correctly

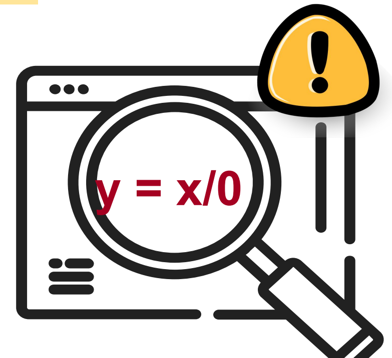


output

NaN, INF result

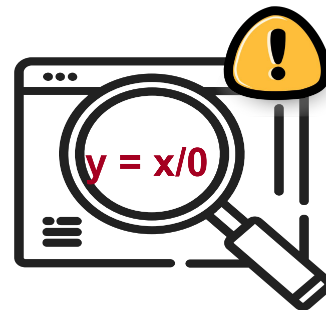
Just not correct answer

All these may stem from a simple floating-point exception



Show-Stopper

Floating-Point Exceptions on GPUs



A x = b

A is a near singular matrix

Run on GPU

No warning raised

Loss became NaN !!

If A < B then P else Q

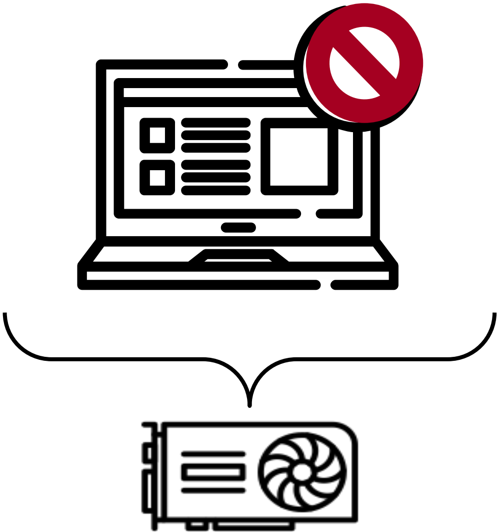
If Either A or B is NaN

then Q is executed

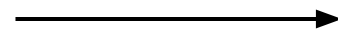
P is ignored ...

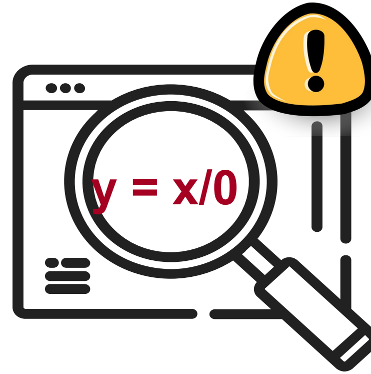
P may contain a NaN too

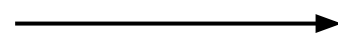
User has no tools to root-cause and fix!



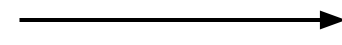
Need tool to detect and analyze Floating-point exceptions

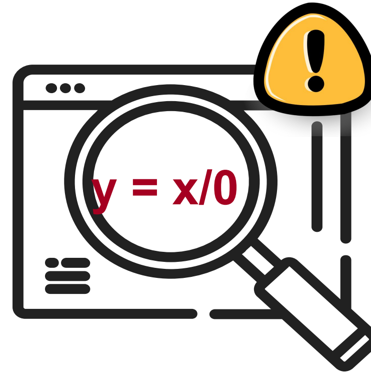




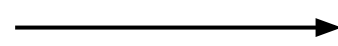


Need tool to detect and analyze Floating-point exceptions





GPU-FPX



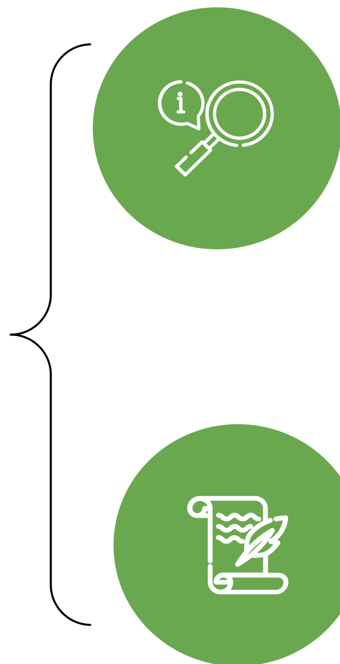
GPU-FPX: 3 features

GPU-FPX

- Fast
- At Binary Level
  - Important libraries are closed source. e.g. cuSolver
  - Compiler may change the floating-point behavior. e.g. -fast-math flag
- Helps Diagnose

GPU-FPX: 2 components

GPU-FPX



Detector

Pinpoints exception-generating locations across all kernels

Analyzer

Reports how exceptions flow within one instruction

GPU-FPX

GPU-FPX: 1 simple demo

GPU-FPX

```
__global__ void dot_prod(float *x, float *y, int size)
{
float d;
for (int i=0; i < size; ++i)
{
float tmp;
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
d += tmp; // d=NaN
}
int tid = blockIdx.x * blockDim.x + threadIdx.x;
if (tid == 0) {
printf("dot: %f\n", d);
}
}
```

GPU-FPX

Use Detector

```
__global__ void dot_prod(float *x, float *y, int size)
{
float d;
```

```
LD_PRELOAD=detector.so ./dot-prod
```

```
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
d += tmp; // d=NaN
}
int tid = blockIdx.x * blockDim.x + threadIdx.x;
if (tid == 0) {
printf("dot: %f\n", d);
}
}
```

Use Detector

```
__global__ void dot_prod(float *x, float *y, int size)
{
float d;
```

```
LD_PRELOAD=detector.so ./dot-prod
```

```
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
```

```
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], DIV0 found @ dot-prod.cu:13 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:13 [FP32]
dot: nan
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:21 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:14 [FP32]
```

```
}
}
```

GPU-FPX

Use Detector

```
__global__ void dot_prod(float *x, float *y, int size)
{
float d;
```

```
LD_PRELOAD=detector.so ./dot-prod
```

```
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
```

```
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], DIV0 found @ dot-prod.cu:13 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:13 [FP32]
dot: nan
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:21 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [dot_prod], NaN found @ dot-prod.cu:14 [FP32]
```

Using the Analyzer

```
LD_PRELOAD=analyzer.so ./dot-prod
```

```
float d;
for (int i=0; i < size; ++i)
{
float tmp;
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
```

```
GPU-FPX-ANA APPEAR : INF appear at the destination  @ dot-prod.cu:13 Instruction: MUFU.RCP
R0, R10 ; We have 2 registers in total. Register 0 is INF. Register 1 is VAL.
#GPU-FPX-ANA APPEAR : NaN appear at the destination  @ dot-prod.cu:13 Instruction: FFMA
R9, -R10, R0, 1 ; We have 3 registers in total. Register 0 is NaN. Register 1 is VAL.
Register 2 is INF.
#GPU-FPX-ANA PROPAGATION: …
```

```
}
```

Using the Analyzer

```
LD_PRELOAD=analyzer.so ./dot-prod
```

```
float d;
for (int i=0; i < size; ++i)
{
float tmp;
// division by zero, produce NaN
tmp = x[i]*y[i] / 0
```

```
#GPU-FPX-ANA APPEAR : INF appear at the destination  @ dot-prod.cu:13 Instruction: MUFU.RCP R0, R10 ; We have 2 registers in total. Register 0 is INF. Register 1 is VAL.
#GPU-FPX-ANA APPEAR : NaN appear at the destination  @ dot-prod.cu:13 Instruction: FFMA R9, -R10, R0, 1 ; We have 3 registers in total. Register 0 is NaN. Register 1 is VAL. Register 2 is INF.
#GPU-FPX-ANA PROPAGATION: ...
```

Next step

Low or mixed-precision exceptions detect and analysis

- AI/ML workload are using lower precision
- Libraries are developed to use mixed precision
  - `torch.autocast`
- Current hardware vendor are developed low-precision unit
  - NVIDIA tensor cores, AMD matrix cores
- More exceptional issues about using mixed precision

Next step

Low or mixed-precision exceptions detect and analysis

- AI/ML workload are using lower precision
- Libraries are developed to use mixed precision
  - `torch.autocast`
- Current hardware vendor are developed low-precision unit
  - NVIDIA tensor cores, AMD matrix cores
- More exceptional issues about using mixed precision

We studied their numerical behaviors

Will cover in the next half

Matrix Accelerators

Nowadays GPUs are always equipped with special hardware for **mixed-precision matrix multiplication** D = A*B+C

$$D = \begin{pmatrix} A_{0,0} & A_{0,1} & A_{0,2} & A_{0,3} \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} \\ A_{2,0} & A_{2,1} & A_{2,2} & A_{2,3} \\ A_{3,0} & A_{3,1} & A_{3,2} & A_{3,3} \end{pmatrix} \begin{pmatrix} B_{0,0} & B_{0,1} & B_{0,2} & B_{0,3} \\ B_{1,0} & B_{1,1} & B_{1,2} & B_{1,3} \\ B_{2,0} & B_{2,1} & B_{2,2} & B_{2,3} \\ B_{3,0} & B_{3,1} & B_{3,2} & B_{3,3} \end{pmatrix} + \begin{pmatrix} C_{0,0} & C_{0,1} & C_{0,2} & C_{0,3} \\ C_{1,0} & C_{1,1} & C_{1,2} & C_{1,3} \\ C_{2,0} & C_{2,1} & C_{2,2} & C_{2,3} \\ C_{3,0} & C_{3,1} & C_{3,2} & C_{3,3} \end{pmatrix}$$

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

Run on

Tensor core

Matrix core

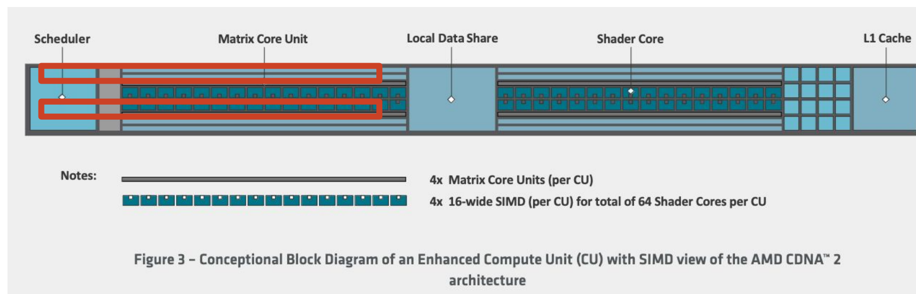


Figure 3 – Conceptional Block Diagram of an Enhanced Compute Unit (CU) with SIMD view of the AMD CDNA™ 2 architecture

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Part 2:
GPU-FPX in practice
Debugging a SRU open issue

Problem description

- Link: https://github.com/asappresearch/sru/issues/193

hoagy-davis-digges commented on Nov 2, 2021 • edited

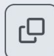
I have run the example code in the readme on both 2.6.0 and 3.0.0-dev and both have nan values in both the output and state objects using pytorch 1.9, I've tried this on my computer using a Titan X and also with a fresh install on a cloud T4, this doesn't seem to relate to the other nan issue raised here https://github.com/asappresearch/sru/issues/185 because this problem appears immediately.

Problem description

- Link: https://github.com/asappresearch/sru/issues/193

taolei87 commented on Nov 5, 2021 Contributor ...

hi **@hoagy-davis-digges** , did you mean you tried the following example and got NaN?



```
import torch
from sru import SRU, SRUCell

# input has length 20, batch size 32 and dimension 128
x = torch.FloatTensor(20, 32, 128).cuda()

input_size, hidden_size = 128, 128

rnn = SRU(input_size, hidden_size,
    num_layers = 2,          # number of stacking RNN layers
    dropout = 0.0,           # dropout applied between RNN layers
    bidirectional = False,   # bidirectional RNN
    layer_norm = False,      # apply layer normalization on the output of each layer
    highway_bias = -2,        # initial bias of highway gate (<= 0)
)
rnn.cuda()

output_states, c_states = rnn(x)      # forward pass
```

Use Detector

```
Running #GPU-FPX: kernel [ampere_sgemm_32x128_nn] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [ampere_sgemm_32x128_nn], NaN found @ /unknown_path in [ampere_sge
mm_32x128_nn]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [ampere_sgemm_32x128_nn], very small quantity (SUB) found
 @ /unknown_path in [ampere_sgemm_32x128_nn]:0 [FP32]
Running #GPU-FPX: kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], NaN
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], INF
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], DIV0
 found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simp
le], very small quantity (SUB) found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward
_kernel_simple]:0 [FP32]
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::CatArrayBatchedCopy] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void at::native::vectorized_elementwise_kernel], NaN found @ /unk
nown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void at::native::vectorized_elementwise_kernel], very sm
all quantity (SUB) found @ /unknown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceReduceSingleTileKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceCompactInitKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceSelectSweepKernel] ...
Running #GPU-FPX: kernel [void at::native::index_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::unrolled_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::reduce_kernel] ...
```

Closed-source library

Use Detector

```
Running #GPU-FPX: kernel [ampere_sgemm_32x128_nn] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [ampere_sgemm_32x128_nn], NaN found @ /unknown_path in [ampere_sge
mm_32x128_nn]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [ampere_sgemm_32x128_nn], very small quantity (SUB) found
 @ /unknown_path in [ampere_sgemm_32x128_nn]:0 [FP32]
Running #GPU-FPX: kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], NaN
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], INF
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], DIV0
 found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simp
le], very small quantity (SUB) found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward
_kernel_simple]:0 [FP32]
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::CatArrayBatchedCopy] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void at::native::vectorized_elementwise_kernel], NaN found @ /unk
nown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void at::native::vectorized_elementwise_kernel], very sm
all quantity (SUB) found @ /unknown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceReduceSingleTileKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceCompactInitKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceSelectSweepKernel] ...
Running #GPU-FPX: kernel [void at::native::index_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::unrolled_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::reduce_kernel] ...
```

Use Detector

Closed-source library

```
Running #GPU-FPX: kernel [ampere_sgemm_32x128_nn] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [ampere_sgemm_32x128_nn], NaN found @ /unknown_path in [ampere_sge
mm_32x128_nn]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [ampere_sgemm_32x128_nn], very small quantity (SUB) found
 @ /unknown_path in [ampere_sgemm_32x128_nn]:0 [FP32]
Running #GPU-FPX: kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], NaN
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], INF
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], DIV0
 in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
O: Warning: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simp
ty (SUB) found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward
_kernel_simple]:0 [FP32]
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::CatArrayBatchedCopy] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void at::native::vectorized_elementwise_kernel], NaN found @ /unk
nown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void at::native::vectorized_elementwise_kernel], very sm
all quantity (SUB) found @ /unknown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceReduceSingleTileKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceCompactInitKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceSelectSweepKernel] ...
Running #GPU-FPX: kernel [void at::native::index_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::unrolled_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::reduce_kernel] ...
```

Not clear how to fix it

Use Analyzer

Speedup by enabling necessary kernels

```
Running #GPU-FPX: kernel [ampere_sgemm_32x128_nn] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [ampere_sgemm_32x128_nn], NaN found @ /unknown_path in [ampere_sge
mm_32x128_nn]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [ampere_sgemm_32x128_nn], very small quantity (SUB) found
 @ /unknown_path in [ampere_sgemm_32x128_nn]:0 [FP32]
simple] ...
rd_kernel_simple], NaN
simple]:0 [FP32]
rd_kernel_simple], INF
found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simple], DIV0
 found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward_kernel_simple]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void (anonymous namespace)::sru_cuda_forward_kernel_simp
le], very small quantity (SUB) found @ /unknown_path in [void (anonymous namespace)::sru_cuda_forward
_kernel_simple]:0 [FP32]
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::CatArrayBatchedCopy] ...
#GPU-FPX LOC-EXCEP INFO: in kernel [void at::native::vectorized_elementwise_kernel], NaN found @ /unk
nown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
#GPU-FPX LOC-EXCEP INFO: Warning: in kernel [void at::native::vectorized_elementwise_kernel], very sm
all quantity (SUB) found @ /unknown_path in [void at::native::vectorized_elementwise_kernel]:0 [FP32]
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceReduceSingleTileKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceCompactInitKernel] ...
Running #GPU-FPX: kernel [void at_cuda_detail::cub::DeviceSelectSweepKernel] ...
Running #GPU-FPX: kernel [void at::native::index_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::unrolled_elementwise_kernel] ...
Running #GPU-FPX: kernel [void at::native::reduce_kernel] ...
```

First and most exceptions happened in this kernel, so we can limit our instrumentation in this kernel

Analysis

```
Running #GPU-FPX: kernel [ampere_sgemm_32x128_nn] ...
#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [ampere_sgemm_32x1
28_nn]:0 Instruction: FFMA R1, R32.reuse, R40.reuse, R1 ; We have 4 registers in total. Register 0 is
 VAL. Register 1 is VAL. Register 2 is NaN. Register 3 is VAL.

#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [ampere_sgemm_32x1
28_nn]:0 Instruction: FFMA R0, R32, R41.reuse, R0 ; We have 4 registers in total. Register 0 is VAL.
Register 1 is VAL. Register 2 is NaN. Register 3 is VAL.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction  @ /unknown_path in [ampere_sgemm_32x12
8_nn]:0 Instruction: FFMA R1, R32.reuse, R40.reuse, R1 ; We have 4 registers in total. Register 0 is
NaN. Register 1 is VAL. Register 2 is NaN. Register 3 is NaN.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction  @ /unknown_path in [ampere_sgemm_32x12
8_nn]:0 Instruction: FFMA R0, R32, R41.reuse, R0 ; We have 4 registers in total. Register 0 is NaN. R
egister 1 is VAL. Register 2 is NaN. Register 3 is NaN.
```

Analysis

```
#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [ampere_sgemm_32x1
28_nn]:0 Instruction: FFMA R1, R32.reuse, R40.reuse, R1                                    ster 0 is
 VAL. Register 1 is VAL. Register 2 is NaN. Register 3 i

#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [ampere_sgemm_32x1
28_nn]:0 Instruction: FFMA R0, R32, R41.reuse, R0 ; We have 4 registers in total. Register 0 is VAL.
Register 1 is VAL. Register 2 is NaN. Register 3 is VAL.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction  @ /unknown_path in [ampere_sgemm_32x12
8_nn]:0 Instruction: FFMA R1, R32.reuse, R40.reuse, R1 ; We have 4 registers in total. Register 0 is
NaN. Register 1 is VAL. Register 2 is NaN. Register 3 is NaN.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction  @ /unknown_path in [ampere_sgemm_32x12
8_nn]:0 Instruction: FFMA R0, R32, R41.reuse, R0 ; We have 4 registers in total. Register 0 is NaN. R
egister 1 is VAL. Register 2 is NaN. Register 3 is NaN.
```

NaN seems already exists within the initial data!

Analysis

This creates a tensor with uninitialized data on GPU memory.

ributor ···

```
import torch
from sru import SRU, SRUCell

# input has length 20, batch size 32 and dimension 128
x = torch.FloatTensor(20, 32, 128).cuda()

input_size, hidden_size = 128, 128

rnn = SRU(input_size, hidden_size,
    num_layers = 2,          # number of stacking RNN layers
    dropout = 0.0,           # dropout applied between RNN layers
    bidirectional = False,   # bidirectional RNN
    layer_norm = False,      # apply layer normalization on the output of each layer
    highway_bias = -2,        # initial bias of highway gate (<= 0)
)
rnn.cuda()

output_states, c_states = rnn(x)      # forward pass
```

Fix

This creates a tensor with uninitialized data on GPU memory.

ributor ···

```
import torch
from sru import SRU, SRUCell

# input has length 20, batch size 32 and dimension 128
x = torch.FloatTensor(20, 32, 128).cuda()

input_size, hidden_size = 128, 128

rnn = SRU(input_size, hidden_size,
    num_layers = 2,          # number of stacking RNN layers
    dropout = 0.0,           # dropout applied between RNN layers
    bidirectional = False,   # bidirectional RNN
    layer_norm = False,      # apply layer normalization on the output of each layer
    highway_bias = -2,        # initial bias of highway gate (<= 0)
)
rnn.cuda()

output_states, c_states = rnn(x)      # forward pass
```

→ torch.randn(20,32,128).cuda()

Matrix Accelerators

Matrix Accelerators

Nowadays GPUs are always equipped with special hardware for **mixed-precision matrix multiplication** D = A*B+C

D = A × B + C

FP16 or FP32 | FP16 | FP16 | FP16 or FP32

Run on

Tensor core

Matrix core

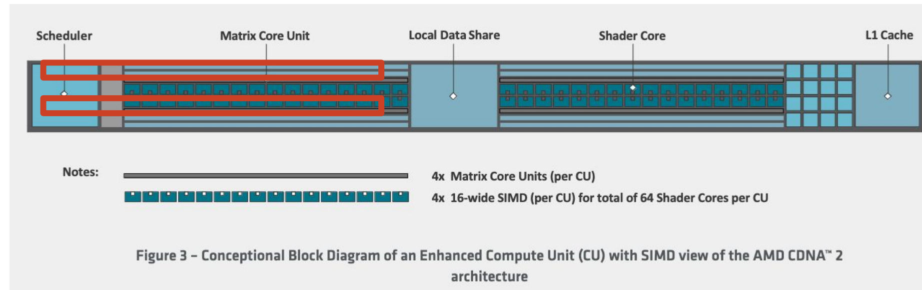
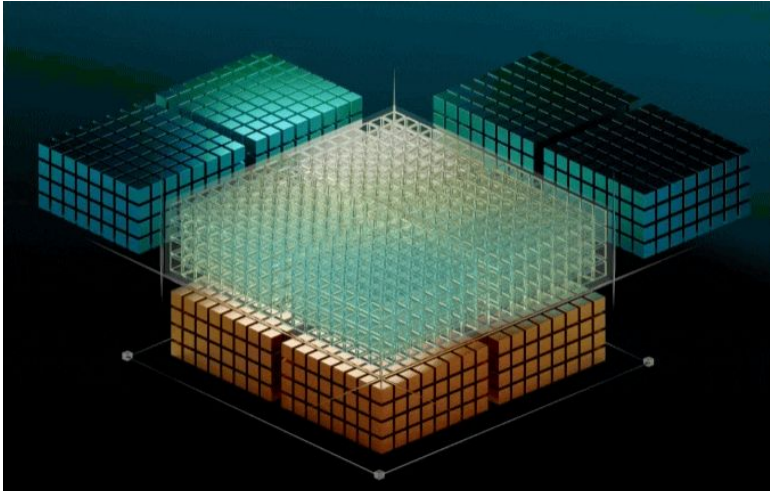


Figure 3 – Conceptional Block Diagram of an Enhanced Compute Unit (CU) with SIMD view of the AMD CDNA™ 2 architecture

Matrix Accelerators Issues

Nowadays GPUs are always equipped with special hardware for (mixed-precision) matrix multiplication D = A*B+C

Matrix Accelerators



A specific hardware to speed up matrix multiplication D=A*B+C

Mixed-precision computation

Block-wise computation

Lack of numerical standardization! Numerical inconsistency

A numerical inconsistent example caused by matrix accelerators

Two $2^{13} \times 2^{13}$ matrix doing the matrix multiplication

$$D = -1 \times A \times B + 1 \times C$$

$$D = -1 \times \begin{bmatrix} 2^{10} & -2^{-2} & -2^{-3} & -2^{-2} & -2^{-3} & \dots \\ \multicolumn{6}{c}{\text{... same as the first row}} \end{bmatrix} \times \begin{bmatrix} 2^{10} & \text{... same as the first column} \\ 2^{-3} & \\ \dots & \end{bmatrix} + 1 \times \begin{bmatrix} 2^{20} & 2^{20} & \dots \\ \multicolumn{3}{c}{\text{... All } 2^{20}} \end{bmatrix}$$

Each element in D should be the same

$$D_{ij} = -(2^{10} * 2^{10} - \Sigma 2^{-2} * 2^{-3} - \Sigma 2^{-3} * 2^{-3}) + 2^{20} = 2^{7} + 2^{6} - 2^{-6} \approx 191.99218$$

Run on different hardware (with matrix accelerators

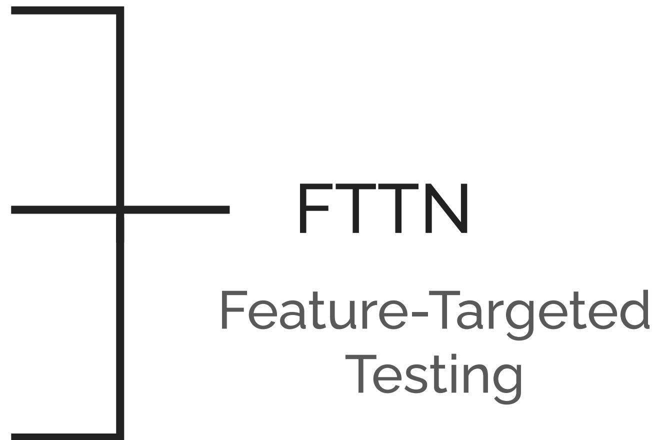
NVIDIA V100	NVIDIA A100	NVIDIA H100	AMD MI100	AMD MI250	CPU
0	0	191.875	255.875	0	0

Numerical behaviors we want to test

Support for subnormals

Rounding mode
- Extra bits?
- Rounding direction?

Block-FMA features
- When to round and normalization
- Width of FMA block



Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Subnormal supported

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Extra bits and rounding mode

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Extra bits and rounding mode

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Extra bits and rounding mode

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Blocked FMA feature

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Blocked FMA feature

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Blocked FMA feature

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Blocked FMA feature

Inputs	GPU	Subnormal inputs handled?	Subnormal outputs handled?	Extra bit present? How many?	Rounding mode exhibited	FMA unit width	Order within one FMA unit is controllable?	Rounding mode for: 1. outputting FP16/BF16 (only for FP16/BF16 inputs) 2. product (only for FP32/FP64 inputs)
FP16	V100	✓	✓	0	truncate	N.A.	N.A.	RTN-TE
	A100	✓	✓	1	truncate	8	✗	RTN-TE
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	4	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
BF16	A100	✓	✓	1	truncate	8	✗	N.A.**
	H100	✓	✓	≥ 2	truncate	≥ 16	✗	RTN-TE
	MI100	✓	✓	3	RTN-TE	2	✗	RTN-TE
	MI250X	✗	✗	3	RTN-TE	1	N.A.	RTN-TE
TF32(NVIDIA) FP32(AMD)	A100	✓	✓	1	truncate	4	✗	N.A.
	H100	✓	✓	≥ 2	truncate	4	✗	N.A.
	MI100	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE
FP64	A100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	H100	✓	✓	3	RTN-TE	1	✗	RTN-TE
	MI250X	✓	✓	3	RTN-TE	1	N.A.	RTN-TE

* Since the V100 does not preserve extra bits, its FMA functionality cannot be evaluated. ** A100 doesn't support BF16 output.
The results can also be applied to GPUs with the same architecture (e.g. A100 to RTX 30 series), as they use the same generation of tensor cores.

Thanks!

1. Questions?
2. Any applications you want us to help?
3. What features you want to add?

Try GPU-FPX!

Try FTTN!

Thanks!