Array Programming on GPUs : Challenges and Opportunities

An Experience Report

Xinyi Li, Mark Baranowski, Harvey Dam, Ganesh Gopalakrishnan

Kahlert School of Computing, University of Utah, Salt Lake City, UT
https://www.cs.utah.edu/~ganesh

Funded by
Department of Energy, Office of Science, Advanced Scientific Computing Research
and the National Science Foundation

KAHLERT SCHOOL OF COMPUTING
THE UNIVERSITY OF UTAH

U.S. DEPARTMENT OF ENERGY

NSF

THE UNIVERSITY OF UTAH

Org

- GPUs rule
- Energy drives everything
- 50% of energy for math
- Number systems help curb energy / bug tradeoff
  - But what's happening in today's ML-driven world is an explosion of ill-specified systems
  - ML hardware has driven out traditional well-specified FP hardware, resulting in bugs
- The bugs are: FP exceptions; Ill-specified HW; and a lack of formal specs + managing their surfeit
- Exceptions first
  - Pytorch evidence abounds - solutions neither air-tight nor efficient - present table
  - What is a gold-standard for exception handling
  - We know of only one (very good definition) : consistent exception handling
  - But that is not happening
    - SRU example
    - Diagonal boosting in HPC example
    - Closed-source libraries without specs - certainly can't happen under that conditon
  - We need to begin concerted efforts
    - Prior GPU-FPX contribution recap
    - Now show how easily we can "go to work with GPU-FPX" and reveal the "rats running around"
    - **Show cartoon**
    - Then show the 3 examples : SRU, Compressors, GPT
  - New work
    - Support for many number systems, Tensor Core → Community infra
- Ill-specified HW next
  - Eye-opening examples → NFM examples + formal specs under construction → Community infra
- Lack of formal specs last
  - Create many
  - Then
    - Either neutralize to one common denominator, OR
    - See how much one can do to live under a different regime → Work in progress is to make HPC like ML to adapt to new order
- Conclusions
  - Exceptions are an unrecognized threat and you can engage as a PL crowd (start with dyn methods and end with static analysis)
  - Manage specs (develop specs, tests, and have a way for the community to grow this) – say like how the mem model people are doing around litmus tests
  - Grow and manage specs - needs more than numerics-level litmus : need contextual analysis and adjustments

context:

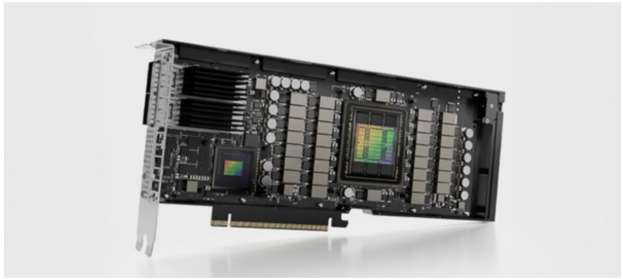
GPUs Rule

They are key to energy-efficient computing: GPT to watches

GPUs : The Ama$$ing Computing-Engine : H100 (Hopper, source)

H100: The Foundational Workhorse of Modern AI

The **NVIDIA H100 Tensor Core GPU** remains one of the most versatile and widely deployed GPUs for AI training and Inference. Available in both **PCIe** and **SXM (via HGX systems)**, it offers a solid balance of performance, compatibility, and cost.



NVIDIA H100 PCIe Form Factor

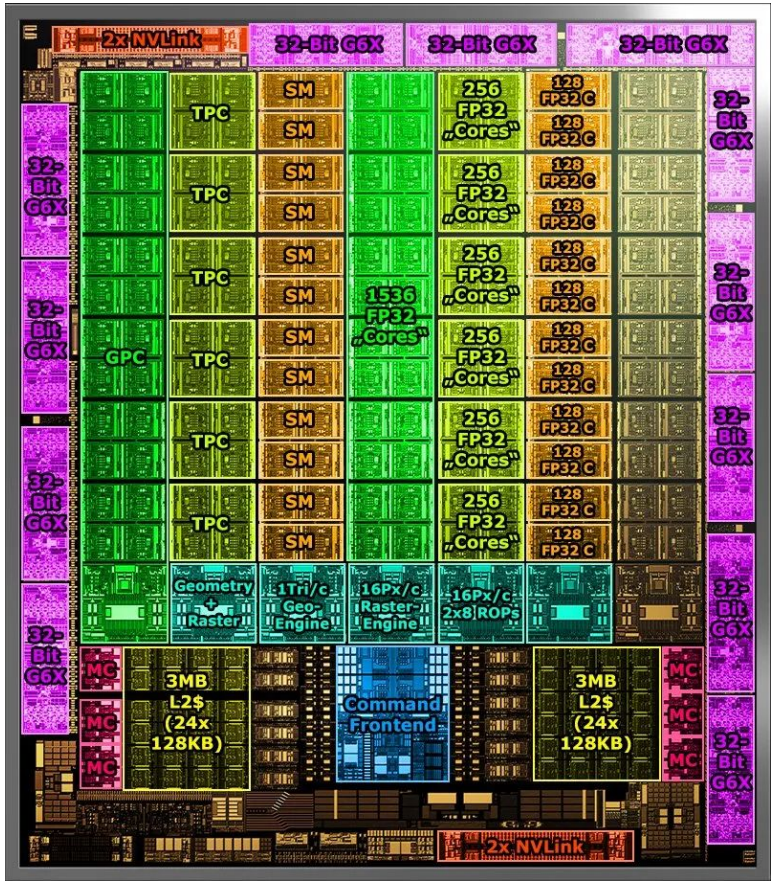
- **Memory**: 80GB HBM2e
- **Form Factors**: PCIe and SXM
- **Peak FP8 Performance**: Up to 4.9 PFLOPs (SXM)
- **Arc's Availability**:
  - **HGX Systems (SXM)**: Starting at **$215,000 USD** – View Servers
  - **PCIe Systems**: Contact Sales
  - **Reserved Cloud (HGX H100)**: Arc's Reserved Cloud

$215K

Why Choose H100?

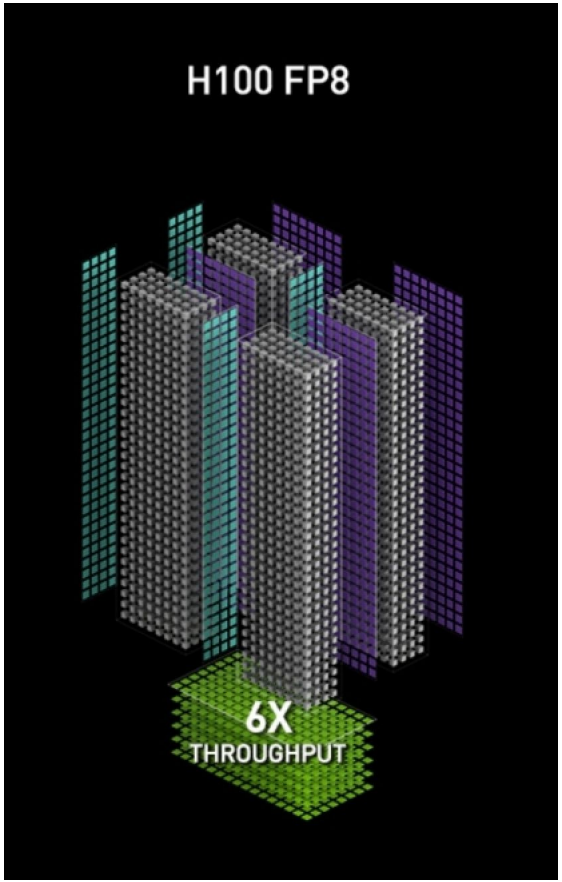
- **PCIe**: Ideal for scalable Inference and modular deployments. Broad compatibility and lower entry costs make it great for startups and production-ready GenAI teams.
- **SXM (HGX)**: Designed for multi-GPU training. With **NVLink and NVSwitch**, up to 8 GPUs can share memory at high bandwidth, and HGX nodes can be networked for large-scale training.

GPUs : The Amazing Computing-Engine of This Century (image source)



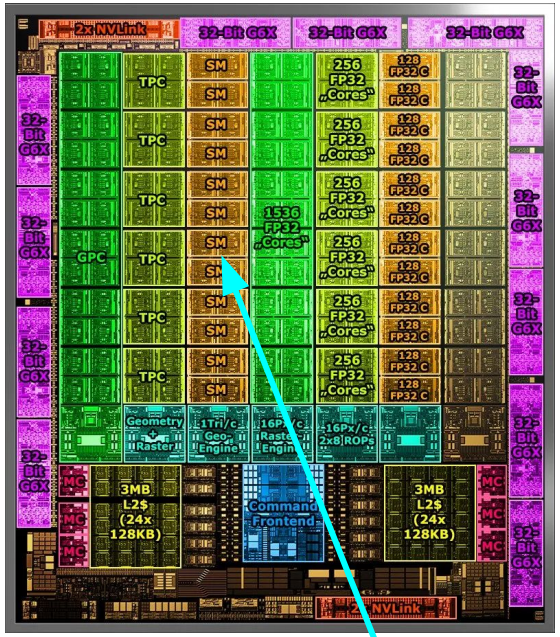
- An affordable GPU: Nvidia GA102
  - Ampere architecture.
    - GeForce RTX 30 series cards,
      - 3090, 3080 Ti, 3080, etc.
- 128 CUDA cores, 4 Tensor cores,
- About 350W
- About $800 per card bought on eBay

GPU Tensor Cores : True Workhorses in ML and rising in HPC



- Google TPUs
  - "Only Tensor Cores"
- NVIDIA GPUs
  - Rising support
  - Growing silicon area
- Crucial for Deep ML
- Growing interest in HPC

GPUs : The Amazing Computing-Engine of This Century (image source)



CUDA C Code

```
int i = threadIdx.x;
c[i] = a[i] * b[i];
```

nvcc

PTX Code

```
ld.global.f32 , mul.f32 , st.global.f32
```

ptxas

SASS Code

```
/*0008*/ LDG.E.SYS R4, [R1];
/*0010*/ FMUL R6, R4, R5;
/*0018*/ STG.E.SYS [R3], R6;
```

Warp

CUDA C Code

```
wmma::mma_sync(c_frag, a_frag, b_frag, d_frag);
```

nvcc

PTX Code

```
wmma.mma.sync.aligned.m16n16k16.row.row.f32.f16.f16.f32
```

ptxas

SASS Code

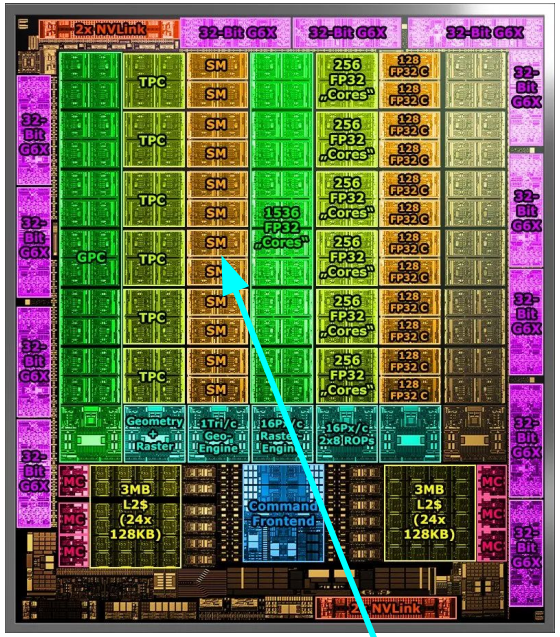
```
HMMA.1688.F32.F16.F16.F32
```

Warp

Single "small matrix Multiplication" call - done in Tensore Core hardware

The SMs contain
Hardware Warp Schedulers
The Warps carry
SIMT-Core instruction "work" and
Tensor-Core Instruction "work"

GPUs : The Amazing Computing-Engine of This Century (image source)



CUDA C Code

```
int i = threadIdx.x;
c[i] = a[i] * b[i];
```

nvcc

PTX Code

```
ld.global.f32 , mul.f32 , st.global.f32
```

ptxas

SASS Code

```
/*0008*/ LDG.E.SYS R4, [R1];
/*0010*/ FMUL R6, R4, R5;
/*0018*/ STG.E.SYS [R3], R6;
```

Warp

CUDA C Code

```
wmma::mma_sync(c_frag, a_frag, b_frag, d_frag);
```

nvcc

PTX Code

```
wmma.mma.sync.aligned.m16n16k16.row.row.f32.f16.f16.f32
```

ptxas

SASS Code

```
HMMA.1688.F32.F16.F16.F32
```

Warp

The SMs contain
Hardware Warp Schedulers
The Warps carry
SIMT-Core instruction "work" and
Tensor-Core Instruction "work"

We Instrument SASS, as that's often what we can access + that's the most honest !!

Single "small matrix Multiplication" call - done in Tensore Core hardware

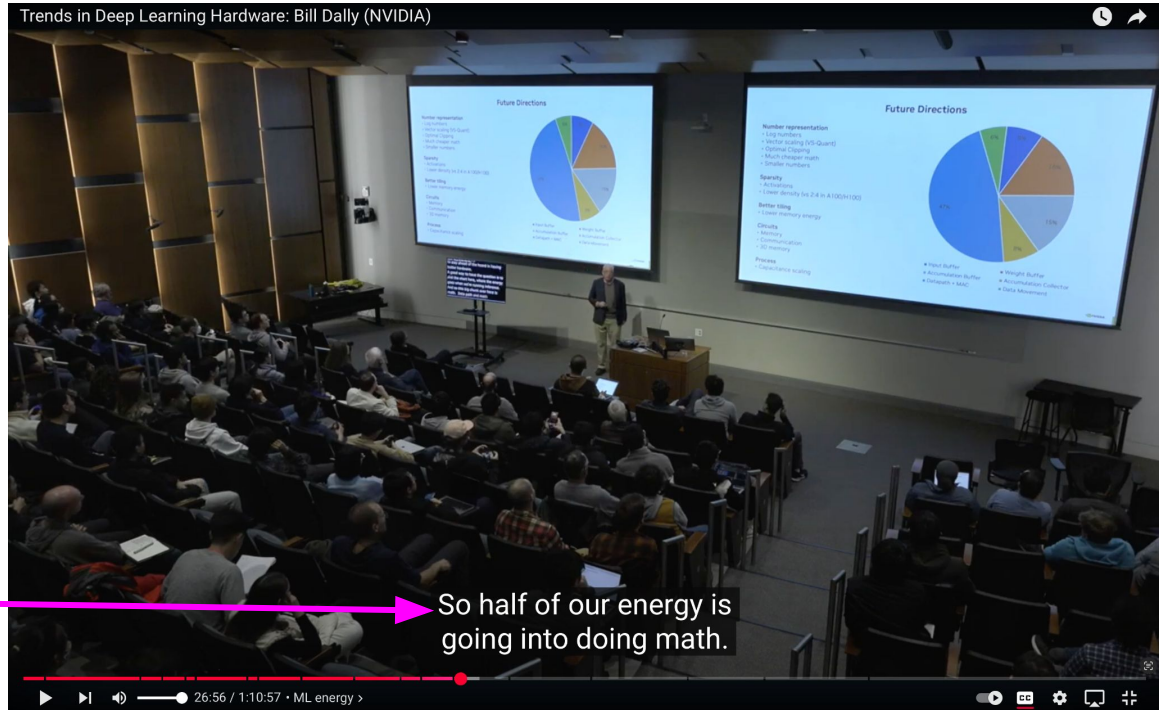
Energy Drives Decisions

Half the energy goes into doing math (for inference)

GPU Array Programming Complexities Stem from Measures to Save Energy!

- Why?
  - Talk-1
    - at UW
  - Talk-2
    - at HotChips'23

- by Dr. Bill Dally, SVP
  - Chief Scientist @ NVIDIA



Must move fewer bits.
This causes
Accuracy and Dynamic Range
to be reduced , leading to bugs

Future Directions

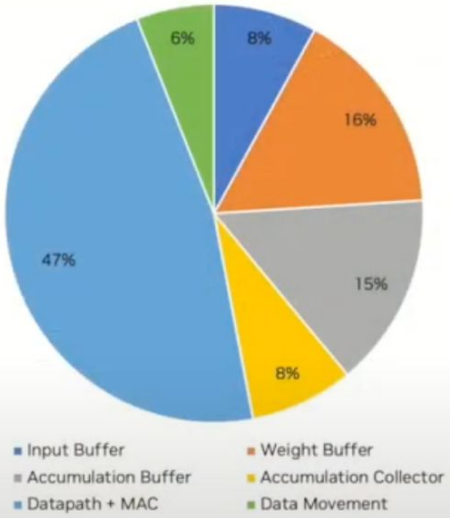
Number representation
- Log numbers
- Vector scaling (VS-Quant)
- Optimal Clipping
- Much cheaper math
- Smaller numbers

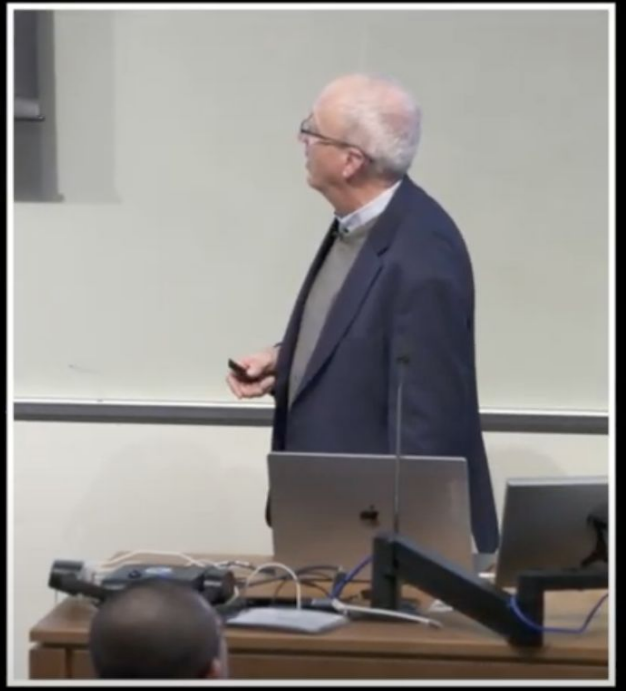
Sparsity
- Activations
- Lower density (vs 2:4 in A100/H100)

Better tiling
- Lower memory energy

Circuits
- Memory
- Communication
- 3D memory

Process
- Capacitance scaling

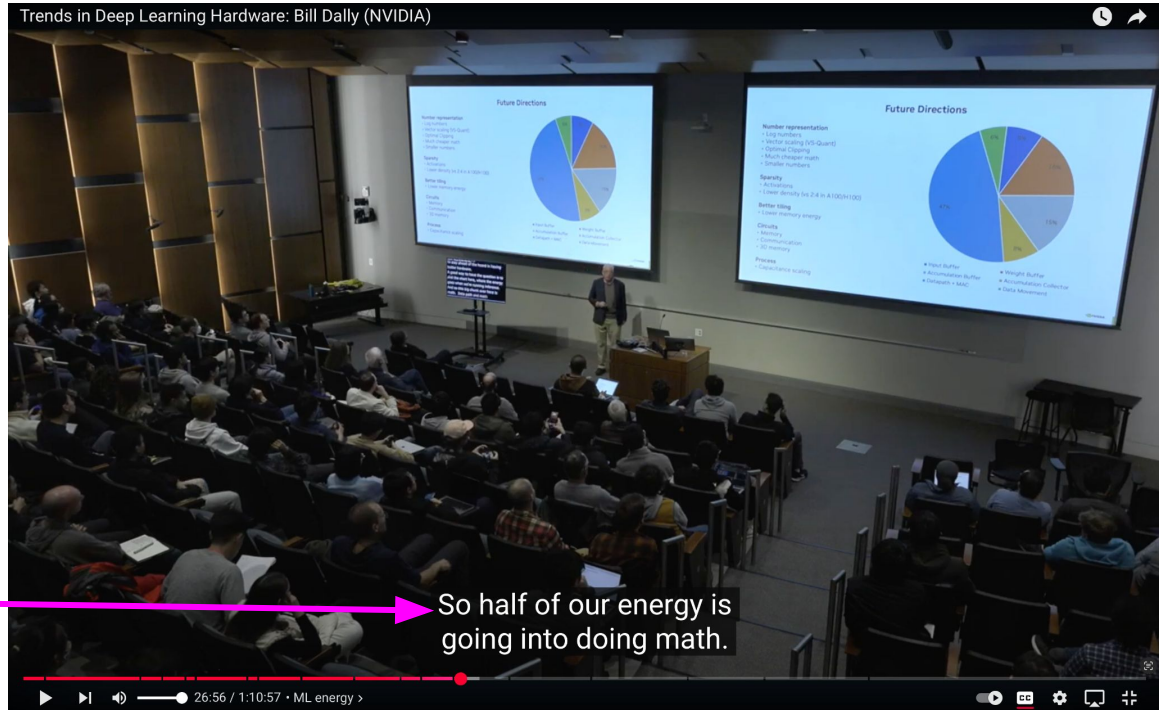




when we're doing inference.

GPU Array Programming Complexities Stem from Measures to Save Energy!

- Why?
  - Talk-1
    - at UW
  - Talk-2
    - at HotChips'23
- by Dr. Bill Dally, SVP
  - Chief Scientist @ NVIDIA



Must move fewer bits.
Unfortunately, the types of hardware created by AI companies are ill-specified.

Number Systems Help Trade Energy and Bugs

The plethora of choices of formats is evidence

Background: Dynamic Range and Error for various number formats

No.	Fmt	Prec	Exp	Sm. norm.	Lrgst norm.	Unit rnd u	ULP @ e_max	2's comp. bits	
1	FP16	11	5	6.10e-5	6.55e4	4.88e-4	2^5	30	
2	BF16	8	8	1.18e-38	3.39e38	3.91e-3	**2^120**	254	
3	TF32	11	8	1.18e-38	3.40e38	4.88e-4	2^117	254	
4	FP32	24	8	1.18e-38	3.40e38	5.96e-8	**2^104**	254	
5	FP64	53	11	2.23e-308	1.80e308	1.11e-16	2^971	2046	
6	E4M3	4	4	1.56e-2	2.48e2	6.25e-2	2^4	14	(fwd, for activations.)
7	E5M2	3	5	6.10e-5	6.14e4	1.25e-1	2^13	30	(bkwd, for grad.)

256 values

The future depends on moving fewer bits while managing the errors caused by this variety of number systems.

We discuss 3 types of bugs / needs

Floating-Point Numbers : A compromise, with many good properties, that can span a large range with a small number of bits

No.	Fmt	Prec	Exp	Sm. norm.	Lrgst norm.	Unit rnd u	ULP @ e_max	2-comp bits
1	FP16	11	5	6.10e-5	6.55e4	4.88e-4	2^5	30
2	BF16	8	8	1.18e-38	3.39e38	3.91e-3	2^120	254
3	TF32	11	8	1.18e-38	3.40e38	4.88e-4	2^117	254
4	FP32	24	8	1.18e-38	3.40e38	5.96e-8	2^104	254
5	FP64	53	11	2.23e-308	1.80e308	1.11e-16	2^971	2046

- Unit Roundoff ("u") : One unit of rounding (assuming RTN) : rnd to nearest ties to even
- u = half of the distance from 1.0 to the next representable number
- Absolute errors are discussed below. Its value is NumberRepresented * u
  - FP32
    - Error measuring earth-to-Moon distance of 384,400 km : 22.9 meters
    - Error measuring height of Human (172 cm) : 0.000103 mm
  - BF16
    - Moon: 1500 km
    - Human: 6.7 mm
  - FP16
    - Moon: Can't represent - INF
    - Human: 0.84 mm

Gist of talk

-
- **What can we do for the PL Community (covering ML and HPC)** so that
  - (contribution 1) **: They can detect show-stopper bugs early ?**
  - (contribution 2) **: Explicate opaque hardware ?**
  - (contribution 3) : **Develop formal specs … even synthesize correct code?**

Problem Addressed: GPU variety grows w/o Tools Support

- **Growth in GPU variety, leading to:**
  - Growth in **two primary aspects** of GPU programming that cause bugs:
    - Parallelism varieties (not emphasized in this talk)
    - Number systems (emphasized here)
- **Debugging tools haven't kept up**
  - **Inability to reason about the exhibited behaviors**
- **Specs don't exist**
  - Must be discovered through testing
- **Having many specs is also undesirable**
  - Which specs can support a given application?

Three Types of "Bugs"

(1) **FP exceptions** (2) **Ill-specified HW** (3) **Lack of formal specs + Managing their surfeit**

Why these three types of "bugs"

- Measures to save energy has resulted in
  - Executions that often generate **FP Exceptions** which arrest overall progress
  - (Tensor Core…) Hardware that is **Weird** as well as **Sparingly Specified**
  - (Overall ISA has…) **No Formal Descriptions**

- ML can survive this level of "mystery" … **(or, so it seems – do we have a choice?)**
  - Libraries provide API to hardware
- **Unfortunately**
  - **ML's success has driven out** standard and well-specified hardware

- **This leads to three challenges that we discuss in this presentation**

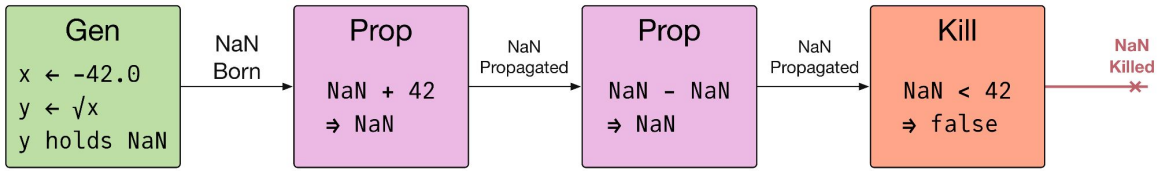
FP Exceptions

- FP32: (1.9×10^{19})^2 → INF … (in E4M3, 16^2 = INF)
  Squaring is widely used (e.g., dot-products)

- INF and NaN are most serious; also they inter-convert INF / INF = NaN

- #define MAX(x,y) ((x >= y) ? x : y) is incorrect
  hint : consider x == NaN or y == NaN

Exception flow, in general



Classical (CPU-era) Solution to Exceptions

- Have the hardware generate traps
  - When trap-handling is enabled, handle it in software

- **NVIDIA GPUs do not generate traps!**

- This approach is incongruous with GPU computing
  - Imagine a million threads generating traps!
    - AMD GPUs do "something in-between"

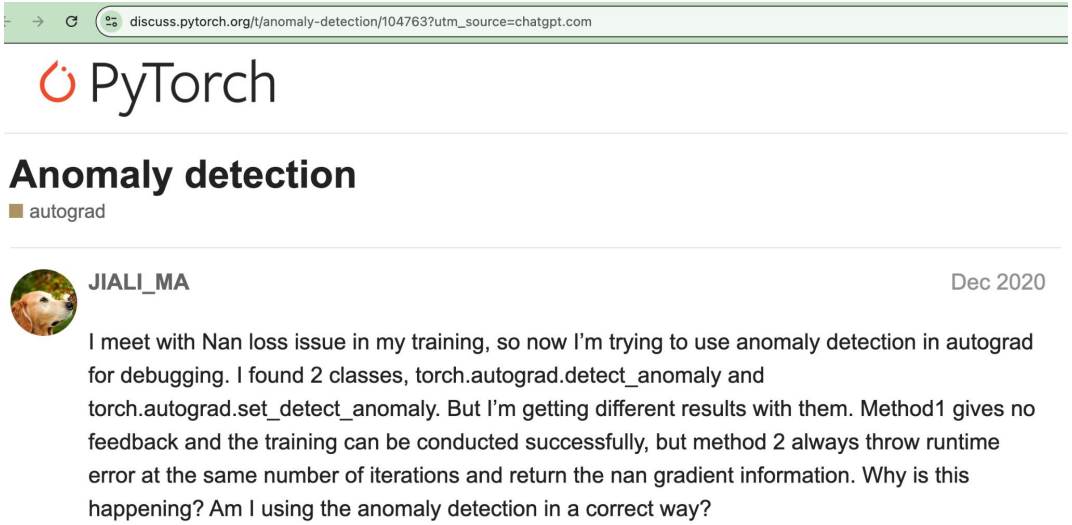
In ML, Frameworks such as PyTorch Silently "Fix" Exceptions, but this can be Slow / Imperfect (believable ChatGPT summary below)

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	

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



In ML, Frameworks such as PyTorch Silently "Fix" Exceptions, but this can be Slow / Imperfect

3 Typical user-level counter-measures in PyTorch

Category	Common tricks
Clamp before trouble	`logits = torch.clamp(logits, -15, 15)` before soft-max
Gradient clipping	`torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm)`
Loss-scale tuning	Lower `init_scale` in `GradScaler(init_scale=2**14)`
Replace non-finites	`tensor = torch.nan_to_num(tensor, nan=0.0, posinf=1e4, neginf=-1e4)`
Batch filtering	Drop or re-initialise batches whose loss isn't finite
Debug builds	Keep `detect_anomaly()` on for the first few epochs only

Current Exception Handling in PyTorch / ML : Sometimes an Overkill

- Key debugging tools in PyTorch: `isnan` and `isfinite`

```
>>> x
tensor([0., 1., nan, inf])
>>> x.isnan()
tensor([False, False,  True, False])
>>> x.isfinite()
tensor([ True,  True, False, False])
```

- Typical safeguards
  - `torch.nn.utils.clip_grad_norm_`
  - Add a small number to potentially-zero denominators.
    - These are built into many modules, e.g. `Adam` (see the "+ ϵ")
- Current Approach Taken in ML
  - Fix NaN / INF exceptions and continue, saving checkpoints
  - **If we can't continue, restart from a "good checkpoint"**
- While we can put exception detection everywhere, it will make it VERY slow
  - **3x slowdown** reported when using `torch.autograd.detect_anomaly(check_nan=True)`
- **Need a solid set of concepts to do formal PL work !!**

Current Exception Handling in HPC : Very Situation-Specific

- No real widely used approach
  - Table of results from our paper GPU-FPX below
- Need Binary-Instrumentation Tools (GPU-FPX) to Peek into Codes
  - Necessary (closed-source libraries) + Accurate (compiler opt. effects)

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

Expert-helped
Black Magic !!

FP Exceptions

How to define their "proper handling"?

We've found only one clear definition … by Demmel

What is Consistent / Sound Exception Handling? How close are we to it ?

Computer Science > Mathematical Software

[Submitted on 19 Jul 2022]

Proposed Consistent Exception Handling for the BLAS and LAPACK

James Demmel, Jack Dongarra, Mark Gates, Greg Henry, Julien Langou, Xiaoye Li, Piotr Luszczek, Weslley Pereira, Jason Riedy, Cindy Rubio-González

Numerical exceptions, which may be caused by overflow, operations like division by 0 or sqrt(−1), or convergence failures, are unavoidable in many cases, in particular when software is used on unforeseen and difficult inputs. As more aspects of society become automated, e.g., self-driving cars, health monitors, and cyber-physical systems more generally, it is becoming increasingly important to design software that is resilient to exceptions, and that responds to them in a consistent way. Consistency is needed to allow users to build higher-level software that is also resilient and consistent (and so on recursively). In this paper we explore the design space of consistent exception handling for the widely used BLAS and LAPACK linear algebra libraries, pointing out a variety of instances of inconsistent exception handling in the current versions, and propose a new design that balances consistency, complexity, ease of use, and performance. Some compromises are needed, because there are preexisting inconsistencies that are outside our control, including in or between existing vendor BLAS implementations, different programming languages, and even compilers for the same programming language. And user requests from our surveys are quite diverse. We also propose our design as a possible model for other numerical software, and welcome comments on our design choices.

When an Overflow, Divide-by-Zero, or Invalid exception occurs, a **sound exception-handling policy** notifies users by either…

1. The presence of EVs in the output
2. Some library-specific mechanism triggered by checking for their presence at some point during the execution.

This is the policy adopted by the reference LAPACK/BLAS implementations [1].

-> How can we test this?

[1] J. Demmel, et al. "Proposed Consistent Exception Handling for the BLAS and LAPACK".

cases, in particular when software is used on unforeseen and difficult inputs. As more aspects of society become automated, e.g., self-driving cars, health monitors, and cyber-physical systems more generally, it is becoming increasingly important to design software that is resilient to exceptions,

Sometimes it takes an event like the crash of the Ariane 5 rocket [1], a naval propulsion failure [2], or a crash in a robotic car race [3] to make people aware of the importance of handling exceptions correctly in numerical software [4, 5]! As applications like self-driving cars, health monitors, and cyber-physical systems more generally be-

EXCVATE: Spoofing Exceptions and Solving Constraints to Test Exception Handling in Numerical Libraries

Jackson Vanover*, James Demmel†, Xiaoye Sherry Li‡, Cindy Rubio-González*

*University of California, Davis
†University of California, Berkeley
‡Lawrence Berkeley National Laboratory

ARITH
2025
Best Paper

We are not even close to consistent exception handling!

- The SRU Example
- The example that needed "diagonal-boosting"
- Two data-compressors illustrated in our paper
- Even the "Baby GPT" of Andrej Karpathy after I ported it to GPU

(image by ChatGPT with prompts HERE)

GPU-FPX (for SIMT-only)
and **nixnan** (for SIMT and Tensor Cores) **are easy to use and tell you a lot** !! and you can get them !!
nixnan is being released as part of this workshop!

(... could be slow, but…)

Ease of use: tracing Karpathy's Baby GPT learning a DFA's n-gram language…

"Classical" run (w/o our tools)

```
time python KarpathyNGramDFA1.py
...
NOW LOSS DECREASES
...
number of parameters: 12656
0 0.6365983486175537
1 0.601229727268219
2 0.5856719613075256
3 0.5780848264694214
...
46 0.3900260329246521
47 0.3890194892883301
48 0.3880999982357025
49 0.3872622549533844
...
real    0m2.409s
user    0m2.423s
sys 0m1.645s
```

Instrumented run (with our tools)

```
time LD_PRELOAD=./analyzer.so python KarpathyNGramDFA1.py
```

```
=======================
time LD_PRELOAD=./analyzer.so python KarpathyNGramDFA1.py
...
#GPU-FPX: Instrument all kernels.
...
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::vectorized_layer_norm_kernel] ...
Running #GPU-FPX: kernel [void gemmSN_TN_kernel]
...
Running #GPU-FPX: kernel [void (anonymous namespace)::softmax_warp_forward] ...
#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [void (anonymous namespace)::softmax_warp_forw
:0 Instruction: FSEL R11, R11, R2, !P0 ; We have 3 registers in total. Register 0 is INF. Register 1 is INF. Register 2 is INF.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction  @ /unknown_path in [void (anonymous namespace)::softmax_warp_forwa
0 Instruction: FSEL R11, R11, R2, !P0 ; We have 3 registers in total. Register 0 is INF. Register 1 is INF. Register 2 is INF.

number of parameters: 12656
...
Running #GPU-FPX: kernel [void at::native::(anonymous namespace)::indexSelectLargeIndex] ...
Running #GPU-FPX: kernel [ampere_sgemm_32x128_tn] ...
Running #GPU-FPX: kernel [ampere_sgemm_128x32_tn] ...
Running #GPU-FPX: kernel [ampere_sgemm_32x32_sliced1x4_tn] ...
#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [void (anonymous namespace)::softmax_warp_forw
:0 Instruction: FSEL R5, R5, R10, !P0 ; We have 3 registers in total. Register 0 is INF. Register 1 is INF. Register 2 is INF.

...
NOW LOSS DECREASES
...
0 0.6365983486175537
1 0.601229727268219
2 0.5856719613075256
3 0.5780848264694214
...
46 0.3900260329246521
47 0.3890194892883301
48 0.3880999982357025
49 0.3872622549533844

#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [void at::native::reduce_kernel]:0 Instruction
EL R6, R6, R13, P0 ; We have 3 registers in total. Register 0 is INF. Register 1 is INF. Register 2 is VAL.
...
#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction  @ /unknown_path in [void gemmSN_TN_kernel]:0 Instruction: FSEL R1
12, R17, !P0 ; We have 3 registers in total. Register 0 is VAL. Register 1 is NaN. Register 2 is VAL.
...

real    4m59.125s
user    9m38.729s
sys 0m3.257s

=======================
```

We are not even close to consistent exception handling!

LD_PRELOAD=./analyzer.so python KarpathyNGramDFA2.py

#GPU-FPX-ANA SHARED REGISTER: Before executing the instruction @ /unknown_path in [void (anonymous namespace)::softmax_warp_forward]:0 Instruction: FFMA R9, R0, 1.4426950216293334961, -R9 ; We have 3 registers in total. Register 0 is VAL. Register 1 is INF. Register 2 is VAL.

#GPU-FPX-ANA SHARED REGISTER: After executing the instruction @ /unknown_path in [void (anonymous namespace)::softmax_warp_forward]:0 Instruction: FFMA R9, R0, 1.4426950216293334961, -R9 ; We have 3 registers in total. Register 0 is NaN. Register 1 is NaN. Register 2 is NaN.

- We uploaded the traces into ChatGPT
- See how ChatGPT "explains" the situation here, plus also recommends code fixes
  - The fixes did not eliminate the exceptions (promising direction we feel)

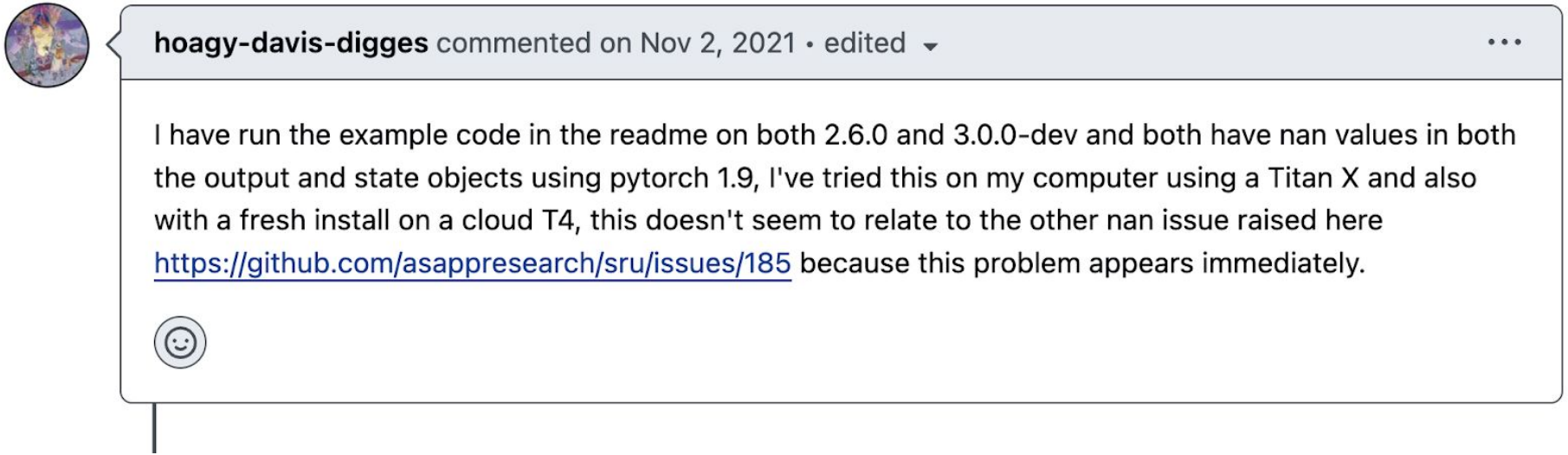
Results obtained when I casually uploaded the GPU-FPX traces to ChatGPT (Seems reasonable. Even if not true, suggests a future direction…)

What the GPU-FPX trace is telling us

Kernel (FPX label)	Where NaN/Inf shows up	Does it disappear before a store?	Likely root-cause	Severity
`softmax_warp_forward`	Large-magnitude logits start as **±Inf**; `FADD` turns `Inf − Inf` into **NaN**. Many of these NaNs are later fed to exponentials (`MUFU.EX2`) and multiplies	Some NaNs are clamped away by `FFMA.SAT`, but others survive several instructions and keep popping back up	Logits are not being shifted by the row-max (the classic "log-sum-exp trick") or they already contain Inf from an earlier layer	**Needs a fix** – risk of NaNs reaching the probability vector or its gradient
GEMM micro-kernels (`gemmSN_TN_kernel`, `ampere_sgemm_*`)	You see `FSEL … Register 1 is NaN` immediately followed by another `FSEL` whose destination register is **VAL**	Yes – the predicate keeps the good operand, so the NaN never leaves a register	One input tile contained NaN/Inf, but the predicate rejected it	**Innocuous** so long as the selected value isn't NaN
`reduce_kernel` (used by `torch.max`, `torch.mean`, etc.)	Alternating **INF ↔ VAL** operands in a chain of `FSEL` instructions	Often **INF** wins the comparison and gets written forward	Up-stream overflow (e.g. `exp()` on large numbers)	**Heed** – an Inf in a reduction will survive to host memory unless filtered
Misc. elementwise / layer-norm kernels	Short bursts of NaNs immediately neutralised by `FFMA.SAT`	Yes	These are saturating clamps or ReLU-like masks	**Benign**
`softmax_warp_backward`, `embedding_backward`, etc.	Only mirror the NaNs created in the forward pass; no new sources	—	Propagation of earlier problems	Depends on fixing the forward pass

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_sgemm_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_simple], 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 @ /unknown_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 small 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

Not clear how to fix it

```
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] ...
```

Use Analyzer

Speedup by enabling necessary kernels

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

```
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] ...
                                                    d_kernel_simple], NaN
                                                    simple]:0 [FP32]
                                                    d_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] ...
```

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 ; 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()

How GPU-FPX can provide added help in AI/ML and also HPC

- GPU-FPX inspects a lower level than `isnan` and `isfinite` but is slower.
  - You can see exceptions in temporary registers that `isnan` and `isfinite` won't see.
- Fused kernels such as FlashAttention can hide NaN and infinity
  - this can slow down training
  - or only cause an obvious error only MANY steps later.
- Even if you see exceptions in PyTorch…
  - GPU-FPX can pinpoint their first appearance
  - Useful for ambitious custom kernels.

- While we can put exception-suppression everywhere, it will make it VERY slow
  - **3x slowdown** reported when using `torch.autograd.detect_anomaly(check_nan=True)`

How a custom-made AI assistant might help explain exceptions

- Learn facts about code-base (Pytorch, Numerical Solver, …) each time a user obtains a trace (and also fixes it)
  - Conversations with experts (two I shall name privately) suggests that this is how humans deal with the infamous NaN or INF
- Can obtain traces at different levels
  - Machine instructions (as with GPU-FPX)
  - LLVM (as with FPChecker)

Summary : Binary Instrumentation-based Exception Checker

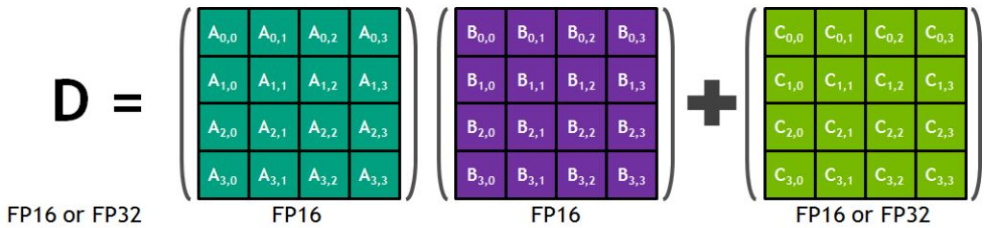
- GPU-FPX Tool has been extended to create a new tool called **nixnan**
  - Released as part of this paper's artifact (see reference [2] but also the QR Code / URL below)
  - Proven success in finding exceptions in HPC Codes
  - GPU-FPX has been **bug-fixed** and **extended to record exceptions from Tensor Cores**
- **Tensor-Cores are Hugely Important for future HPC and AI/ML**
  - **nixnan** now covers many number formats,
  - and also ships amalgamated exception-reports across threads
    - A valuable community infrastructure
    - Please use it
    - All our experiments **plus paper PDF** are HERE ⇒

Ill-Specified Hardware

Discovering specs through tests; Grounding them via FM

Blocked FMA

“Each Tensor Core operates on a 4x4 matrix”



- V100 whitepaper

The matrix multiplications

$$D = A*B+C$$

Involve block-FMA steps
(error in blocks of 4 are one rounding for */+ ...
as in a standard FMA

"Multi-Term Adders"

Each matrix accelerator unit operate on a small size matrix multiplication!

Porting danger illustrated

- D = C - A.B
- Fill A,B,C "cleverly"
  - See paper
- Code run on five TCs
  - Ideal answer : 191.984375
  - Other ans:
    - 0.0 on A100, V100, MI250, CPU
    - 255.875 on MI100
    - 191.875 on H100
- Extreme porting danger!

generally, computations of the form $\mathbf{D} = \mathbf{C} - \mathbf{A} \cdot \mathbf{B}$ fall under the Level-3 category of the Basic Linear Algebra Subprograms (BLAS) standard.

To investigate the numerical behavior of this pattern, we design a test case involving matrices $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$, each of size $2^{13} \times 2^{13}$ (i.e., 8192×8192). The matrices are populated with carefully chosen values to exercise edge cases in potential multi-term implementations, as detailed in [24].

Specifically, we initialize $\mathbf{C}_{i,j} = 2^{20}$ for all i, j; $\mathbf{A}_{i,0} = 2^{10}$; $\mathbf{A}_{i,j} = 2^{-2}$ for odd j; and $\mathbf{A}_{i,j} = 2^{-3}$ for even j (excluding $j = 0$). For matrix $\mathbf{B}$, we set $\mathbf{B}_{0,j} = 2^{10}$ and all other $\mathbf{B}_{i,j}$ entries to 2^{-3}. All values are represented in FP16 precision. Under real-number semantics, this construction yields $\mathbf{D}_{i,j} \approx 191.99218$ for all i, j

$$\begin{aligned}\mathbf{D}_{i,j} &= -(\mathbf{A}_{i,0} \cdot \mathbf{B}_{0,j} + \sum_{j\%2=1} \mathbf{A}_{i,j} \cdot \mathbf{B}_{i,j} + \sum_{\substack{j\%2=0 \\ j\neq 0}} \mathbf{A}_{i,j} \cdot \mathbf{B}_{i,j}) + \mathbf{C}_{i,j} \\ &= -(2^{10} \cdot 2^{10} - \sum_{2^{12}} 2^{-2} \cdot 2^{-3} - \sum_{2^{12}-1} 2^{-3} \cdot 2^{-3}) + 2^{20} \\ &= 2^7 + 2^6 - 2^{-6} = 191.984375\end{aligned}$$

Unfortunately (as specified under “Story-2” earlier), thanks to the differences in multi-term addition implementation, the answer can be “all zeros” on NVIDIA A100, V100, AMD MI250, and a CPU; a matrix filled with 255.875 on the AMD MI100; and one filled with 191.875 on the NVIDIA H100.

Can discover specs via tests : Block-FMA width illustrated

(details not important – see our HPDC'23 paper…)

Algorithm 1: Test Minimum Unit for FMA property preservation. The idea is to assign a moving position the $2^{-1}ulp$ value and when that position goes beyond the width of the block FMA, we get a 1 output. That index is the FMA block width.

Data: Matrices a, b, c, and d. $a's$ row's length K.

Initialize all values in a, b, c to 0
$c_{11} \leftarrow 1.$
$a_{11} \times b_{11} \leftarrow 2^{-1} \times \text{ulp}$
for $i \leftarrow 1$ **to** K **do**
    if $i > 1$ **then**
        $a_{1(i-1)} \times b_{(i-1)1} \leftarrow 0$
    $(a_{1i} \times b_{i1}) \leftarrow (2^{-1} \times \text{ulp})$
    Call **wmma**(a, b, c, d)
    if $d_{11} = 1.$ **then**
        break
if *index* $< K$ **then**
    $min\_preserve\_uint \leftarrow index$
else
    $min\_preserve\_uint$ is larger than K

Here are the kinds of features we've discovered via tests

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.

Rounding errors are a concern… but Exceptions are more of a concern

- FP32
  - Error measuring **Earth-to-Moon Distance** of 384,400 km : 22.9 meters
  - Error measuring **Height of Human** (172 cm) : **0.000103 mm**
- BF16
  - Moon: 1500 km
  - Human: **6.7 mm**
- FP16
  - Moon: Can't represent - INF
  - Human: **0.84 mm**

Rounding errors often "OK"; **Exceptions are Show-Stoppers!**

- How much rounding error is too much?
  - No definite answer
    - ML → matters
      - but backprop is "one-stop redemption of all evil"
    - HPC → matters more
      - still if the residual converges, things will often work
- However, exceptions are almost always show-stoppers!
  - Value-ranges being exceeded → INF
    - (e.g. 1.9 * 10^19) when squared → INF in FP32
      - dot-products and mat-mat sum such squares
  - INF / INF → NaN
    - NaN : a show-stopper in most cases!
- #define MAX(x,y) (x >= y ? x : y) ← buggy MAX macro in FP…

What are some challenges of Array Programming on GPUs?

- How Pytorch guards against NaNs
  - https://discuss.pytorch.org/t/outputing-nan-as-predictions-in-my-neural-network-training-loop/183151/2?utm_source=chatgpt.com
- How "Lightning" helps implement NaN callbacks and help debug NaNs
  - https://chaimrand.medium.com/debugging-the-dreaded-nan-ac3f9feac5b2
- CuSparse notes on NaN-handling
  - https://docs.nvidia.com/cuda/archive/12.8.1/pdf/CUDA_Toolkit_Release_Notes.pdf?utm_source=chatgpt.com
- ChatGPT link of a lot of NaN solutions
  - https://chatgpt.com/share/683db7c8-6520-8013-b4d2-baab767bbc65
- NaN in training
  - https://discuss.pytorch.org/t/anomaly-detection/104763/7?utm_source=chatgpt.com

Lack of formal specs + Managing their surfeit

Create specs; then manage surfeit by
Either finding ways to bridge to standard APIs, or by
Adapting applications to work using the new math

Specifying GPUs and their Versions

- A Huge Variety of Behaviors
  - Unclear how the community can keep up
    - Test-based Spec Discovery is always handy
    - Maybe we can automate tests → Spec creation
- One attractive use of Specs
  - Automated code synthesis
    - Users might do this anyhow (example given in paper)
    - So it behooves some of us to provide verification safety-nets
- How to manage the surfeit of specifications?
  - Simulate all variants on top of one fixed scheme
    - e.g. IEEE FP32
  - OR, Find a way to run existing code on the given platform
    - … and find a way to detect and recover from failures
      - Many HPC and ML codes may run robustly on "strange" HW anyhow

Synthesized efficient code: find the max element in a matrix

seems correct here – but in general, is fallible in subtle ways

```
// local thread ID within a warp
int warpThreadId = threadIdx.x & (WARP_SIZE - 1);
 float localMax   = -65504.0f;  // half-prec. min is roughly -65504

// Step 1: Each thread checks some subset of D
for (int idx = warpThreadId; idx < M * N; idx += WARP_SIZE) {
    float val   = __half2float(D[idx]);
    localMax    = fmaxf(localMax, val);
}
// Step 2: Warp-wide reduce using shuffle intrinsics
unsigned int mask = __activemask(); // active lanes
#pragma unroll
for (int offset = WARP_SIZE / 2; offset > 0; offset >>= 1) {
    float temp = __shfl_down_sync(mask, localMax, offset);
    localMax   = fmaxf(localMax, temp);
}
// Step 3: The lane 0 in the warp stores the final max to MaxVal
if (warpThreadId == 0) {
    *MaxVal = __float2half(localMax);
}
```

This best paper award winner at NFM 2025 + released SMT artifacts is a significant step in this direction…

An SMT Formalization of Mixed-Precision Matrix Multiplication

Modeling Three Generations of Tensor Cores

Benjamin Valpey[1 [0000−0002−2245−3022]] ✉, Xinyi Li[2 [0009−0005−7276−7715]], Sreepathi Pai[1 [0000−0002−3691−7238]], and Ganesh Gopalakrishnan[2 [0000−0002−4161−9278]]

[1] University of Rochester, Rochester, NY 14627, USA
{bvalpey,sree}@cs.rochester.edu
[2] University of Utah, Salt Lake City, UT 84112, USA {xin_yi.li@utah.edu, ganesh@cs.utah.edu}

The types of questions one can answer once we have formal specifications for GPU Hardware:

Do numerical algorithms such as this port reliably across GPUs?

- Will we get the same convergent results, provided the dynamic range and unit-roundoff are within acceptable limits
  - many choices at each stage of such algorithms

repeat
  Solve: $Ax = b$ @ prec. $u_1 < u_2$
  Compute Residual: $r = b - Ax$ @ prec. u_2
  Solve: $Ae = r$ @ prec. $u_3 < u_2$
  Update solution: $x = x + e$ @ prec. u_2
until Convergence

Figure 3: Steps in Iterative Refinement

Concluding Remarks

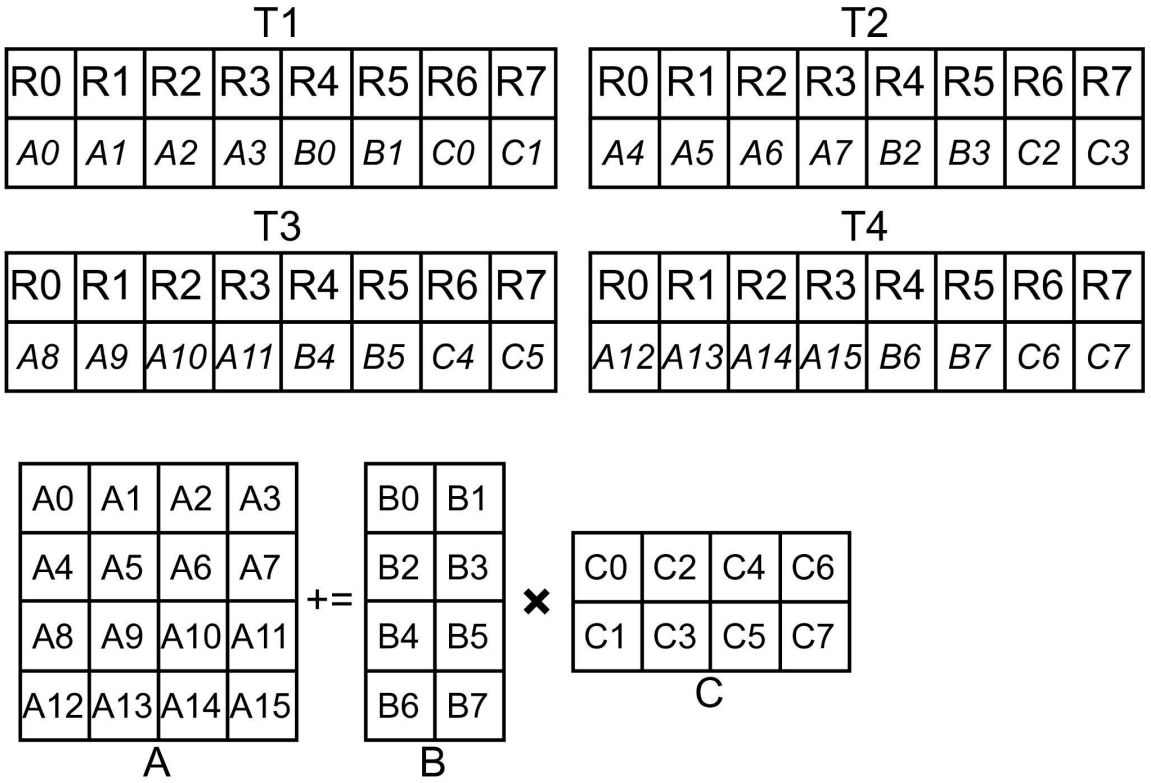
- Need for energy reduction causes GPU number-system variety
- Exceptions are a given… due to reduced precision
  - Credible PL methods critically depend on robust and versatile solutions to handle exceptions
- Test-based specification discovery is quite handy
  - Even when formal specs are available
    - Easily shared amongst groups
      - May help create specs, one day
      - Test-fusion enabled by formal specs…
        - One concrete test that covers many features
- Must manage surfeit of specs
  - Either emulate common number system
    - or migrate to it with an abundance of caution

Concluding Remarks (added..)

- Tooling in the GPU space needs a lot of attention
- Tooling such as Binary Instrumentation
  - Is provided by manufacturers
    - Long-term availability is unknown
  - Tool bug-fixes must be incorporated by tool-users
    - Community that helps convey "undocumented knowledge" (blogs, word-of-mouth, BoF,...) yet to grow
- Other tool bugs are also show-stoppers
  - e.g. how we found and helped fix an nvcc bug
- Conclusion : without sufficient number of tools and experience reports, Array Programming on GPUs will not be reliable and trustworthy.

Tensor core register layout

Each thread holds part of the matrices in its registers

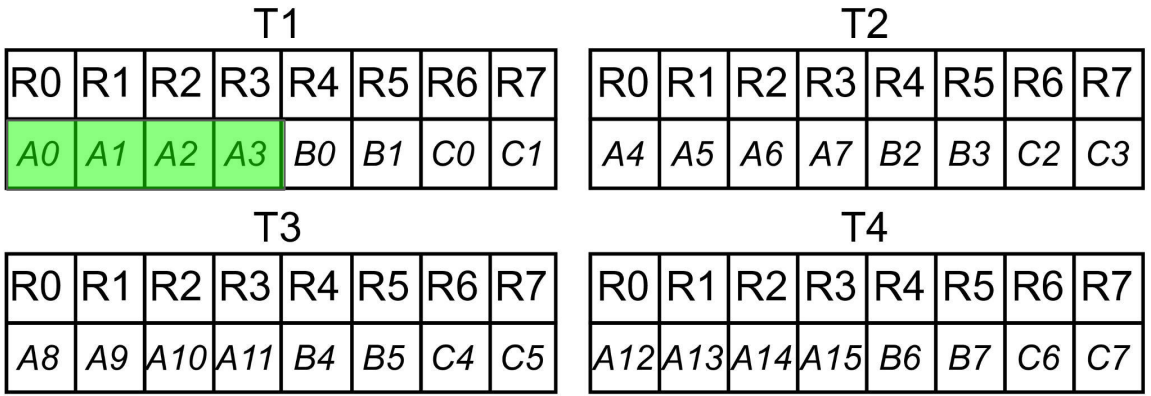


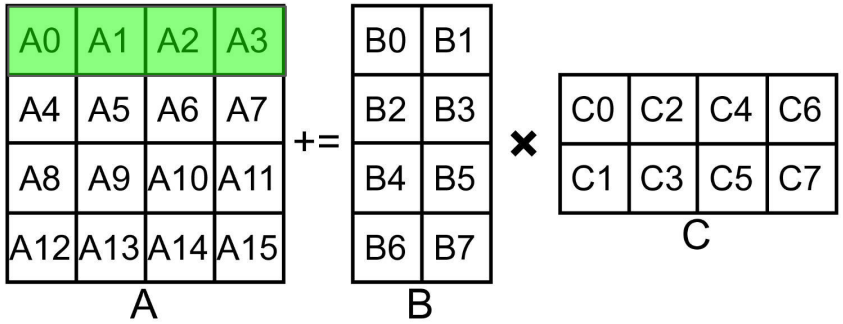
Tensor core register layout

Each thread holds part of the matrices in its registers

Thread T1 holds:

- The first four elements of A



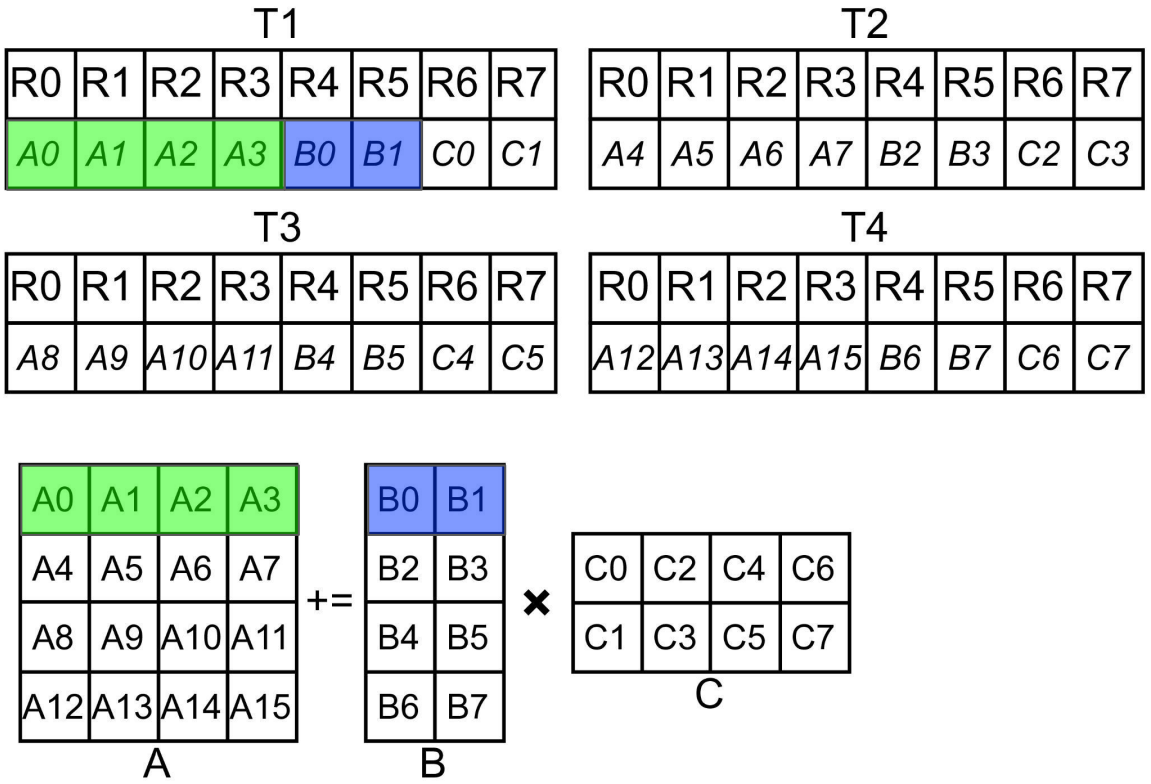


Tensor core register layout

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