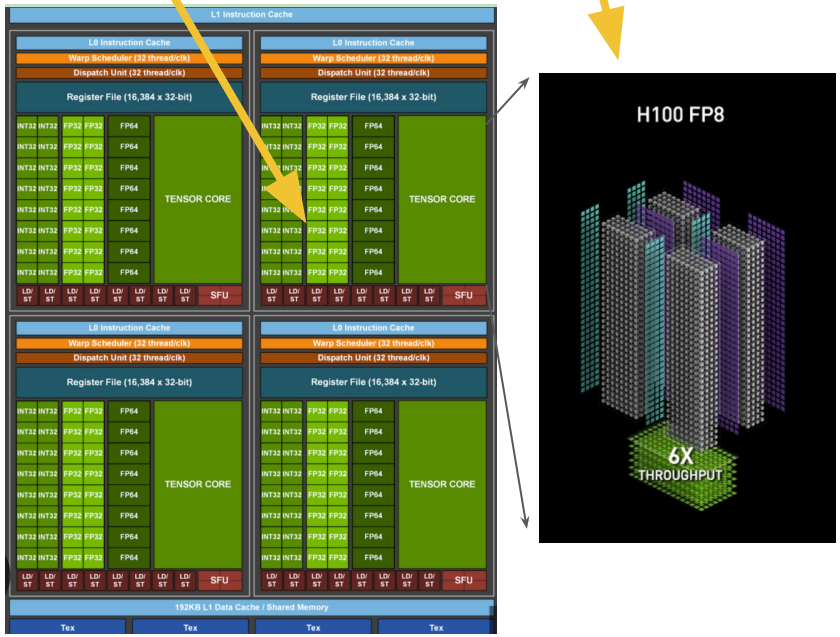
Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- **09:35 : NixNan (Part-1)**
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

NixNan : Detecting Exceptions in NVIDIA GPUs

Detects exceptions in

SIMT Cores as well as Tensor Cores



NixNan : Detecting Exceptions in NVIDIA GPUs

Detects exceptions in

SIMT Cores as well as Tensor Cores

- NVIDIA GPUs do not generate any traps when an FP exception occurs
- Thus, we have to detect exceptions in software
  - By decoding the exponent and mantissa of the computed answer

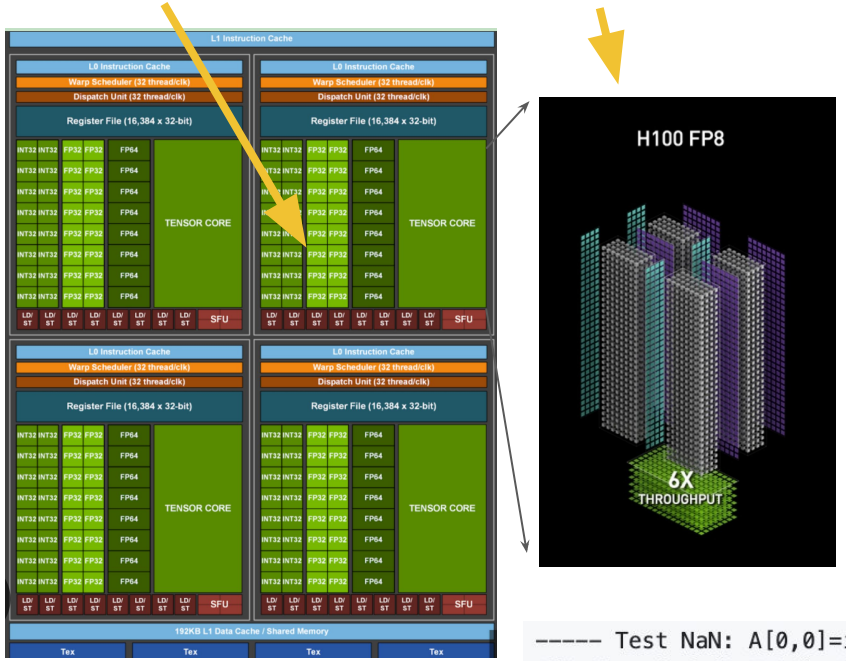
S	E	M

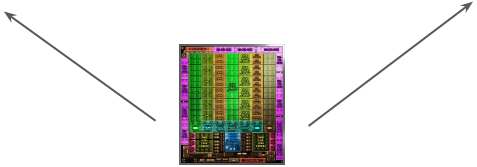
FF	INF (if 0) or NaN (if not)
0	non-zero => subnormal

NixNan Detects Exceptions in NVIDIA GPUs per Kernel Launch

Detects exceptions in

SIMT Cores as well as Tensor Cores





> LD_PRELOAD = nixnan.so
> Pre-Compiled Application Binary

Intercept all "Kernel Launch" CUDA Driver API Calls

For Each FP instruction :

Inject_instrumentation();

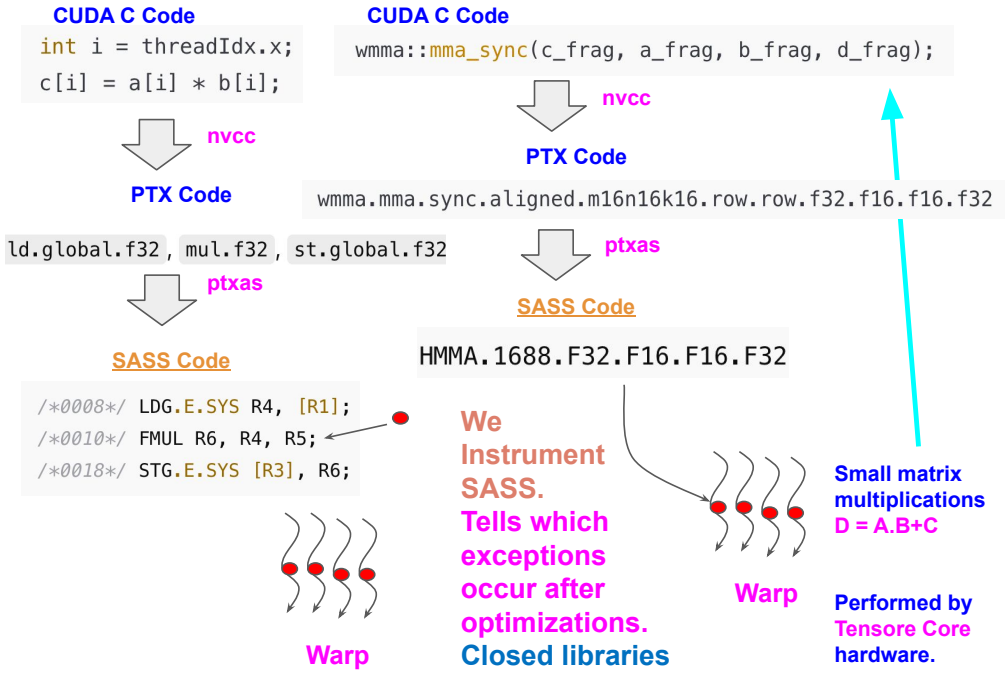
CUDA Driver APIs

```
#nixnan: ------------ nixnan Report

#nixnan: --- FP16 Operations ---
#nixnan: NaN: 4
#nixnan: Infinity: 3
#nixnan: Subnormal: 2
#nixnan: Division by 0: 0

#nixnan: --- FP32 Operations ---
#nixnan: NaN: 0
#nixnan: Infinity: 0
#nixnan: Subnormal: 0
#nixnan: Division by 0: 0

#nixnan: --- FP64 Operations ---
#nixnan: NaN: 0
#nixnan: Infinity: 0
#nixnan: Subnormal: 0
#nixnan: Division by 0: 0
```

```
----- Test NaN: A[0,0]=inf, B[0,0]=1, C[0,0]=-inf -----
A[0,0] = inf (0x7c00), B[0,0] = 1.000000 (0x3c00), C[0,0] = -inf (0xfc00)
Computing D = A * B + C with Tensor Cores...
#nixnan: error [NaN,infinity] detected in instruction HMMA.16816.F16 R20, R4.reuse, R16, RZ ;
#nixnan: error [NaN] detected in instruction HMMA.16816.F16 R22, R4, R18, RZ ; in function WMM
D[0,0]=nan (0x7fff)
```

History: GPU-FPX → Evolved to become NixNan

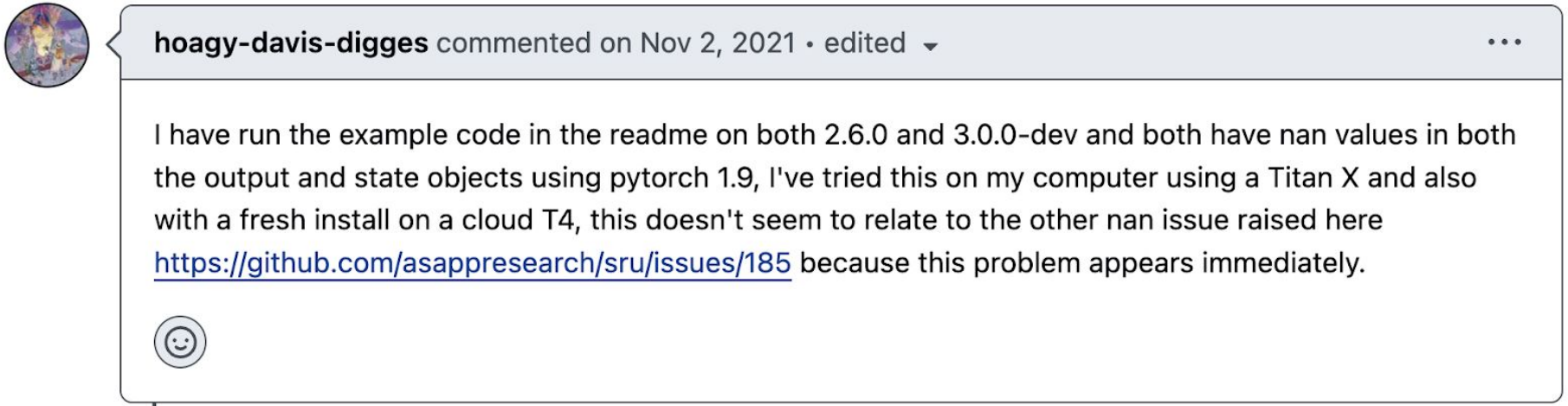
First demonstrated during HPDC 2023
Has a detector and an analyzer component that does flow analysis
Exits early when a thread detects exceptions
No support for FP16 or Tensor Cores



Continuing to refine it in response to tests (discovering missing cases each time…)
Has a single component that detects as well as flow-analyzes
Finds all exceptions (without quitting early)
Support for FP16 and Tensor Cores

Real-world Issue: **NixNan usage**

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



This case-study was done by Dr. Xinyi Li who developed GPU-FPX, the precursor to NixNan

(the latter is a refined version with bug-fixes, and Tensor-Core Support)

We have adapted Dr. Li's test and fix to NixNan.

Issue

- 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
```

Tool Usage

```
1999  python run_sru.py
2000  LD_PRELOAD=~/nixnan.so python run_sru.py
2001  LD_PRELOAD=~/nixnan.so python run_sru_fixed.py
```

Seeing NaNs early, we examined the source-code...

```
#nixnan: running kernel [void at::native::vectorized_elementwise_kernel] ...
#nixnan: running kernel [ampere_sgemm_32x128_nn] ...
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R1, R32.reu
se, R40.reuse, R1 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R0, R32, R4
1.reuse, R0 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R10, R34, R
43.reuse, R10 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R5, R33.reu
se, R40.reuse, R5 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R3, R32.reu
se, R42, R3 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R4, R33, R4
1.reuse, R4 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 2 of instruction FFMA R0, R32, R4
1.reuse, R0 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 0 of instruction FFMA R1, R36.reu
se, R44.reuse, R1 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [subnormal] detected in operand 3 of instruction FFMA R8, R38.reu
se, R45.reuse, R8 ; in function ampere_sgemm_32x128_nn of type f32
#nixnan: error [NaN,subnormal] detected in operand 0 of instruction FFMA R5, R37
.reuse, R44.reuse, R5 ; in function ampere_sgemm_32x128_nn of type f32
```

Found uninitialized the tensor; setting that fixes NaNs

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
```

→ torch.randn(20,32,128).cuda()

Exceptions Written Into Memory

Before Fixing SRU

```
#nixnan: --- FP16 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                     843 (2418829 repeats)
#nixnan: Infinity:                 13 (15635 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:               746 (251293 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                       3 (5373 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
```

Our fix was applied

After Fixing SRU

```
#nixnan: --- FP16 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: Infinity:                  0 (0 repeats)
#nixnan: -Infinity:                 0 (0 repeats)
#nixnan: Subnormal:                 0 (0 repeats)
#nixnan: Division by 0:             0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                       0 (0 repeats)
```

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- **09:55 : Plz Fill Survey**
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- **10:00 : Coffee Break**
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

Agenda + Single QR Code for those coming post-Coffee

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- **10:30 : Recap of First Half**
- **10:35 : NixNan (Part-2)**
- 11:05 : FloatGuard
- 11:50 : Closing Remarks
- 11:55 : Fill Survey

Demo of NixNan : HPC

- **We show two cases of solving a linear system (next slide)**
  - Matrix with a low condition number : **matrix_0.csv**
  - Matrix with a high condition number: **matrix_5.csv**

FP Exceptions in HPC : Ill-conditioned Matrix

```
#nixnan: ------------ nixnan Report -----------

#nixnan: --- FP16 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
```

```
--- Solution X Vector (10)
1.12338
0.799707
7.82785
-1954.76
-0.000778511
1.07278
-0.535868
-3.61902
8.83203
0.585185
```

LD_PRELOAD=~/nixnan.so
CUDA_MATRIX=matrix_0.csv ./LU_solver

LD_PRELOAD=~/nixnan.so
CUDA_MATRIX=matrix_5.csv ./LU_solver

```
--- Solution X Vector (10)
1.4665
inf
2.15945
0.895458
10.9171
1.11917
inf
24.0914
0.421027
-1.39342
```

```
#nixnan: ------------ nixnan Report -----------

#nixnan: --- FP16 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- BF16 Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- FP32 Operations ---
#nixnan: NaN:                      4 (0 repeats)
#nixnan: Infinity:                 0 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            0 (0 repeats)

#nixnan: --- FP64 Operations ---
#nixnan: NaN:                     48 (0 repeats)
#nixnan: Infinity:                 2 (0 repeats)
#nixnan: -Infinity:                0 (0 repeats)
#nixnan: Subnormal:                0 (0 repeats)
#nixnan: Division by 0:            2 (2 repeats)

#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                      0 (0 repeats)
```

Demo of NixNan : ML

- A simple GPT (due to Karpathy)
  - Observed many exceptions
    - They do not seem to affect the loss
- Also ran a medical agent (BioMistral)
  - Also observed many exceptions
    - Also does not affect the loss

Innocuous (?) FP Exceptions in BioMistral

```
ganesh@beast:~/repos/newmed$ source venv/bin/activate
(venv) ganesh@beast:~/repos/newmed$ python simple_medical_agent.py
============================================================
Medical Dialog Agent - Initializing...
============================================================
Model: BioMistral/BioMistral-7B
4-bit Quantization: True

Device: cuda
Loading tokenizer...
Loading model with 4-bit quantization (low VRAM mode)...
✓ Model loaded successfully!


============================================================
          MEDICAL DIALOG AGENT
============================================================

⚠️  DISCLAIMER: This is for educational/research purposes only.
   NOT a substitute for professional medical advice.

Commands:
  - Type your symptoms or questions
  - 'clear' - Clear conversation history
  - 'save' - Save conversation to file
  - 'quit' or 'exit' - Exit the program

============================================================

You: dizzy

Agent: what did you do to yourself? You're bleeding.

You:
```

LD_PRELOAD=~/nixnan.so
python simple_medical_agent.py

In case of ML routines, a huge number of NaNs and INF are generated. Some do leak into memory, but most seem to not.

```
#nixnan: error [NaN,subnormal] detected in operand 1 of instruction FADD R11, R14, R13 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [NaN,subnormal] detected in operand 2 of instruction FADD R11, R14, R13 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [subnormal] detected in operand 1 of instruction HADD2 R5, -R4.H0_H0, -RZ.H0_H0 ; in function void at::native::elementwise_kernel of type f16
#nixnan: error [subnormal] detected in operand 0 of instruction HADD2 R5, -R4.H0_H0, -RZ.H0_H0 ; in function void at::native::elementwise_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R44.reuse, R28, RZ ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R72.reuse, R48, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 2 of instruction HMMA.16816.F32 R40, R56.reuse, R60, R40 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R56.reuse, R44, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [NaN] detected in operand 2 of instruction HMMA.16816.F32 R32, R12.reuse, R72, R32 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 2 of instruction HMMA.16816.F32 R40, R52.reuse, R68, R40 ; in function void pytorch_flash::flash_fwd_kernel of type f16
#nixnan: error [subnormal] detected in operand 0 of instruction @!P1 FMUL R103, R103, R103 ; in function void pytorch_flash::flash_fwd_kernel of type f32
#nixnan: error [NaN,subnormal] detected in operand 1 of instruction FADD R103, R103, R106 ; in function void pytorch_flash::flash_fwd_kernel of type f32
```

```
#nixnan: ---------- nixnan Report ----------
#nixnan: --- FP16 Operations ---
#nixnan: NaN:              1000 (10890400 repeats)
#nixnan: Infinity:          253 (8631 repeats)
#nixnan: -Infinity:         119 (8037 repeats)
#nixnan: Subnormal:        1195 (1868286503 repeats)
#nixnan: Division by 0:       0 (0 repeats)
#nixnan: --- BF16 Operations ---
#nixnan: NaN:                 0 (0 repeats)
#nixnan: Infinity:            0 (0 repeats)
#nixnan: -Infinity:           0 (0 repeats)
#nixnan: Subnormal:           0 (0 repeats)
#nixnan: Division by 0:       0 (0 repeats)
#nixnan: --- FP32 Operations ---
#nixnan: NaN:              2550 (79727147 repeats)
#nixnan: Infinity:          548 (75451 repeats)
#nixnan: -Infinity:        1102 (2786872 repeats)
#nixnan: Subnormal:         263 (20232 repeats)
#nixnan: Division by 0:       0 (0 repeats)
#nixnan: --- FP64 Operations ---
#nixnan: NaN:                 0 (0 repeats)
#nixnan: Infinity:            0 (0 repeats)
#nixnan: -Infinity:           0 (0 repeats)
#nixnan: Subnormal:           0 (0 repeats)
#nixnan: Division by 0:       0 (0 repeats)
#nixnan: --- FP16 Memory  Operations ---
#nixnan: NaN:                 0 (0 repeats)
#nixnan: --- BF16 Memory  Operations ---
#nixnan: NaN:                 0 (0 repeats)
#nixnan: --- FP32 Memory  Operations ---
#nixnan: NaN:                 0 (0 repeats)
#nixnan: --- FP64 Memory  Operations ---
#nixnan: NaN:                 0 (0 repeats)
```

One NixNan Use: Vary Model Parameters; then Observe Exception Diffs

Experimental Variation Studied (Claude-created)	What It Changes (Claude)	Expected NaN Reduction (Claude)	BF16 (as observed)	FP32 (as observed)
baseline	Nothing (reference)	0%		NaN: 4592 (22306080) inf: 1010 (666054) -inf: 1616 (9361693)
bfloat16	Compute dtype FP16→BF16	70-90%	NaN: 1298 (11852994) inf: 396 (189380) -inf: 586 (918390) subn: 1124 (11440312)	
eager	Disable Flash Attention	60-80%		NaN: 1336 (191306408) inf: 744 (863768) -inf: 824 (1226989)
layernqorm_eps	LayerNorm epsilon 1e-5→1e-3	20-40%		NaN: 4759 (218503344) inf: 1129 (730696) -inf: 1799 (9400170) subn: 1237 (42303)
attention_scale	Scale attention scores	30-50%	TBD	TBD
attention_clip	Clip attention logits	40-60%		Nan: 458 (4623) -inf: 248 (28452)

In ML, Frameworks such as PyTorch Silently "Fix" Exceptions, but this can be Slow / Imperfect (believable ChatGPT summary below)

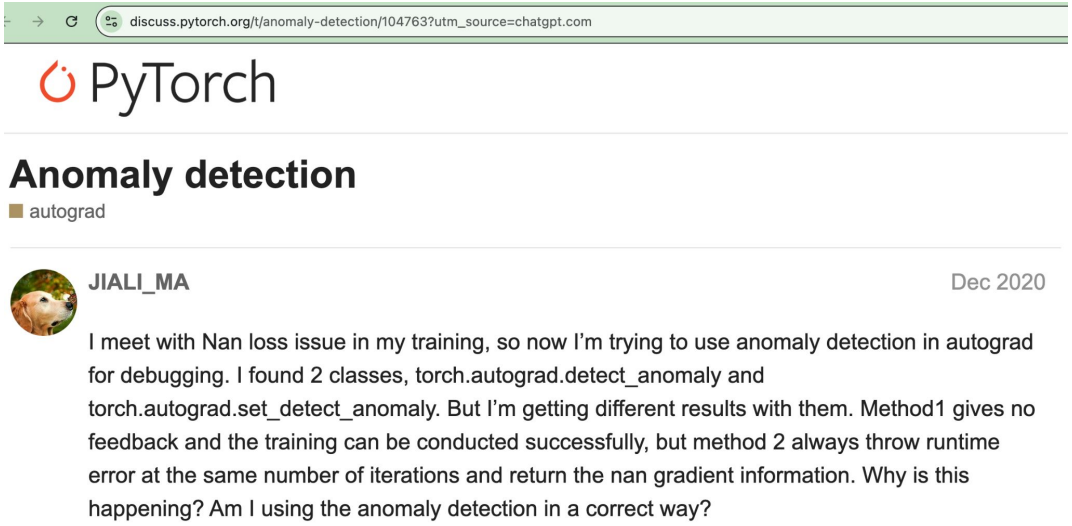
2 What PyTorch does automatically

Layer	What happens internally	Key API / behaviour
Forward kernels	Produces INF/NaN normally (IEEE "masked exceptions")—no trap.	`torch.isfinite`, `tensor.isnan()` etc. for explicit checks.
autograd anomaly detection	Wrap graph execution; throws when the *first* backward op creates a non-finite grad and prints the forward traceback.	`torch.autograd.detect_anomaly()` or `set_detect_anomaly(check_nan=True)` PyTorch Forums PyTorch
Mixed-precision scaler	`GradScaler` un-scales grads, then runs a fused `isfinite` reduction **on-GPU**. If any grad is non-finite the *optimizer step is skipped* and the loss-scale is halved.	`scaler.step(optim)` / `scaler.update()` ; see AMP docs PyTorch PyTorch Forums
High-level trainers	Lightning, DeepSpeed, Accelerate all keep a "gradient overflow" flag; a skipped step just logs **OVERFLOW** and carries on.	DeepSpeed fp16 messages "Overflow detected. Skipping step." GitHub
Linalg ops	PyTorch's `torch.linalg` wrappers pass inputs to cuSOLVER/CPU LAPACK **without extra guards**; docs explicitly warn to pre-check with `torch.isfinite`. PyTorch	

PyTorch Issue 160016 tells that this can be "porous"

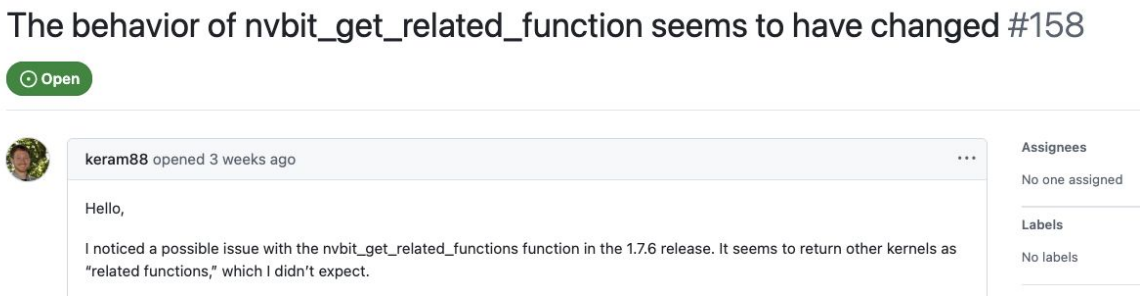
Framework-Recommended Solutions Don't Work Always

- Experts have told me how they deal with show-stopper exceptions
  - Try a bag of tricks (similar to those present in PyTorch - listed shortly)
- Thousands of reports that go like this



The uphill battles we are waging… and path ahead

- We are dependent on NVIDIA's binary instrumentation framework
  - NVBit
  - Our work "merely" extends NVBit
- NVBit issues keep popping up
  - We are at present waiting for a fix (w/o which we can't use their latest release)
    - earlier releases have issues
- Without NVBit, we have (and the community has) **no tools whatsoever**
  - **Other issues: which instructions does nvcc generate?**
    - **and have we covered it?**



Exceptions are a menace in ML – NixNan helped below!

SRU

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

- Throws NaN
- User unable to diagnose

  We traced it to Misunderstood Pytorch allocation ; does not clear Memory as the user thought

Julia Bug

https://discourse.julialang.org/t/neuralpde-on-gpu-throws-nans-when-i-use-a-source-term-elevated-to-some-power/114048

- User has NaN issue during training in Julia that launches to GPUs; claims they have a fix

  We found the NaN still coming, (albeit much later)

BatchNorm1d

https://github.com/pytorch/pytorch/issues/162489

- User posted Sep’25
- Finds CPU / GPU differences wrt NaNs in BatchNorm1d

  GPT finds a fix when fed our Tool’s traces.

  "need more precision"

  This is a "newbie bug"

This table lists the PyTorch issues we have examined using NixNan

NixNan is unique in that it can shed more light

At present, certain NVBit issues are making NixNan less effective

Original developers may have benefited a lot more, than us!

Selected Examples Tried	Type of Issue	Conclusions on Issue - following NixNan run
Karpathy's simple LLM	NaN during training	Does not affect training
Lossy compressor PyBlaz	NaN during compression	Does not affect operation
Issue 125674	Nan in grad. of scaled dotprod attention	Can produce traces that may help customer
Issue 156020	Value too high	Trace suggests NaN written into memory
Issue 156707	Issue with MPS	Tracing on GPUs possible
Issue 152737	NaN prop in Conv1d	Two NaN Stores into memory observed
Issue 157272	Reciprocal NaN issue	Bug in CPU; ran GPU code for comparison (traces)
Issue 159333	Foreach.copy bug	NaN store into memory present. Traces informative?
Issue 68425	Can run min instance	NaN due to predicated execution known
Issue 160016	PyTorch's own anomaly detection can be "porous" (does not report in the cases shown here)	Reproduces issues pointed out
BioMistral	None; runs	Lots of NaN/INF observed; harmless?

The Reality : In systems such as PyTorch, must use Tools such as NixNan **during Kernel Design-Time!**

(Else, it is too hopeless to find bugs later.)

Here are a few summary observations wrt PyTorch, to date

Easy bugs : can often spot and use (with manual effort)	Short and Tricky : Basic Logic of Kernel can be broken!	Too Large to Debug : Don't wait till this time!
BatchNorm1d : Switch from FP32 to FP64 with better accumulation	#160876: silu(x) = x * sigmoid(x) returns NaN; x = -inf (used to be 0)	Many issues are “too large to be posted” – very little can be done
Evident that non-experts suffer. Still gotta help them (non-uppity)	Naive implementation: -inf * 0 L’Hopital’s rule: 0	Solution: Avoid inserting bugs into simpler APIs by having unit-testing

For effective root-causing, we need to bridge across levels of calls

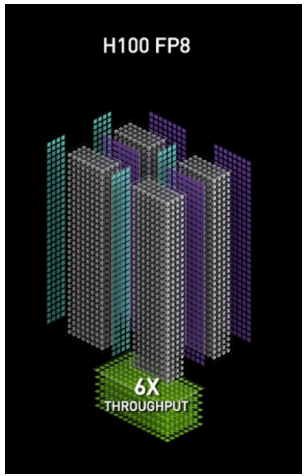
GPTs may help bridge **PyTorch** calls, **GPU** calls, and **Exception-annotated SASS**

Folder	NameOfKernel	FunctionsRun	
medgemma	void at::native::vectorized_elementwise_kernel	{}	[]
medgemma	void at::native::elementwise_kernel	{}	[]
medgemma	void at::native::reduce_kernel	{}	[]
medgemma	void at_cuda_detail::cub::DeviceScanInitKernel	{}	[]
medgemma	void at_cuda_detail::cub::DeviceScanKernel	{}	[]
medgemma	void at::native::unrolled_elementwise_kernel	{}	[]
medgemma	void at::native::	{}	[]
medgemma	void gemmk1_kernel	{at::native::vectorized_elementwise_kernel}	Details 1
medgemma	void cutlass::Kernel2	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2}	Details 2
medgemma	void pytorch_flash::flash_fwd_kernel	{cutlass::Kernel2, pytorch_flash::flash_fwd_kernel}	Details 3
medgemma	void cublasLt::splitKreduce_kernel	{}	[]
medgemma	ampere_s16816gemm_bf16_256x128_ldg8_stages_32x3_tn	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2}	Details 4
medgemma	ampere_bf16_s16816gemm_bf16_64x64_ldg8_f2f_stages_64x6_tn	{at::native::vectorized_elementwise_kernel, cutlass::Kernel2, pytorch_flash::flash_fwd_kernel}	Details 5

Note that all line-numbers are 0 (feature "vanished" in one NVBit release…)

```
#nixnan: error [NaN] detected in operand 1 of instruction @P0 FADD.FTZ R6, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction @P0 FADD.FTZ R6, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction MUFU.RCP R7, R9 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction MUFU.RCP R7, R9 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FADD R0, R0, -1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction FADD R0, R0, -1 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 2 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 0 of instruction FMUL R10, R6, R7 ; in function archimedes_kernel_fp16 at line 0 of type f32
#nixnan: error [NaN] detected in operand 1 of instruction FFMA R7, R0, R0, 1 ; in function archimedes_kernel_fp16 at line 0 of type f32
```

Deep-Dive : Tensor-Core Support of NixNan



> LD_PRELOAD = nixnan.so
> Pre-Compiled Application Binary

Intercept all "Kernel Launch" CUDA Driver API Calls

For Each FP instruction :

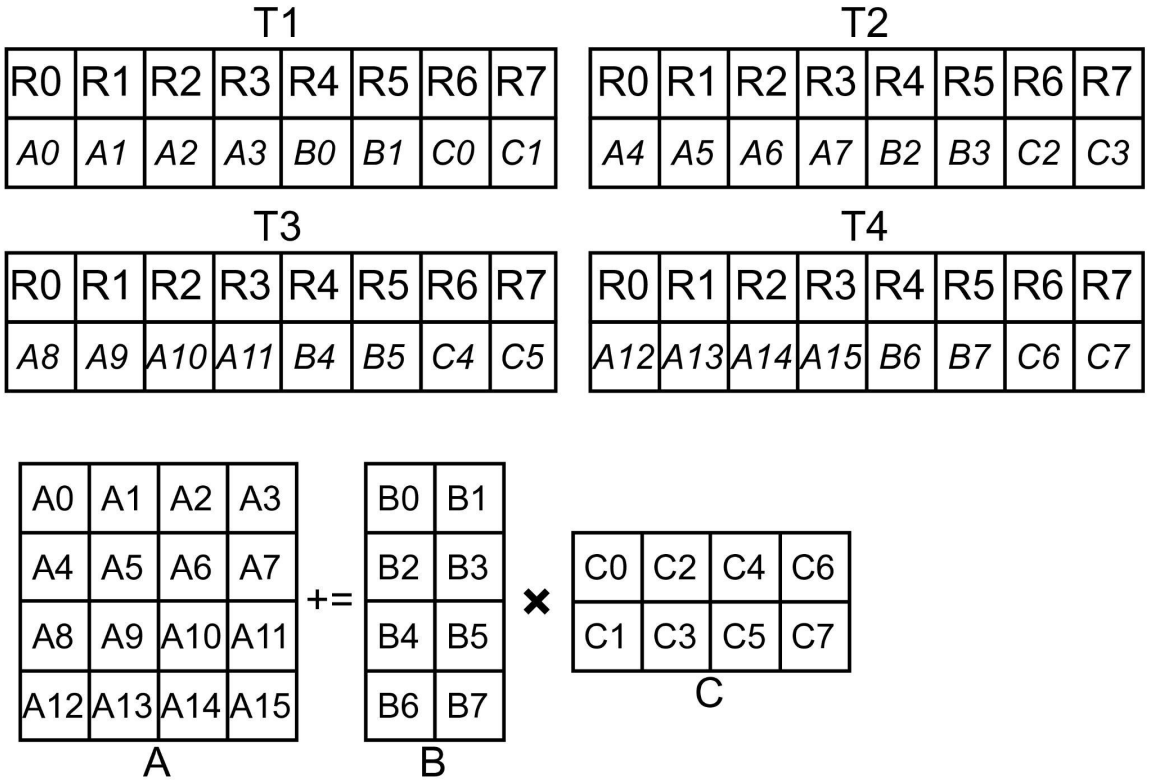
Inject_instrumentation();

CUDA Driver APIs

```
----- Test NaN: A[0,0]=inf, B[0,0]=1, C[0,0]=-inf -----
A[0,0] = inf (0x7c00), B[0,0] = 1.000000 (0x3c00), C[0,0] = -inf (0xfc00)
Computing D = A * B + C with Tensor Cores...
#nixnan: error [NaN,infinity] detected in instruction HMMA.16816.F16 R20, R4.reuse, R16, RZ ;
#nixnan: error [NaN] detected in instruction HMMA.16816.F16 R22, R4, R18, RZ ; in function WMM
D[0,0]=nan (0x7fff)
```

Tensor core register layout

Each thread holds part of the matrices in its registers

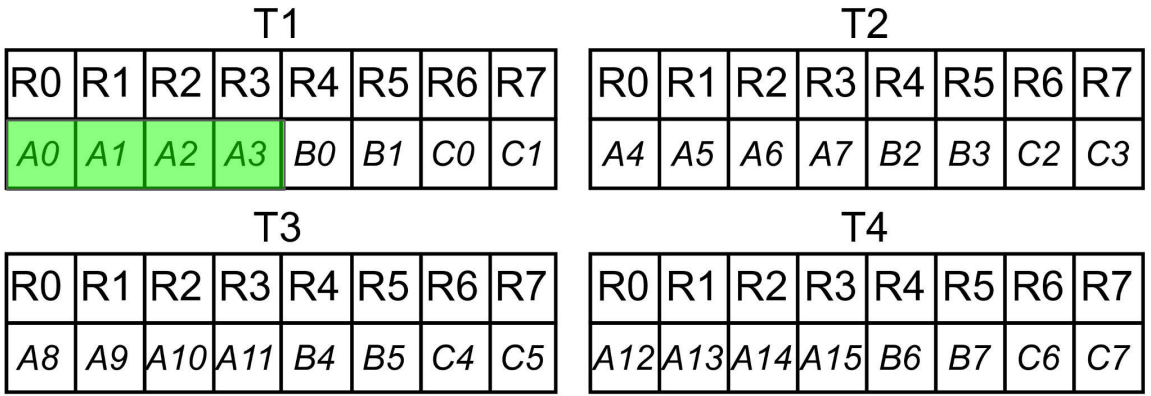


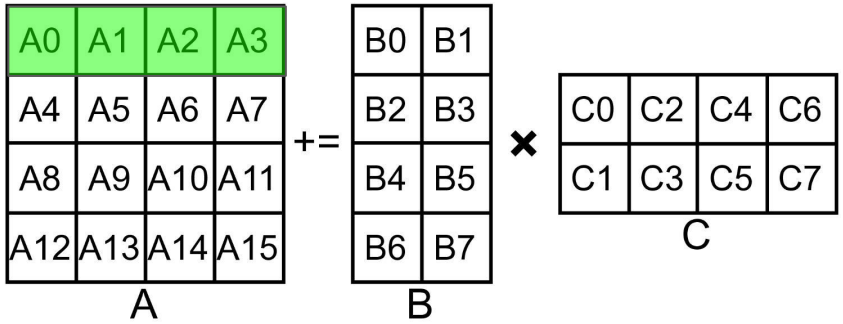
Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A



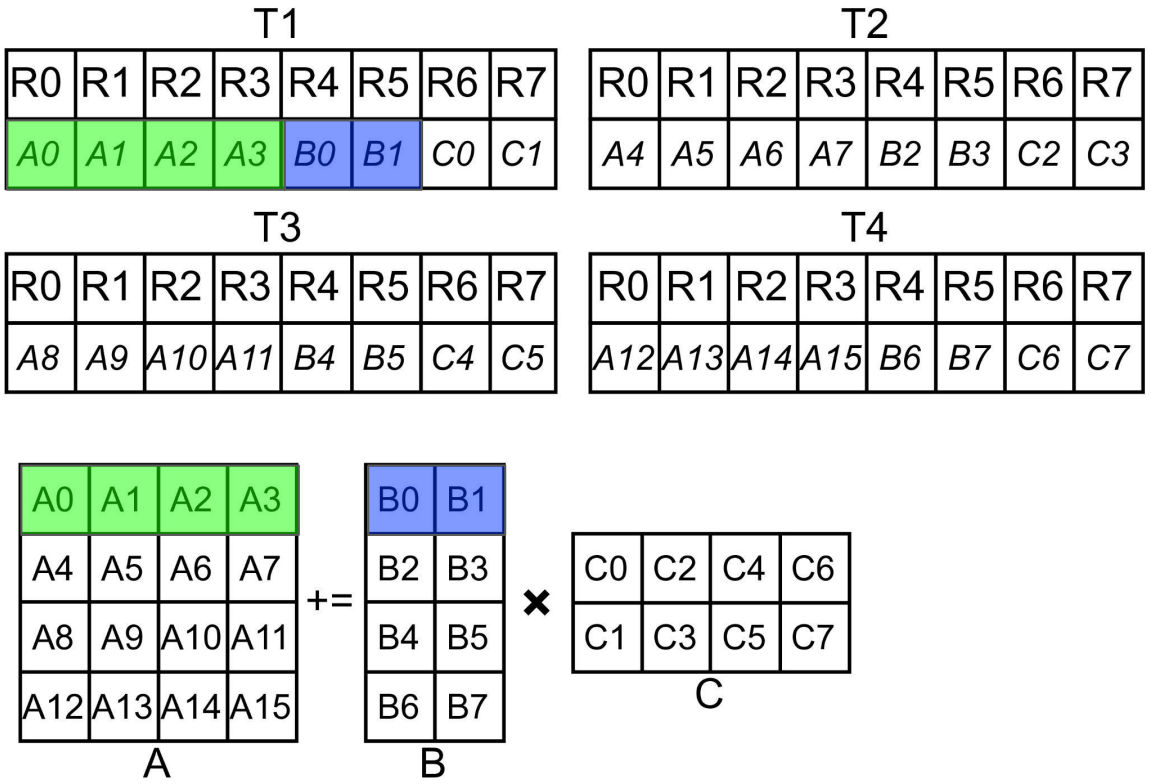


Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A
- The first two elements of B

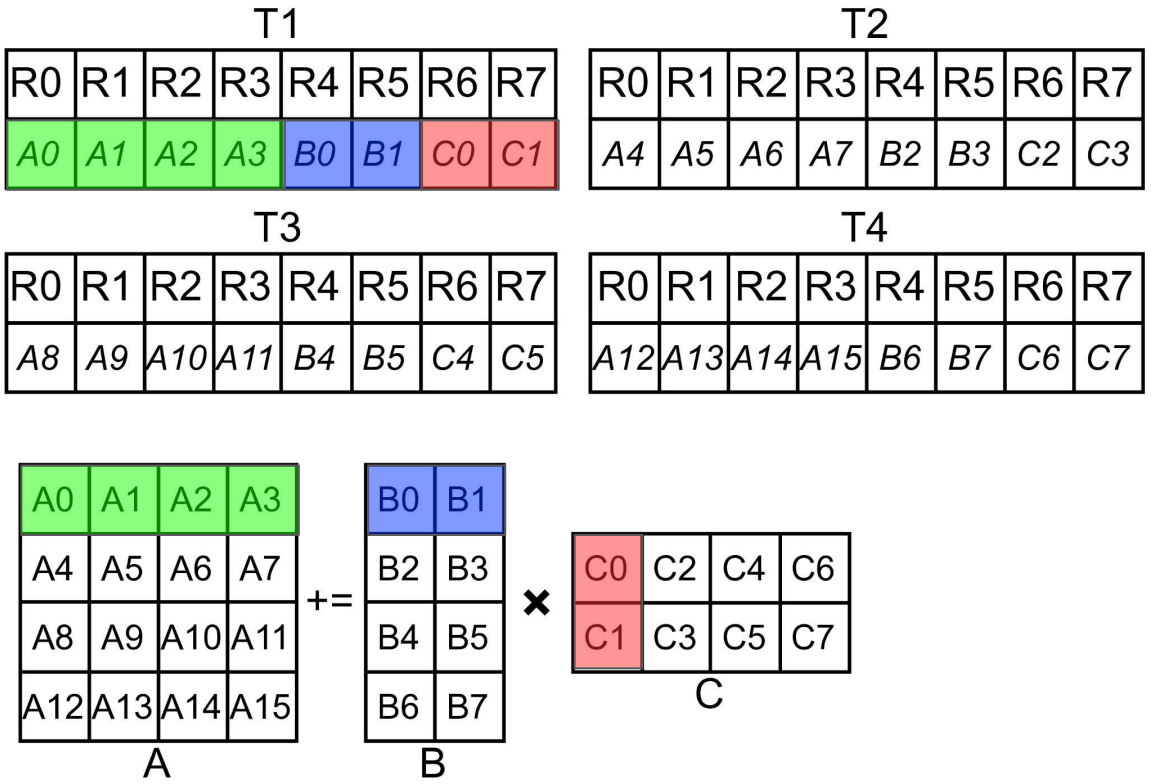


Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A
- The first two elements of B
- The first two elements of C



Concluding Remarks : Let's visit all the topics discussed

- NixNan helps reveal exceptions via Binary Instrumentation
  - Helpful for closed-source libraries (many are)

- Knowing which exceptions matter is not straightforward
  - Easier with HPC codes
    - NaN / INF Exceptions almost always are bad
  - Not well-understood with ML codes
    - NaN / INF Exceptions seem to fly around with abandon
    - Can "jiggle" ML models and see exceptions change (as on Slide 32)
      - This may give insights for improvement / debugging
      - Users are suffering from exceptions (e.g., PyTorch Open issues)

- Limitations, Need for Community + Manufacturer Cooperation
  - GPUs are not well-documented
    - Reverse-engineering GPU behaviors was necessary
      - Not easy to scale, considering the rate of arrival of new GPUs
  - Help from GPU manufacturers can greatly help advance FP debugging

Agenda

- 08:30 : Introduction
- 08:35 : FPChecker
- 09:35 : NixNan (Part-1)
- 09:55 : Fill Survey
- 10:00 : Coffee Break
- 10:30 : Brief Recap
- 10:35 : NixNan (Part-2)
- **11:05 : FloatGuard**
- 11:50 : Closing Remarks
- 11:55 : Fill Survey