



A crash course on GPU Optimization

Mark Saroufim

GPUs are expensive




ASUS ROG Strix GeForce RTX™ 4060 OC Edition Gaming Graphics...

★★★★☆ 1,695

\$386.94 ✓prime

Shop now

Back to results



Click image to open expanded view

🔖

Tesla H100 80GB NVIDIA Deep Learning GPU Compute Graphics Card

Brand: Generic

3.3 ★★★★★ 10 ratings | Search this page

\$43,989⁰⁰

Or \$3,665.75 /mo (12 mo). Select from 2 plans

Eligible for Return, Refund or Replacement within 30 days of receipt

Graphics Coprocessor	NVIDIA Tesla H100
Brand	Generic
Graphics Processor Manufacturer	NVIDIA
Compatible Devices	Desktop
Graphics Card Interface	PCI Express

About this item

- NVIDIA H100 is a high-performance GPU designed for data center and cloud-based applications, optimized for AI workloads designed for data center and cloud-based applications
- Based on the NVIDIA Ampere architecture, it has 640 Tensor Cores and 160 SMs, delivering 2.5x more compute power than the V100 GPU
- With a memory bandwidth of 1.6TB/s and PCIe Gen4 interface, it can handle large-scale data processing tasks efficiently
- Advanced features include Multi-Instance GPU (MIG) technology, enhanced NVLink, and enterprise-grade reliability tools

Report an issue with this product or seller

\$43,989⁰⁰

FREE delivery **May 30 - June 5.** Details

📍 Deliver to Mark - Sunnyvale 94087

Only 1 left in stock - order soon.



Add to Cart

Buy Now

Ships from	Global Tech Discount
Sold by	Global Tech Discount
Returns	Eligible for Return, Refund or Replacement...
Payment	Secure transaction

✓ See more

Add to List

  Astonishing speed to unleash gaming performance!

-5% \$142⁹⁹ \$149.99 ✓prime

Sponsored

How to get your money's worth buying GPUs

Mark Saroufim

Who am I

PyTorch core developer

CUDA MODE co-founder discord.gg/cudamode

These days mostly working on quantization <https://github.com/pytorch/ao>

How to make PyTorch models faster?

Fuse more

Use tensor cores

Reduce overhead

Quantize

Use a custom kernel

📄 ubsan.supp	Upgrade Pybind submodule to 2.10.4 (#103989)	last year
📄 ufunc_defs.bzl	move build_variables.bzl and ufunc_defs.bzl from pytorch-ro...	2 years ago
📄 version.txt	[release] Increase version 2.3.0->2.4.0 (#121974)	2 months ago

[📖 README](#)[💡 Code of conduct](#)[📄 License](#)[🔒 Security](#)[✎](#) [☰](#)

PyTorch

PyTorch is a Python package that provides two high-level features:

- Tensor computation (like NumPy) with strong GPU acceleration
- Deep neural networks built on a tape-based autograd system

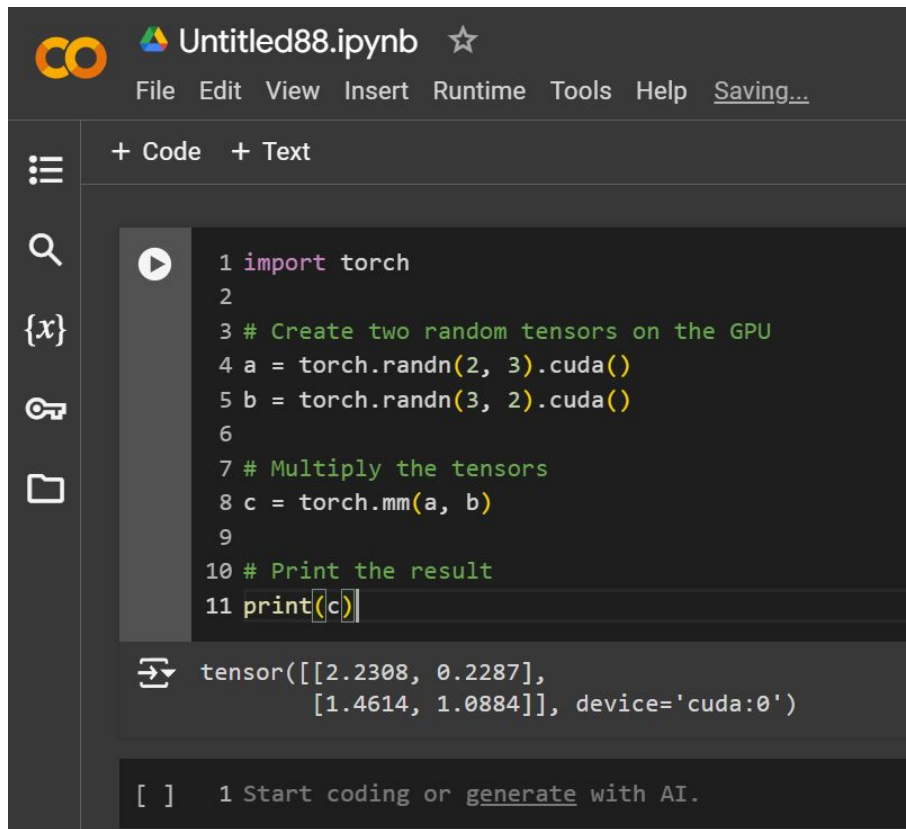
How to multiply matrices in PyTorch

Just call `.cuda()`

You don't need to explicitly write CUDA kernels

More generally need to codegenerate kernels that work over multiple devices, dtypes and layouts

https://pytorch.org/docs/stable/tensor_attributes.html



The screenshot shows a Jupyter Notebook interface with a dark theme. The title bar at the top reads 'Untitled88.ipynb' with a star icon on the right and a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', 'Help', and 'Saving...'. On the left is a sidebar with icons for a menu, search, code cells, outputs, and files. The main area contains a code cell with the following Python code:

```
1 import torch
2
3 # Create two random tensors on the GPU
4 a = torch.randn(2, 3).cuda()
5 b = torch.randn(3, 2).cuda()
6
7 # Multiply the tensors
8 c = torch.mm(a, b)
9
10 # Print the result
11 print(c)
```

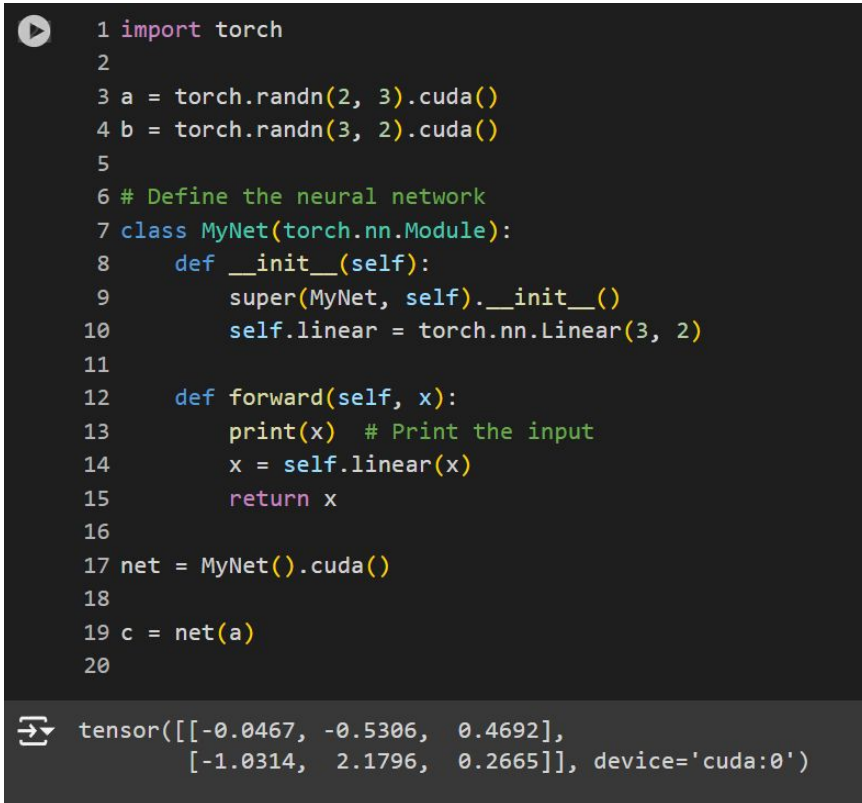
Below the code cell, the output is displayed: a tensor object with values `[[2.2308, 0.2287], [1.4614, 1.0884]]` and `device='cuda:0'`. At the bottom of the notebook, there is a status bar that says `[] 1 Start coding or generate with AI.`

Why do people love PyTorch

You can add print statements

Because PyTorch executes line by line i.e: eagerly

Not as performant as analyzing all the operations up front i.e: graph mode



```
1 import torch
2
3 a = torch.randn(2, 3).cuda()
4 b = torch.randn(3, 2).cuda()
5
6 # Define the neural network
7 class MyNet(torch.nn.Module):
8     def __init__(self):
9         super(MyNet, self).__init__()
10         self.linear = torch.nn.Linear(3, 2)
11
12     def forward(self, x):
13         print(x) # Print the input
14         x = self.linear(x)
15         return x
16
17 net = MyNet().cuda()
18
19 c = net(a)
20
```

→ tensor([[-0.0467, -0.5306, 0.4692],
 [-1.0314, 2.1796, 0.2665]], device='cuda:0')

But the world changed

If models are converging then
performance becomes key



https://en.wikipedia.org/wiki/Llama#/media/File:Llamas,_Vernagt-Stausee,_Italy.jpg

Pointwise ops

$$f(x) = \max(0, x)$$

```
bla.cu  relu.py  X
relu.py > ...
1  import torch
2  import torch.nn as nn
3
4  # Create a tensor with some negative values
5  x = torch.tensor([-1.0, 0.0, 1.0, -0.5, 2.5])
6
7  relu = nn.ReLU()
8
9  # Apply the ReLU module
10 y = relu(x)
11
12 # Print the result
13 print(y)
14
```

```
bla.cu
1 // CUDA kernel to apply ReLU activation element-wise
2 __global__ void reluKernel(float *d_in, float *d_out, int size) {
3     int idx = threadIdx.x + blockDim.x * blockIdx.x;
4     if (idx < size) {
5         d_out[idx] = (d_in[idx] > 0) ? d_in[idx] : 0;
6     }
7 }
```

Zoom into CUDA kernel

Assign every element to a thread

Threads that are close hold elements that are close in memory

Each problem typically has a unique threading strategy - [learn more in lecture 8](#)

```
1  // CUDA kernel to apply ReLU activation element-wise
2  __global__ void reluKernel(float *d_in, float *d_out, int size) {
3      int idx = threadIdx.x + blockDim.x * blockIdx.x;
4      if (idx < size) {
5          d_out[idx] = (d_in[idx] > 0) ? d_in[idx] : 0;
6      }
7  }
```

Memory Hierarchy

```
share.cu
1  __global__ void reluKernelShared(float *d_in, float *d_out, int size) {
2      int idx = threadIdx.x + blockDim.x * blockIdx.x;
3      extern __shared__ float s_data[]; // Allocate shared memory
4
5      // Load data into shared memory
6      if (idx < size) {
7          s_data[threadIdx.x] = d_in[idx];
8      }
9      __syncthreads(); // Ensure all writes to shared memory have completed
10
11     // Apply ReLU in shared memory
12     if (idx < size) {
13         s_data[threadIdx.x] = (s_data[threadIdx.x] > 0) ? s_data[threadIdx.x] : 0;
14     }
15     __syncthreads(); // Ensure all computations are done before writing back
16
17     // Write the result to global memory
18     if (idx < size) {
19         d_out[idx] = s_data[threadIdx.x];
20     }
21 }
```

TABLE IV
THE MEMORY ACCESSES LATENCIES

Memory type	CPI (cycles)
Global memory	290
L2 cache	200
L1 cache	33
Shared Memory (ld/st)	(23/19)

One of the benefits of Triton is you don't think about shared memory explicitly

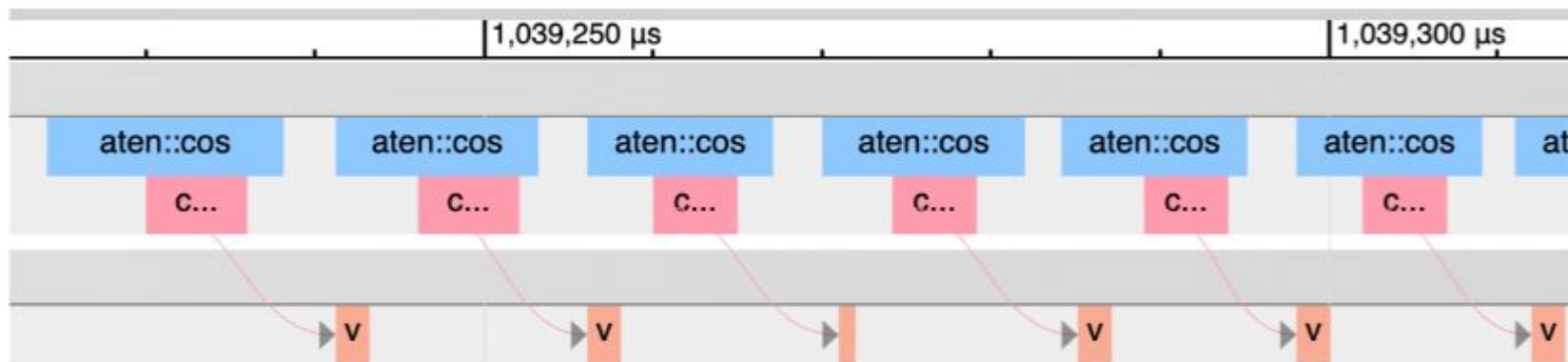
Pointwise ops

For each `torch.sin` we will launch a cuda kernel where we copy `input_tensor` to the GPU and copy it back to CPU

```
1 import torch
2
3 def apply_sin_four_times(input_tensor):
4     input_tensor = torch.sin(input_tensor)
5     input_tensor = torch.sin(input_tensor)
6     input_tensor = torch.sin(input_tensor)
7     input_tensor = torch.sin(input_tensor)
8     return input_tensor
9
10 apply_sin_four_times(torch.randn(2, 3))

tensor([[ -0.4847,  0.6178, -0.1576],
        [ 0.6072, -0.1736, -0.1104]])
```

PyTorch profiler



Lots of gaps on the GPU while it's waiting for CPU overhead

https://horace.io/brrr_intro.html

<https://pytorch.org/docs/stable/profiler.html>

Memory bandwidth

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS 624 TFLOPS*	
INT8 Tensor Core	624 TOPS 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s
Max Thermal Design Power (TDP)	300W	400W ***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB

Arithmetic intensity

ReLU on a vector: $f(x) = \max(0, x)$

For each element x_i assuming float32

- 1 read: 4 bytes
- 1 comparison
- 1 write: 4 bytes

Arithmetic intensity: $1 / 8 < 1$

Memory bound

Autoregressive decoding in LLMs is memory bandwidth bound esp for small bs

Pointwise ops

For each `torch.sin` we will launch a cuda kernel where we copy `input_tensor` to the GPU and copy it back to CPU

Can we launch a single kernel instead?

```
1 import torch
2
3 def apply_sin_four_times(input_tensor):
4     input_tensor = torch.sin(input_tensor)
5     input_tensor = torch.sin(input_tensor)
6     input_tensor = torch.sin(input_tensor)
7     input_tensor = torch.sin(input_tensor)
8     return input_tensor
9
10 apply_sin_four_times(torch.randn(2, 3))

tensor([[ -0.4847,  0.6178, -0.1576],
        [ 0.6072, -0.1736, -0.1104]])
```

Torch.compile is a fusion compiler

```
relu.py > ...
1 import torch
2 import os
3
4 def apply_sin_four_times(input_tensor):
5     input_tensor = torch.sin(input_tensor)
6     input_tensor = torch.sin(input_tensor)
7     input_tensor = torch.sin(input_tensor)
8     input_tensor = torch.sin(input_tensor)
9     return input_tensor
10
11 apply_sin_four_times = torch.compile(apply_sin_four_times)
12 apply_sin_four_times(torch.randn(1,100).cuda())
```

```
bla.cu  relu.py  cjuf6xko7pwf7bqcmuhkwfmgqhnbavqjdhyklw73iowwupedfh.py X  c6mh5f3zj5ppvj2qze3pcxxlrmhyay3v47jf6m3bso3hb3j7crp.py  ▶ ▢ ...
tmp > torchinductor_zeus > ju > cjuf6xko7pwf7bqcmuhkwfmgqhnbavqjdhyklw73iowwupedfh.py > ...
31 triton_poi_fused_sin_0 = async_compile.triton("triton_", '''
43     triton_meta={'signature': {0: '*fp32', 1: '*fp32', 2: 'i32'}, 'device': 0, 'device_type': 'cuda', 'constants': {}, 'configs': [inst:
44         inductor_meta={'autotune_hints': set(), 'kernel_name': 'triton_poi_fused_sin_0', 'mutated_arg_names': []}],
45         min_elem_per_thread=0
46     )
47 @triton.jit
48 def triton(in_ptr0, out_ptr0, xnumel, XBLOCK : tl.constexpr):
49     xnumel = 100
50     xoffset = tl.program_id(0) * XBLOCK
51     xindex = xoffset + tl.arange(0, XBLOCK)[:XBLOCK]
52     xmask = xindex < xnumel
53     x0 = xindex
54     tmp0 = tl.load(in_ptr0 + (x0), xmask)
55     tmp1 = tl.sin(tmp0)
56     tmp2 = tl.sin(tmp1)
57     tmp3 = tl.sin(tmp2)
58     tmp4 = tl.sin(tmp3)
59     tl.store(out_ptr0 + (x0), tmp4, xmask)
60 '''
61
62 import triton

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS
h.Float32)
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG] fn = lambda: call([arg0_1])
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG] return print_performance(fn, times=times, repeat=repeat)
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG]
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG]
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG] if __name__ == "__main__":
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG]     from torch._inductor.wrapper_benchmark import compiled_module_main
[2024-05-22 02:51:38,095] [0/0] torch._inductor.graph.__output_code: [DEBUG]     compiled_module_main('None', benchmark_compiled_module)
[2024-05-22 02:51:38,096] [0/0] torch._inductor.graph.__output_code: [INFO] Output code written to: /tmp/torchinductor_zeus/ju/cjuf6xko7pwf7bqcmuhkwf
mgqhnbavqjdhyklw73iowwupedfh.py
⚡ ~ TORCH_LOGS="output_code" python relu.py]
```

<https://pytorch.org/blog/pytorch-2-paper-tutorial/>

Tensor cores

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS 624 TFLOPS*	
INT8 Tensor Core	624 TOPS 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s
Max Thermal Design Power (TDP)	300W	400W ***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB

TENSOR CORE 4X4X4 MATRIX-MULTIPLY ACC

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

FP16 or FP32 FP16 FP16 or FP32

How to use tensor cores

```
torch.set_float32_matmul_precision("high")
```

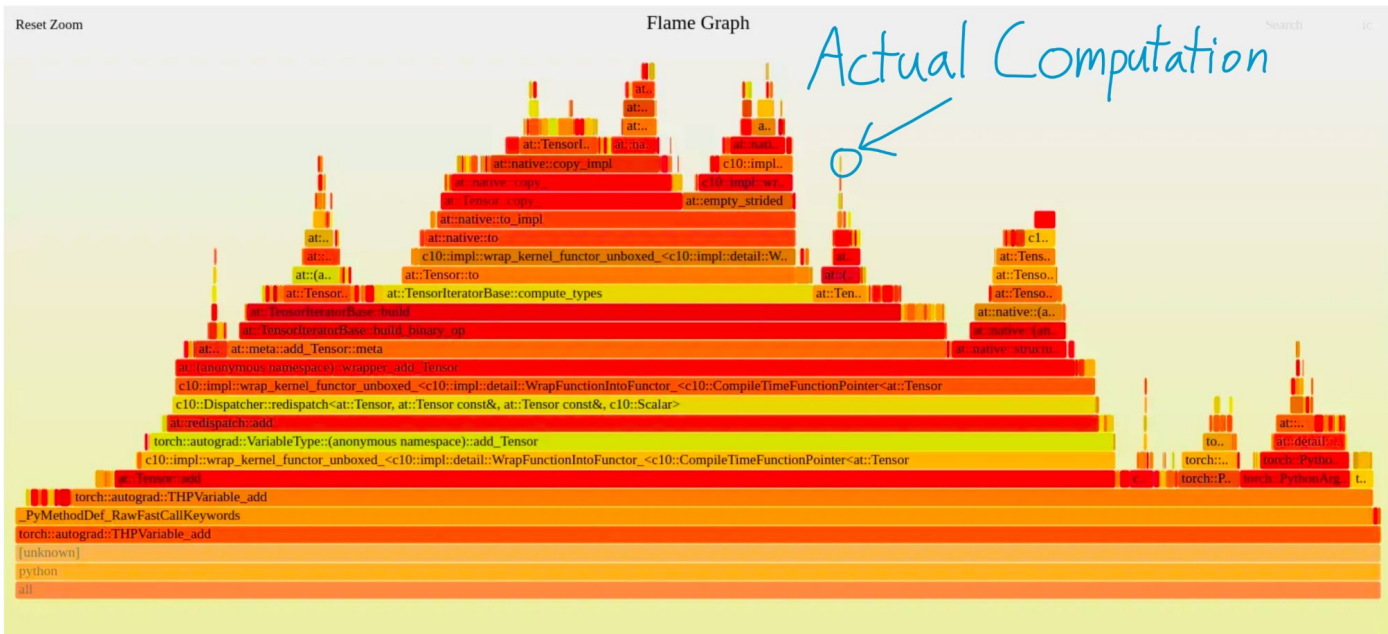
Enabling by default had some accuracy issues - see

<https://dev-discuss.pytorch.org/t/pytorch-and-tensorfloat32/504>

Not much public docs but NVIDIA team speaking at CUDA MODE on this on June

7 <https://discord.com/events/1189498204333543425/1242345685475524701>

Overhead reduction



https://horace.io/brrr_intro.html

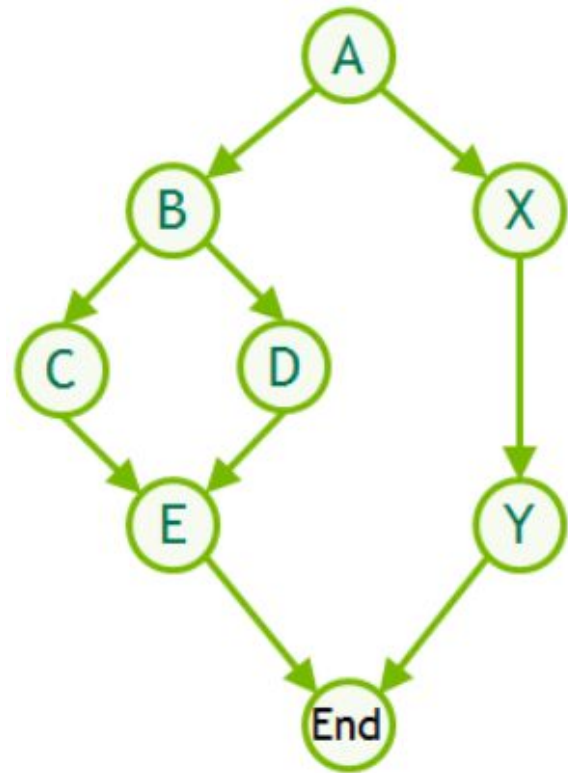
How to hide overhead

CUDA kernels are async so queue them up -> CUDA graphs

```
torch.compile(model, mode="reduce-overhead")
```

Has a small memory overhead but we'd like to turn this on by default

<https://github.com/pytorch/pytorch/issues/121968>



Quantization

	A100 80GB PCIe	A100 80GB SXM
FP64	9.7 TFLOPS	
FP64 Tensor Core	19.5 TFLOPS	
FP32	19.5 TFLOPS	
Tensor Float 32 (TF32)	156 TFLOPS 312 TFLOPS*	
BFLOAT16 Tensor Core	312 TFLOPS 624 TFLOPS*	
FP16 Tensor Core	312 TFLOPS 624 TFLOPS*	
INT8 Tensor Core	624 TOPS 1248 TOPS*	
GPU Memory	80GB HBM2e	80GB HBM2e
GPU Memory Bandwidth	1,935 GB/s	2,039 GB/s
Max Thermal Design Power (TDP)	300W	400W ***
Multi-Instance GPU	Up to 7 MIGs @ 10GB	Up to 7 MIGs @ 10GB

Quantization also helps memory bound workloads

ReLU on a vector: $f(x) = \max(0, x)$

For each element x_i assuming int8

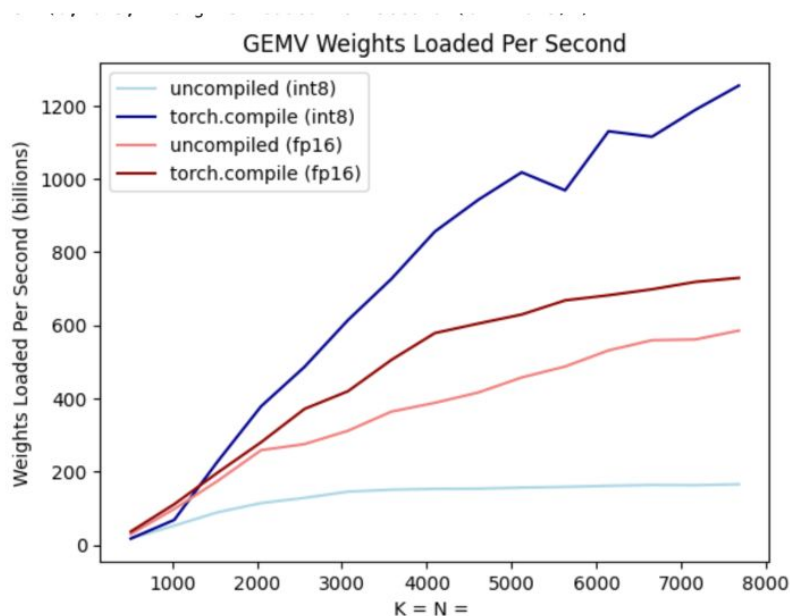
- 1 read: 1 bytes
- 1 comparison
- 1 write: 1 bytes

Arithmetic intensity: $1/2 < 1$

Much less memory bandwidth bound!

Int8 weight only quantization

```
x: bf16[1, K]
weight: int8[K, N]
@torch.compile
def int8_mm(x, weight):
    return F.linear(x, weight.to(torch.bfloat16))
```



Terminology rant

Int 8 is ambiguous

W8A16 -> Int 8 Weights with FP 16 activation

What about gradients and optimizer?

Not applied over all the model! Only the linear layers

Also how the heck do people run smaller than int 8 without Pytorch support?

Digression: Bit packing

```
1 import torch
2
3 def pack(uint4_1, uint4_2):
4     return (uint4_1 & 0x0F) | ((uint4_2 & 0x0F) << 4)
5
6 def unpack(uint8):
7     uint4_1 = uint8 & 0x0F
8     uint4_2 = (uint8 >> 4) & 0x0F
9     return uint4_1, uint4_2
10
11
12 def uint4_vector_addition(vec1, vec2):
13     uint4_1_1, uint4_1_2 = unpack(vec1)
14     uint4_2_1, uint4_2_2 = unpack(vec2)
15
16     sum_1 = (uint4_1_1 + uint4_2_1) % 16
17     sum_2 = (uint4_1_2 + uint4_2_2) % 16
18
19     return pack(sum_1, sum_2)
20
21 # Create example vectors
22 vec1 = torch.tensor([pack(torch.tensor(3), torch.tensor(7)), pack(torch.tensor(12), torch.tensor(1))], dtype=torch.uint8)
23 vec2 = torch.tensor([pack(torch.tensor(4), torch.tensor(2)), pack(torch.tensor(6), torch.tensor(9))], dtype=torch.uint8)
24
25 # Perform uint4 vector addition
26 result = uint4_vector_addition(vec1, vec2)
27
28 # Unpack the result to see the uint4 values
29 result_unpacked = [unpack(r) for r in result]
30 print(result_unpacked)
```

What about compute bound problems?



Public Domain, <https://commons.wikimedia.org/w/index.php?curid=656166>

Compilers kernel authors

Response

Official Comment  Authors  23 Nov 2023, 04:00  Everyone

Comment:

Thank you for the detailed and constructed feedback.

Q: Figure 5 baselines.

A: Here we measure just the attention decoding operation, not the end-to-end latency. We agree that CUDA graph helps end-to-end generation speed, especially for small models where the CPU overhead can be large. We have benchmarked the kernel generated by torch.compile which is faster than Pytorch eager but still not as fast as the hand-written kernel from FasterTransformer. FA2 decoding kernel is faster still.

Q: Comparison to compilers.

A: Compilers can generally perform fusion. However, optimizations that require mathematical rewriting of the same expression (while maintaining numerical stability) are generally harder for compilers. Jax uses the powerful XLA compiler, but currently the best implementation of attention on TPUs is a version of FA2 implemented in Pallas, where the programmer still specifies some details (which elements should be in HBM vs SRAM) leaving the Pallas compiler to generate code to invoke the systolic array for matmuls or vector units for other computation.

Q: gap between different implementations, e.g. xformers, Cutlass, Triton.

xformers uses cutlass to implement a similar algorithm to FA, while Triton generates ptx directly. The differences in speed are due to low-level implementation: e.g. the Triton compiler only deals with power-of-2 block sizes, while with cutlass one can use non-power-of-2 block sizes. The triton compiler will use async memory copy automatically, while with Cutlass one has to spend more effort to use these hardware features.

Q: Change in hardware trend.

This is a great question. HBM bandwidth not increasing as fast as matmul FLOPs means that techniques such as FA/FA2 will be even more relevant in the future. New hardware features (asynchronous computation, dataflow architecture) might require new ideas to efficiently use the hardware, and is an exciting future area.

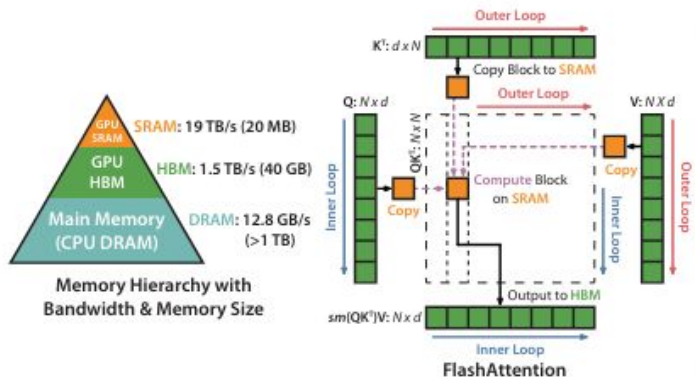
Q: FA2 with variable sequence lengths.

FA2 does support variable sequence lengths. Sequences are concatenated with no padding token, with an array indicating the lengths of each sequences. This is convenient for settings where sequences can have different lengths, and avoids wasting computation on padded tokens.

Add: [Public Comment](#)

<https://openreview.net/forum?id=mZn2Xyh9Ec>

Flash Attention



Algorithm 3 Safe softmax with online normalizer calculation

```

1:  $m_0 \leftarrow -\infty$ 
2:  $d_0 \leftarrow 0$ 
3: for  $j \leftarrow 1, V$  do
4:    $m_j \leftarrow \max(m_{j-1}, x_j)$ 
5:    $d_j \leftarrow d_{j-1} \times e^{m_{j-1}-m_j} + e^{x_j-m_j}$ 
6: end for
7: for  $i \leftarrow 1, V$  do
8:    $y_i \leftarrow \frac{e^{x_i-m_V}}{d_V}$ 
9: end for
  
```

Learn the basics of CUDA

Read the book

Do the exercises

Ask questions on

discord.gg/cudamode

Books • Computers & Technology • Hardware & DIY

1 You purchased this edition on August 9, 2023. | View this order | View Your Books

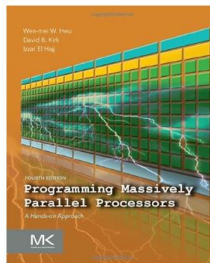
Follow the author



Wen-mei W. Hwu

+ Follow

More books from this author



Click image to open expanded view

Read sample

Programming Massively Parallel Processors: A Hands-on Approach 4th Edition

by Wen-mei W. Hwu (Author), David B. Kirk (Author), Izzat El Hajj (Author)

4.9 ★★★★★ 22 ratings

#1 Best Seller in Parallel Computer Programming

See all formats and editions

100+ bought or read in past month

Programming Massively Parallel Processors: A Hands-on Approach shows both students and professionals alike the basic concepts of parallel programming and GPU architecture. Concise, intuitive, and practical, it is based on years of road-testing in the authors' own parallel computing courses. Various techniques for constructing and optimizing parallel programs are explored in detail, while case studies demonstrate the development process, which begins with computational thinking and ends with effective and efficient parallel programs. The new edition includes updated coverage of CUDA, including the newer libraries such as cuDNN. New chapters on frequently used parallel patterns have been added, and case studies have been updated to reflect current industry practices.

- **Parallel Patterns** Introduces new chapters on frequently used parallel patterns (stencil, reduction, sorting) and major improvements to previous chapters (convolution, histogram, sparse matrices, graph traversal, deep learning)
- **Ampere** Includes a new chapter focused on GPU architecture and draws examples from recent architecture generations, including Ampere
- **Systematic Approach** Incorporates major improvements to abstract discussions of problem decomposition strategies and performance considerations, with a new optimization checklist

Report an issue with this product or seller

ISBN-10



0323912311

ISBN-13



978-0323912310

Edition



4th

Publisher



Morgan Kaufmann

Publication date



August 18, 2022

Enhance your NLP proficiency with modern frameworks, explore mathematical foundations and code samples, and gain expert insights into future trends



Mastering NLP from Foundations to LLMs: Apply advanced rule-based techniques to... by Lior Gazit
★★★★☆ 4
\$50.34 Paperback

Sponsored

Kindle
\$27.66 - \$56.79
Available instantly

Paperback
\$54.98 - \$76.40

Other Used and New from \$54.98

Delivery

Pickup

Buy new:

-28% \$65¹⁶

List Price: \$89.96

FREE delivery June 3, Details

Deliver to Mark - Sunnyvale 94087

Only 15 left in stock - order soon.

Quantity: 1

Add to Cart

Buy Now

Ships from

HealthSciences&Technology

Sold by

HealthSciences&Technology

Returns

Eligible for Return, Refund or Replacement within 30 days of receipt

Customer Service

HealthSciences&Technology

See more

Save with Used - Acceptable

\$54⁹⁸

\$3.99 delivery May 29 - June 3

Ships from: the_nps_store

Sold by: the_nps_store

Add to List

Tools of the trade:

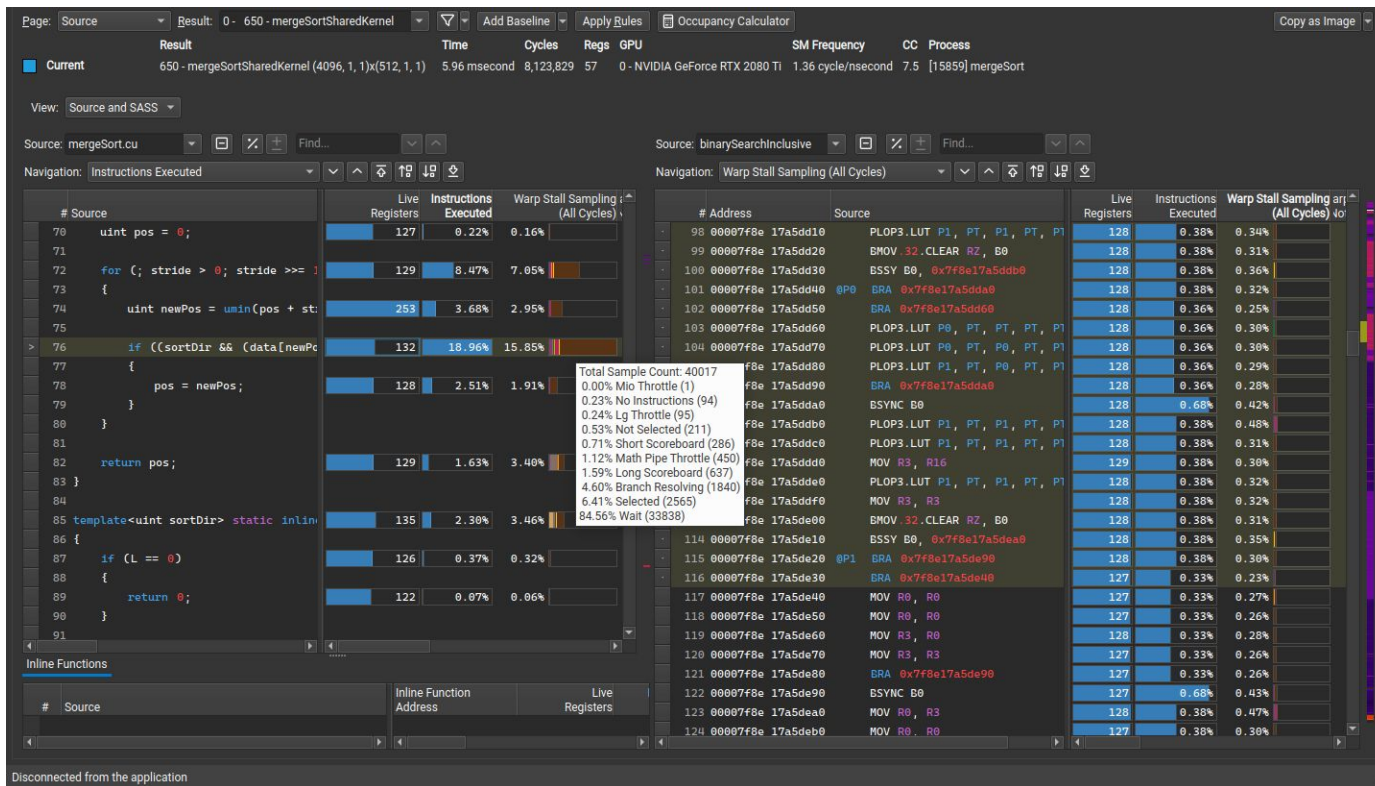
`torch.utils.cpp_extension.load_inline()`

Load a cuda file or string directly in
your PyTorch programs

Will auto generate files with the right
headers and build scripts for you

```
cpp_source = "torch::Tensor square_matrix(torch::Tensor matrix);"  
  
# Load the CUDA kernel as a PyTorch extension  
square_matrix_extension = load_inline(  
    name='square_matrix_extension',  
    cpp_sources=cpp_source,  
    cuda_sources=cuda_source,  
    functions=['square_matrix'],  
    with_cuda=True,  
    extra_cuda_cflags=["-O2"],  
    build_directory='./load_inline_cuda',  
    # extra_cuda_cflags=['--expt-relaxed-constexpr']  
)  
  
a = torch.tensor([[1., 2., 3.], [4., 5., 6.]], device='cuda')  
print(square_matrix_extension.square_matrix(a))
```

Tools of the trace: ncu



<https://docs.nvidia.com/nsight-compute/NsightCompute/index.html>

<https://www.youtube.com/watch?v=SGhfUhlwB4>

Go deeper

Learn Triton <https://github.com/cuda-mode/triton-index/> (ChatGPT is not great at Triton let's create more tokens)

Start a working group in discord.gg/cudamode to keep yourself accountable

Ship your custom kernels

1. CUDA: <https://github.com/pytorch/ao/issues/137>
2. Triton: https://pytorch.org/tutorials/recipes/torch_compile_user_defined_triton_kernel_tutorial.html

Why write custom CUDA/Triton kernels?

Intrinsic: It's fun

Extrinsic: People will never want slower models

Public docs are not great on these topics so easiest to learn via mentorship that's why I'm so passionate about discord.gg/cudamode

How to make PyTorch models faster?

Fuse more: `torch.compile(model)`

Use tensor cores: `torch.set_float32_matmul_precision("high")`

Reduce overhead: `torch.compile(model, mode="reduce-overhead")`

Quantize: Moving fast <https://github.com/pytorch/ao>

Use a custom kernel: Join discord.gg/cudamode

Why I'm working on ao

torch.compile has solved my fusion, overhead and tensor core problems

Quantization and custom kernel space is moving really fast

1. fp4/6/8
2. Int 3/4/5
3. Bitnet

A lot of kernel development is social so my online diet has become

1. <https://github.com/pytorch/ao>
2. discord.gg/cudamode

If you want your first GPU optimization project please reach out to me on CUDA MODE

Learn more

Free hardware

- <https://lightning.ai/>
- <https://colab.research.google.com/>

The basics

- discord.gg/cudamode
- <https://www.youtube.com/@CUDAMODE>
- <https://github.com/cuda-mode/resource-stream>
- [PMPP book](#)

Optimizing PyTorch code

- <https://pytorch.org/blog/accelerating-generative-ai/>
- <https://pytorch.org/blog/accelerating-generative-ai-2/>

Optimizing CUDA code

- <https://thundercats.fandom.com/wiki/Thunderkittens>
- <https://github.com/NVIDIA/cutlass>
- <https://github.com/NVIDIA/cccl>
- <https://github.com/NVIDIA/cuCollections>

Optimizing Triton code

- <https://github.com/cuda-mode/triton-index/>

Enjoy the rest of the
workshop!