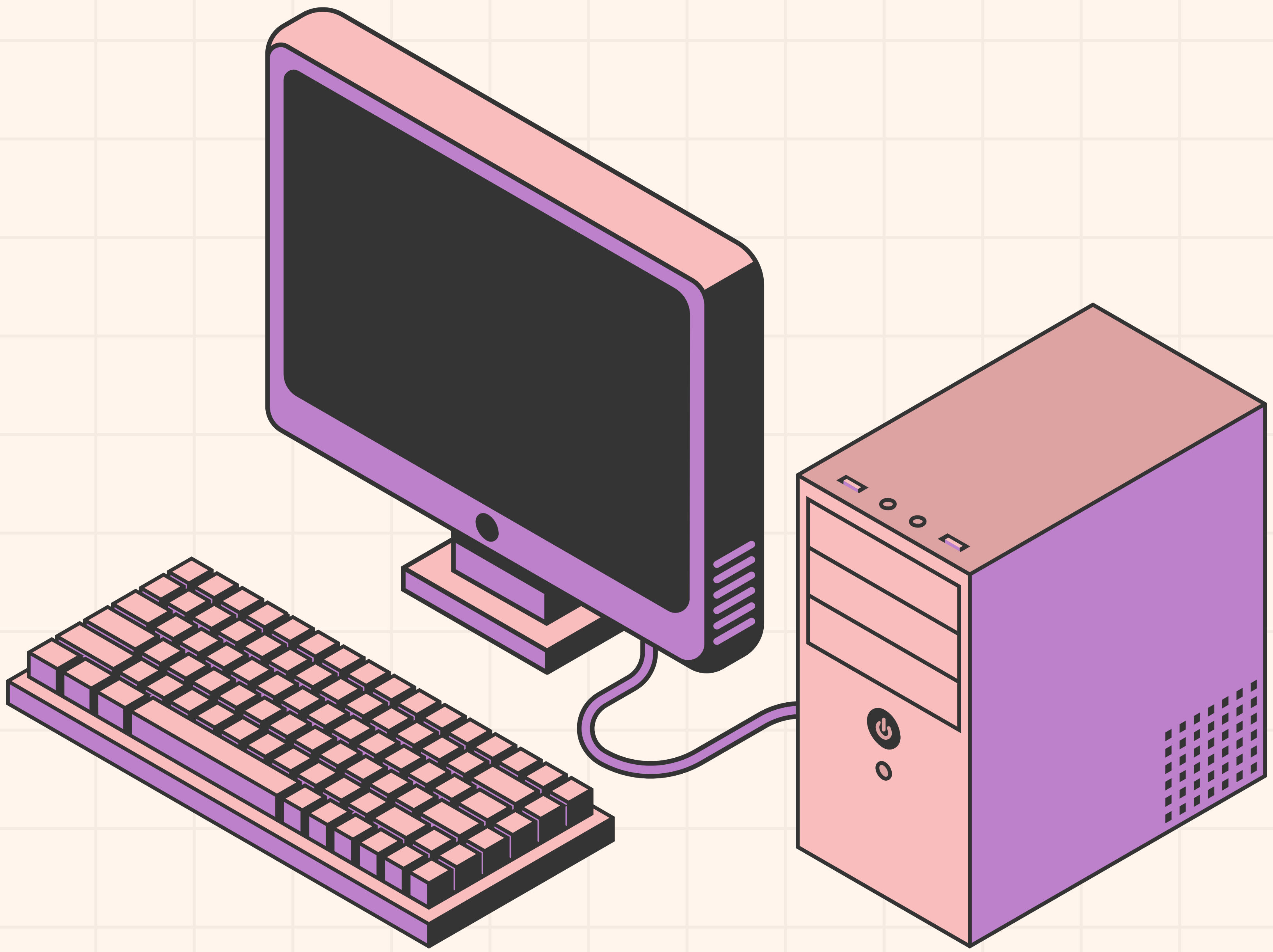
MNIST DIGIT CLASSIFICATION *ACCELERATED WITH CUDA*

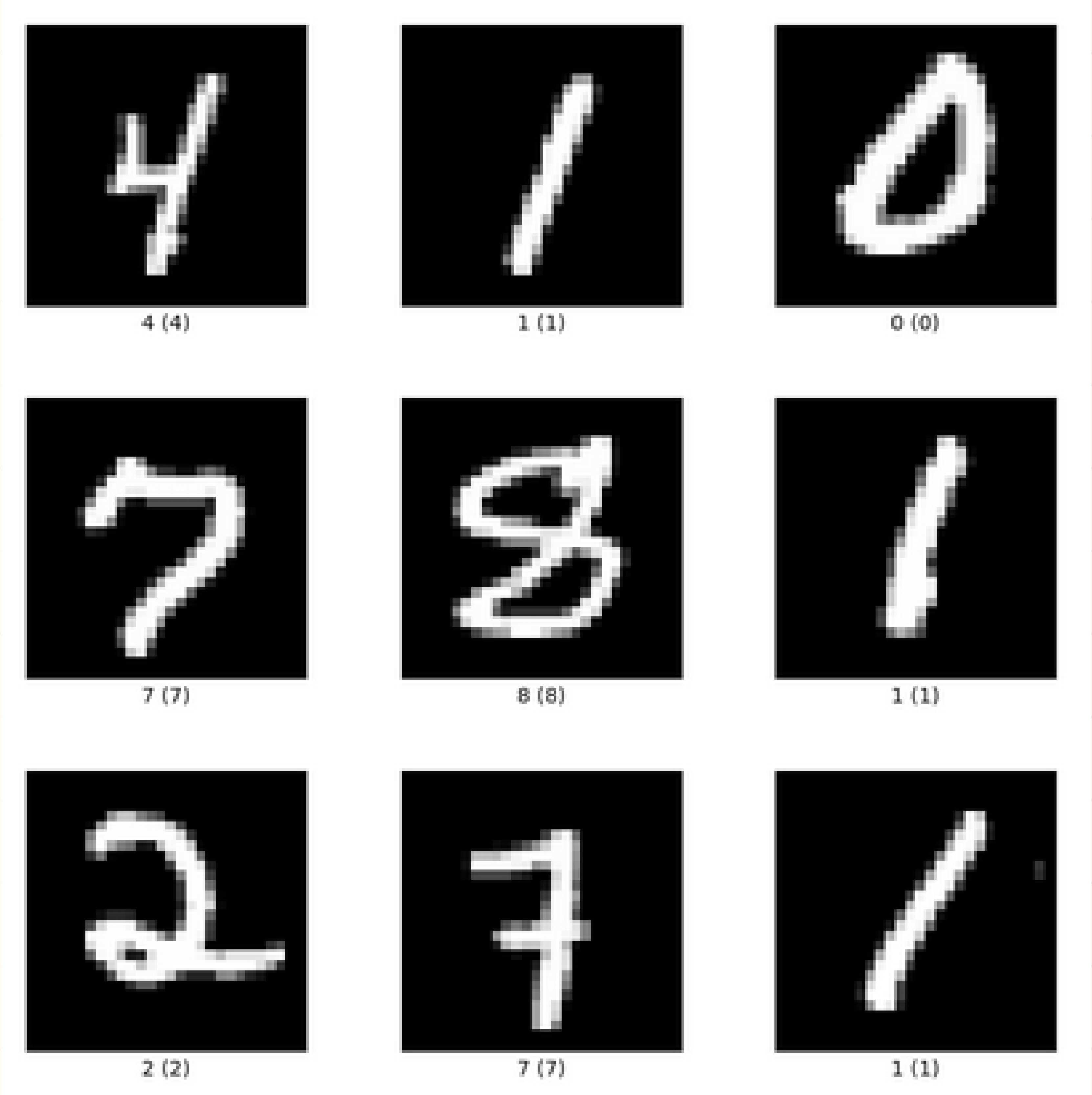


Sahrish Mustafa 22i0977
Aalyan Raza Kazmi 22i0833

MNIST

(Modified National Institute of Standards and Technology database)

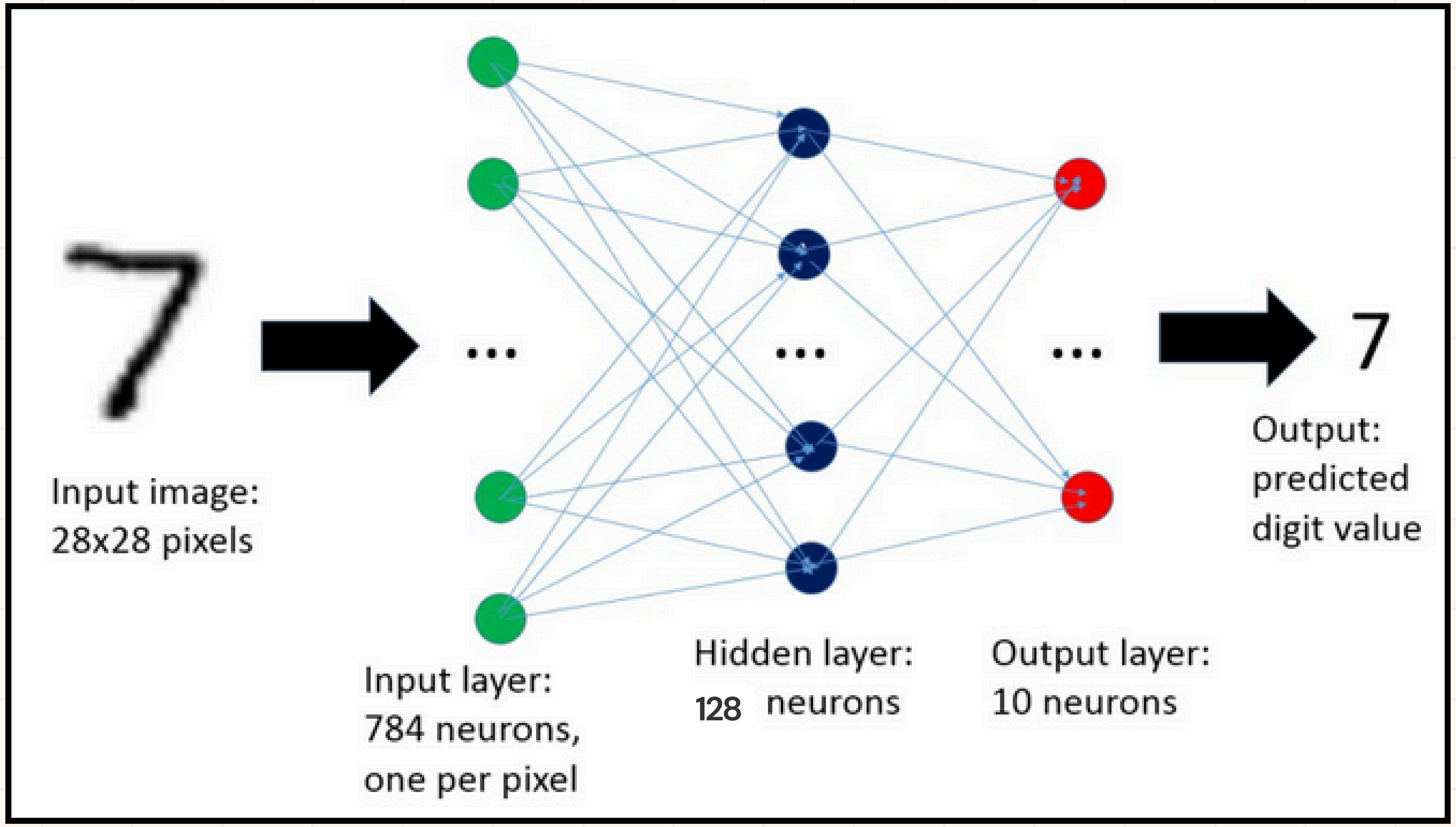
- Handwritten Digits
- 70,000 Images
- 28x28 pixels
- 10 classes (0-9)



NEURAL NETWORK - V1

INPUT LAYER

- Size: 784 neurons
- 28x28 pixel grayscale images. Flattened, each image becomes a 784-dimensional vector.

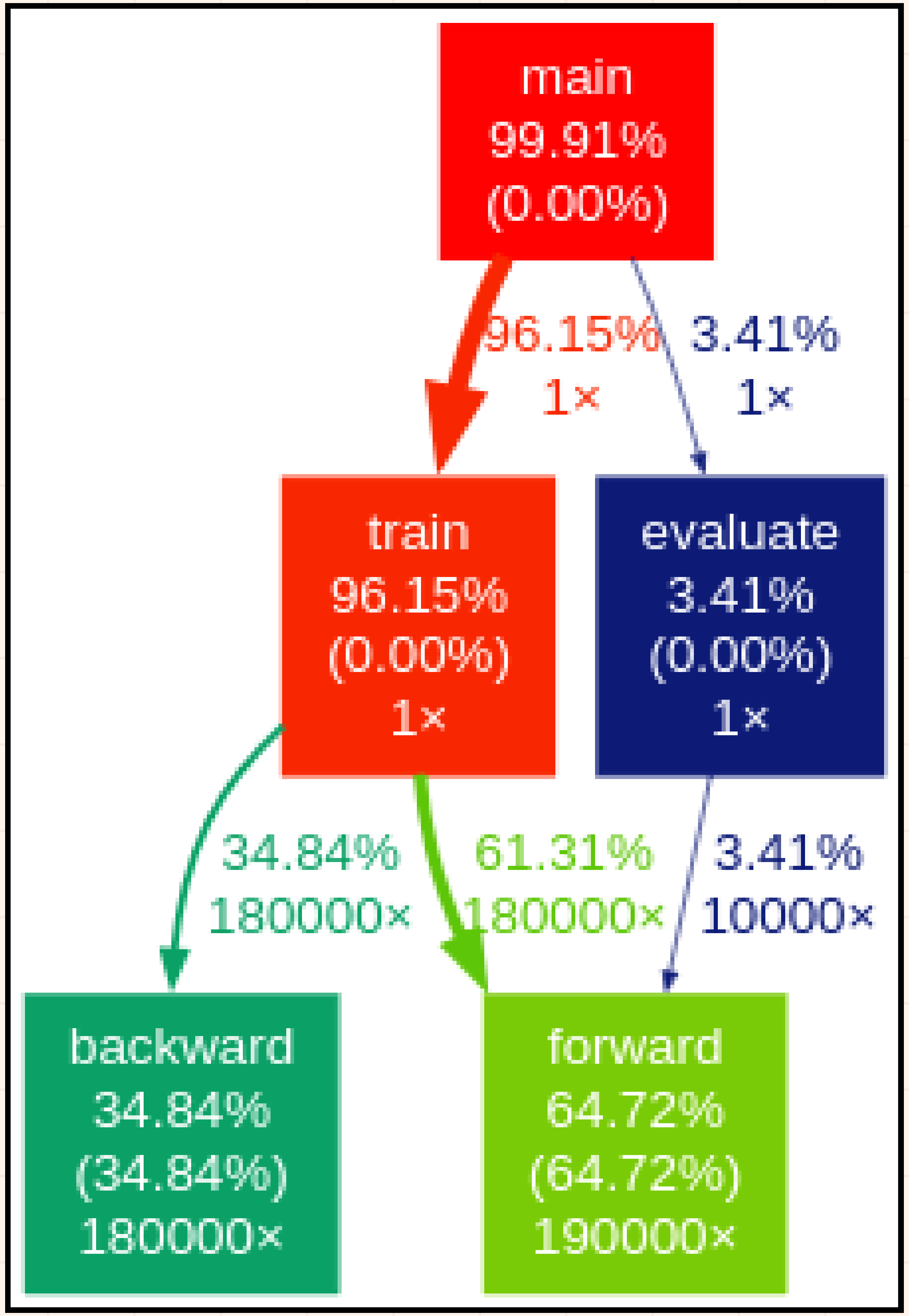


OUTPUT LAYER

- Size: 10 neurons
- Weights (W2): A matrix of size 10 x 128
- Biases (b2): A vector of size 10
- Activation Function: Softmax (raw outputs to probabilities)

HIDDEN LAYER

- Size: 128 neurons
- Weights (W1): A matrix of size 128 x 784
- Biases (b1): A vector of size 128
- Activation Function: ReLU (max(0, x))



ACCELERATION PLAN

- parallelize forward() and backward() by converting loops to kernels running on the GPU

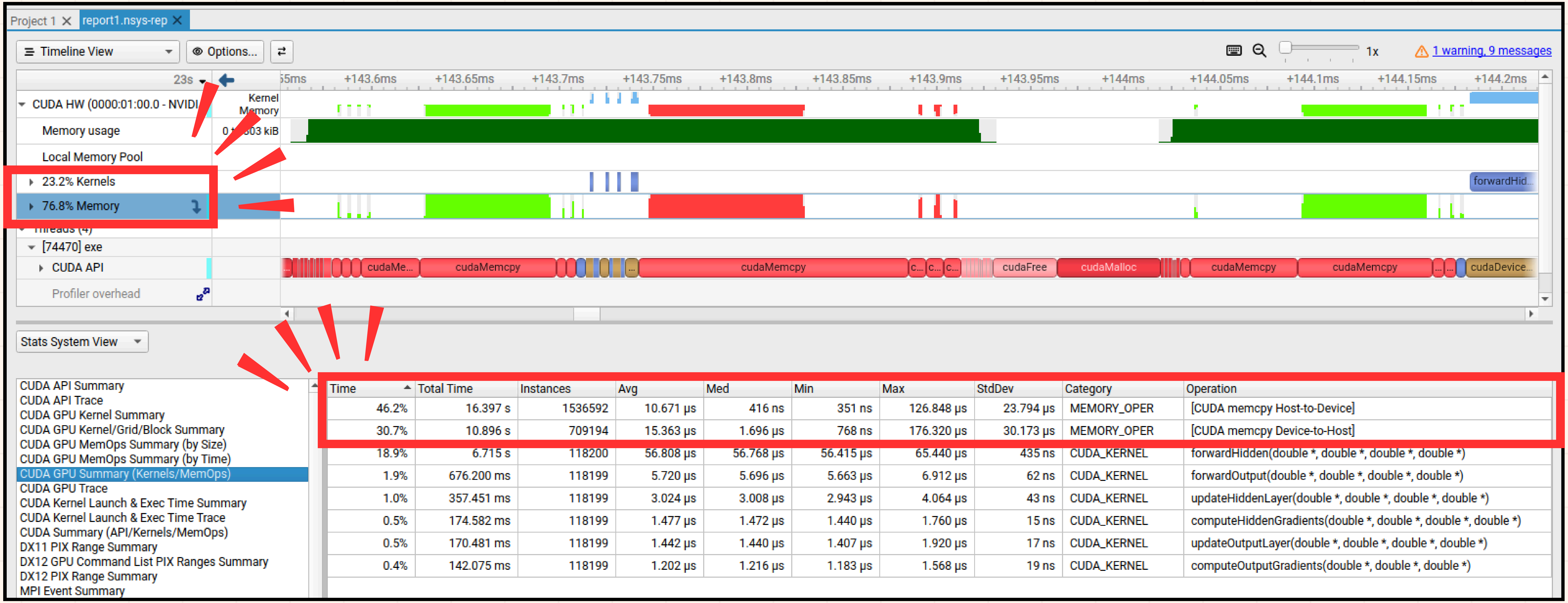
FORWARD PROPAGATION

V1

```
FUNCTION FORWARD_PROPAGATION(INPUT):
    // COMPUTE HIDDEN LAYER
    FOR I IN RANGE(HIDDEN_SIZE):
        SUM = BIAS1[I]
        FOR J IN RANGE(INPUT_SIZE):
            SUM += W1[I][J] * INPUT[J]
        HIDDEN[I] = RELU(SUM)

    // COMPUTE OUTPUT LAYER
    FOR I IN RANGE(OUTPUT_SIZE):
        SUM = BIAS2[I]
        FOR J IN RANGE(HIDDEN_SIZE):
            SUM += W2[I][J] * HIDDEN[J]
        OUTPUT[I] = SUM  // SOFTMAX APPLIED LATER
```

V2

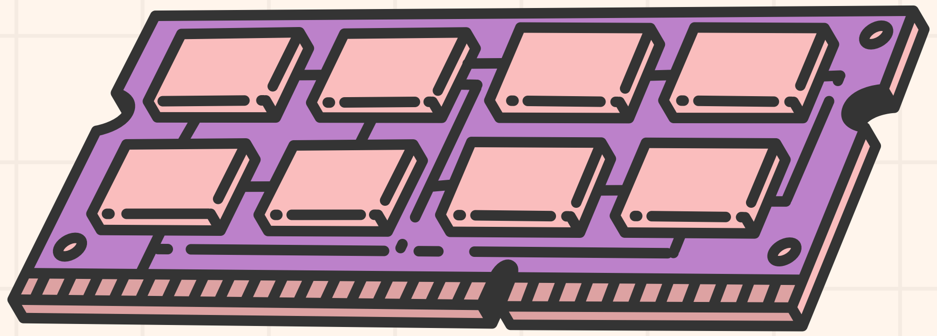
```
KERNEL FORWARDHIDDEN(W1, B1, INPUT, HIDDEN):
    I = THREADIDX.X
    IF I < HIDDEN_SIZE:
        SUM = B1[I]
        FOR J IN RANGE(INPUT_SIZE):
            SUM += W1[I][J] * INPUT[J]
        HIDDEN[I] = RELU(SUM)

KERNEL FORWARDOUTPUT(W2, B2, HIDDEN, OUTPUT):
    I = THREADIDX.X
    IF I < OUTPUT_SIZE:
        SUM = B2[I]
        FOR J IN RANGE(HIDDEN_SIZE):
            SUM += W2[I][J] * HIDDEN[J]
        OUTPUT[I] = SUM  // SOFTMAX ON HOST
```



Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
46.2%	16.397 s	1536592	10.671 μs	416 ns	351 ns	126.848 μs	23.794 μs	MEMORY_OPER	[CUDA memcpy Host-to-Device]
30.7%	10.896 s	709194	15.363 μs	1.696 μs	768 ns	176.320 μs	30.173 μs	MEMORY_OPER	[CUDA memcpy Device-to-Host]
18.9%	6.715 s	118200	56.808 μs	56.768 μs	56.415 μs	65.440 μs	435 ns	CUDA_KERNEL	forwardHidden(double *, double *, double *, double *)
1.9%	676.200 ms	118199	5.720 μs	5.696 μs	5.663 μs	6.912 μs	62 ns	CUDA_KERNEL	forwardOutput(double *, double *, double *, double *)
1.0%	357.451 ms	118199	3.024 μs	3.008 μs	2.943 μs	4.064 μs	43 ns	CUDA_KERNEL	updateHiddenLayer(double *, double *, double *, double *)
0.5%	174.582 ms	118199	1.477 μs	1.472 μs	1.440 μs	1.760 μs	15 ns	CUDA_KERNEL	computeHiddenGradients(double *, double *, double *, double *)
0.5%	170.481 ms	118199	1.442 μs	1.440 μs	1.407 μs	1.920 μs	17 ns	CUDA_KERNEL	updateOutputLayer(double *, double *, double *, double *)
0.4%	142.075 ms	118199	1.202 μs	1.216 μs	1.183 μs	1.568 μs	19 ns	CUDA_KERNEL	computeOutputGradients(double *, double *, double *)

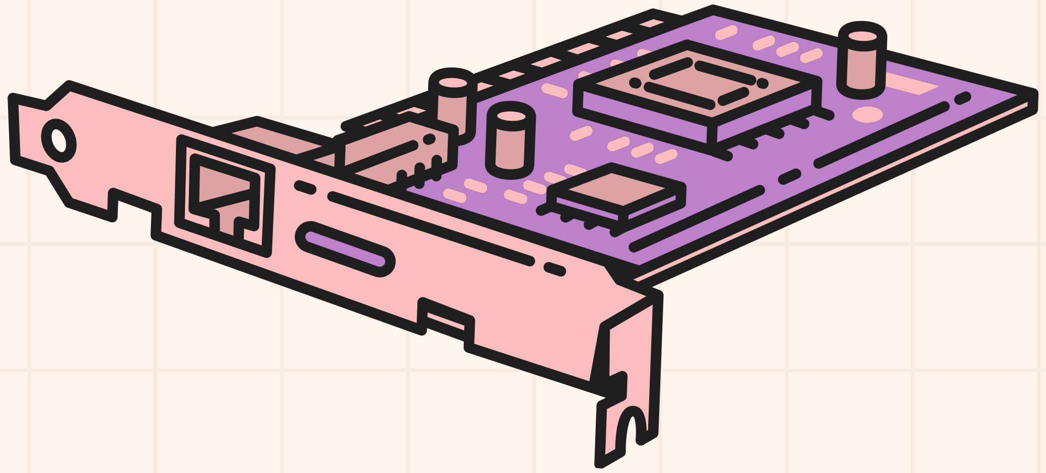
ACCELERATION PLAN

- TOO MUCH MEMORY TRANSFER TIME!
- . . . . REDUCE MEMORY TRANSFER (DTH / HTD)

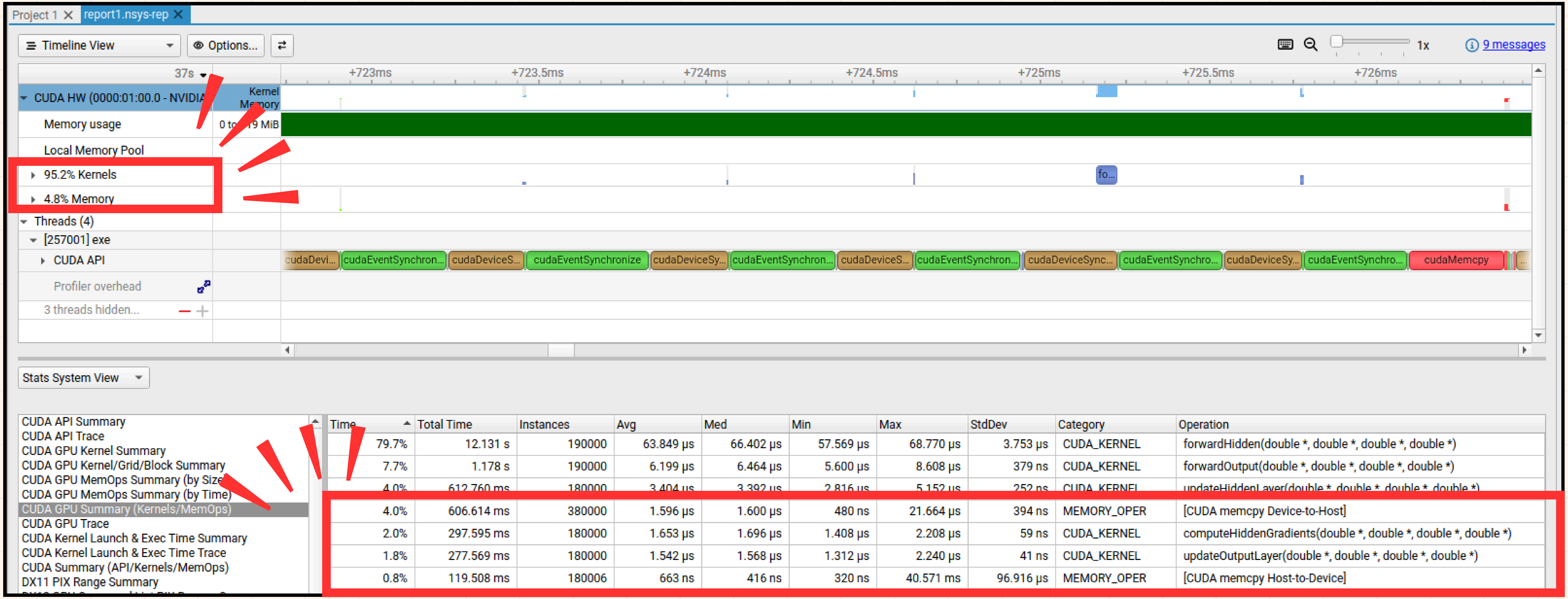


CPU RAM (HOST STORAGE)

previously, we were copying each image from the host to the device 60,000 times per epoch during training.
Now, we copy the entire dataset in the beginning only once



GPU GLOBAL MEMORY (DEVICE STORAGE)



ACCELERATION PLAN

- > **13 SECONDS** ON KERNELS
- . . . SPEED UP KERNELS

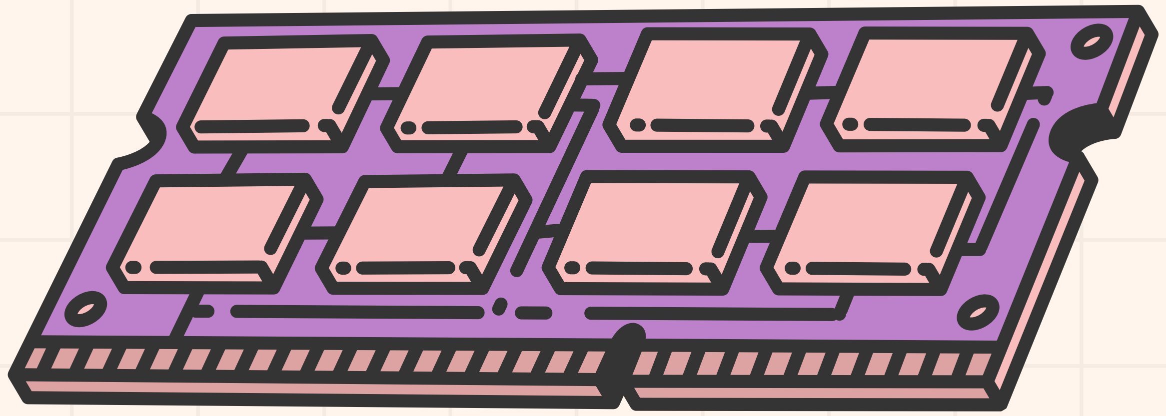
OPTIMIZED GPU VERSION - V3

1. Optimizing launch configurations
2. Shared memory
3. Half-precision (FP16) datatype
4. Coalesced memory access

LAUNCH CONFIGURATIONS

```
FORWARDHIDDEN<<<4, 32>>>
FORWARDOUTPUT<<<1, 64>>>
COMPUTEHIDDENGRADIENTS<<<16, 4>>>
UPDATEOUTPUTLAYER<<<OUTPUT_SIZE, HIDDEN_SIZE>>>
UPDATEHIDDENLAYER<<<HIDDEN_SIZE, INPUT_SIZE>>>
```

OPTIMIZED GPU VERSION – V3



SHARED MEMORY

Shared memory is used to cache input data, hidden layer activations, and output gradients within each thread block. This reduces global memory access and speeds up forward and backward passes in the neural network.

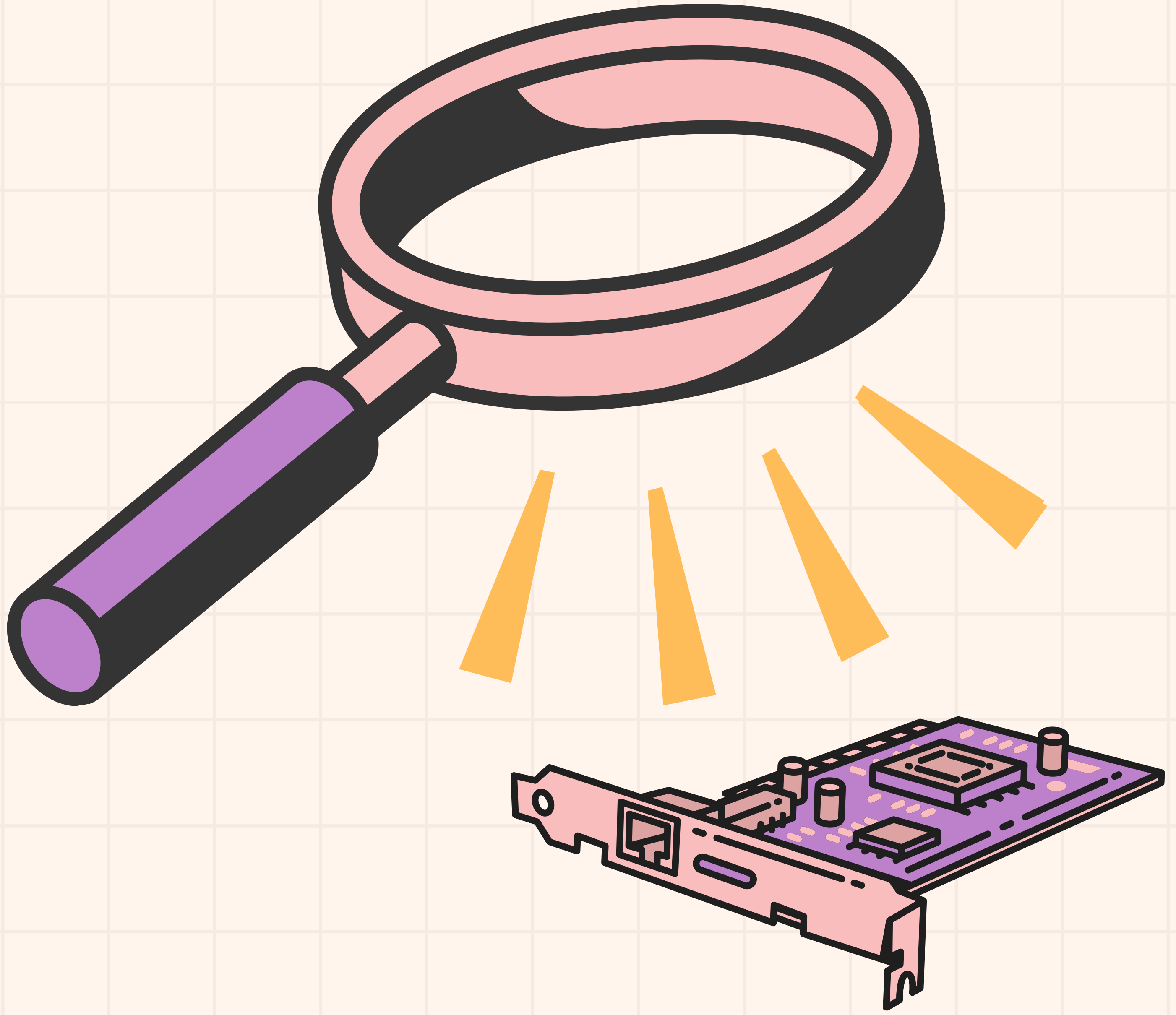
MEMORY COALESCION

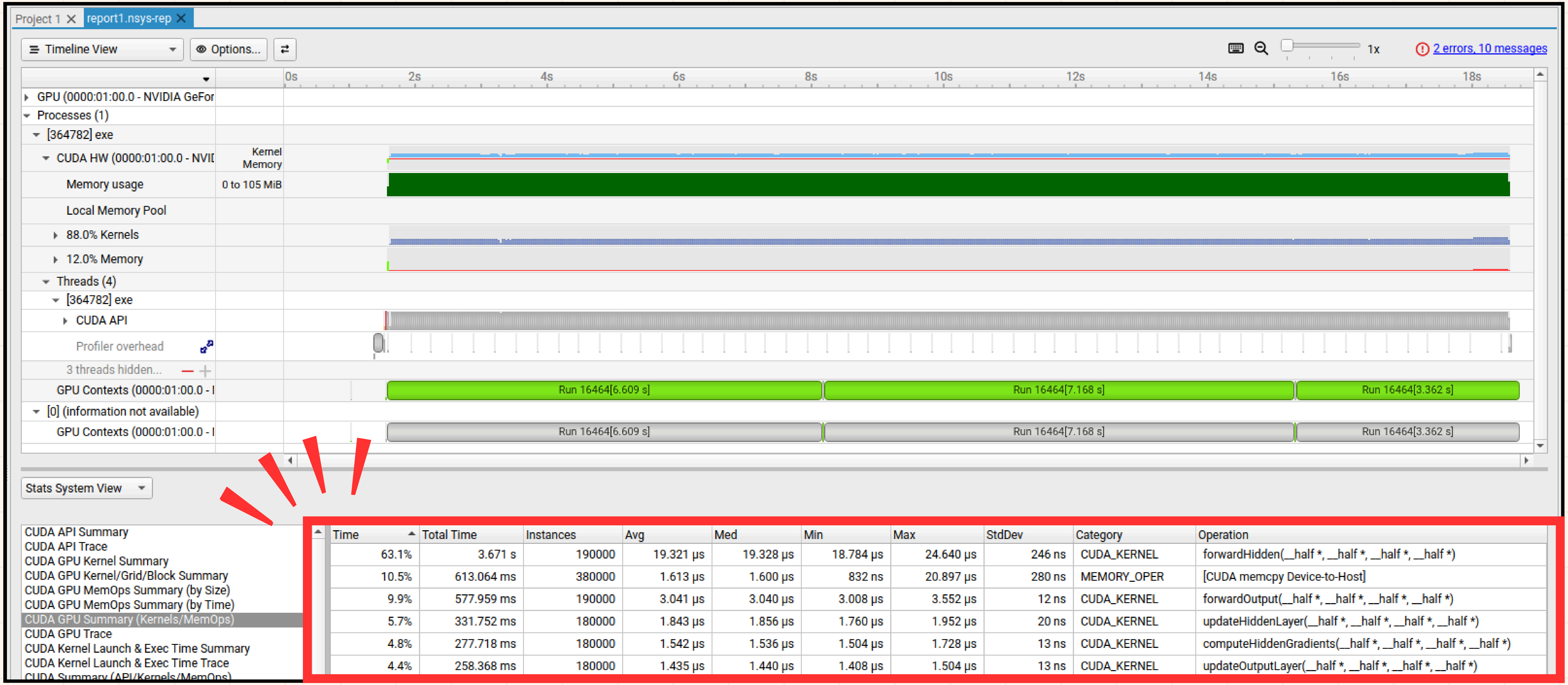
The weight matrix is transposed to ensure memory coalescing during access by parallel threads. This improves global memory access efficiency, reducing latency during matrix multiplications.

FP 16

HALF PRECISION

- Uses half type to store inputs, weights, and activations.
- Functions like __float2half() and __half2float() are used to convert between 32-bit and 16-bit values.
- Reduces memory usage and enables faster computation on supported GPUs.
- Ideal for deep learning workloads where precision trade-offs are acceptable.

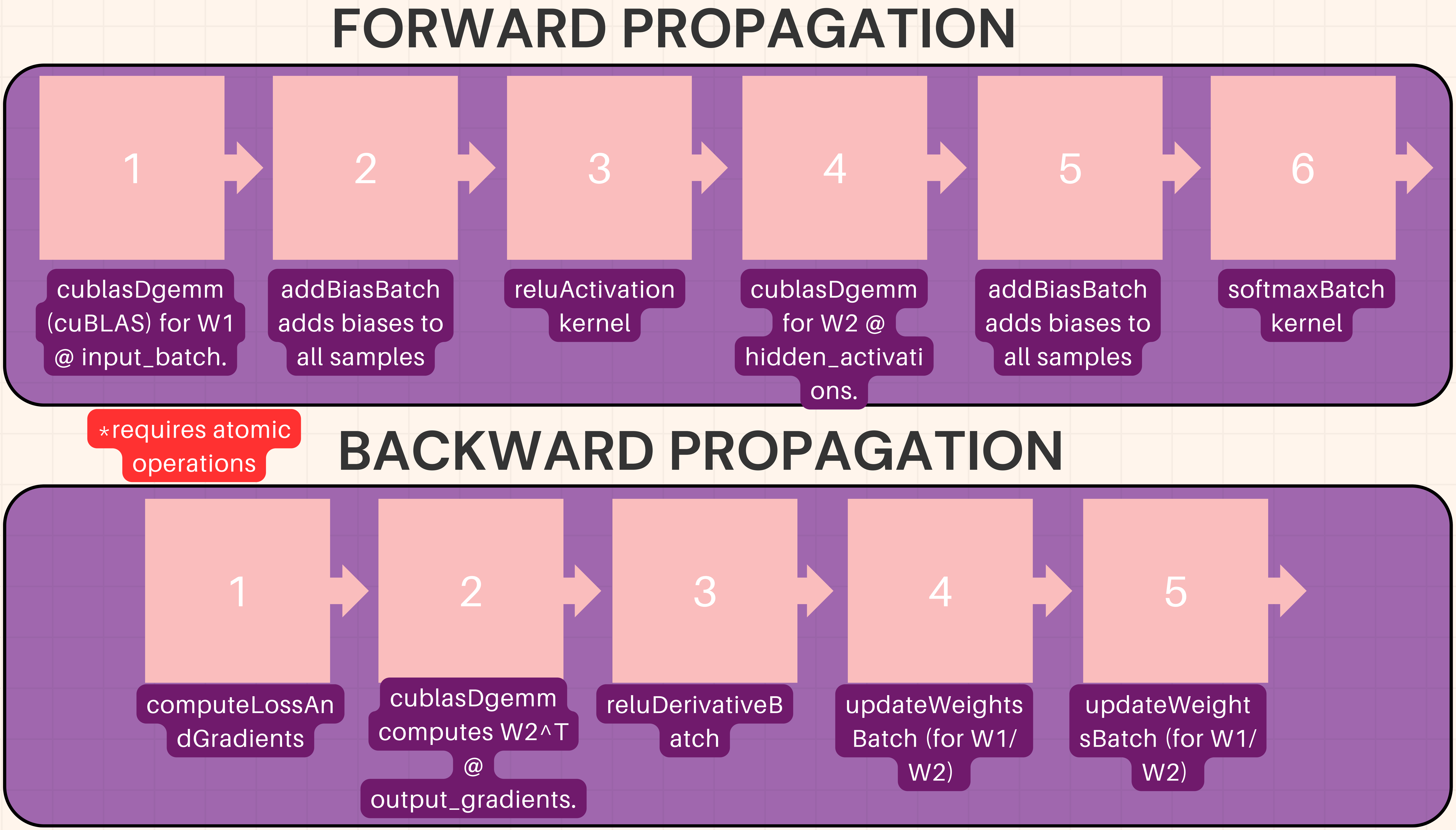


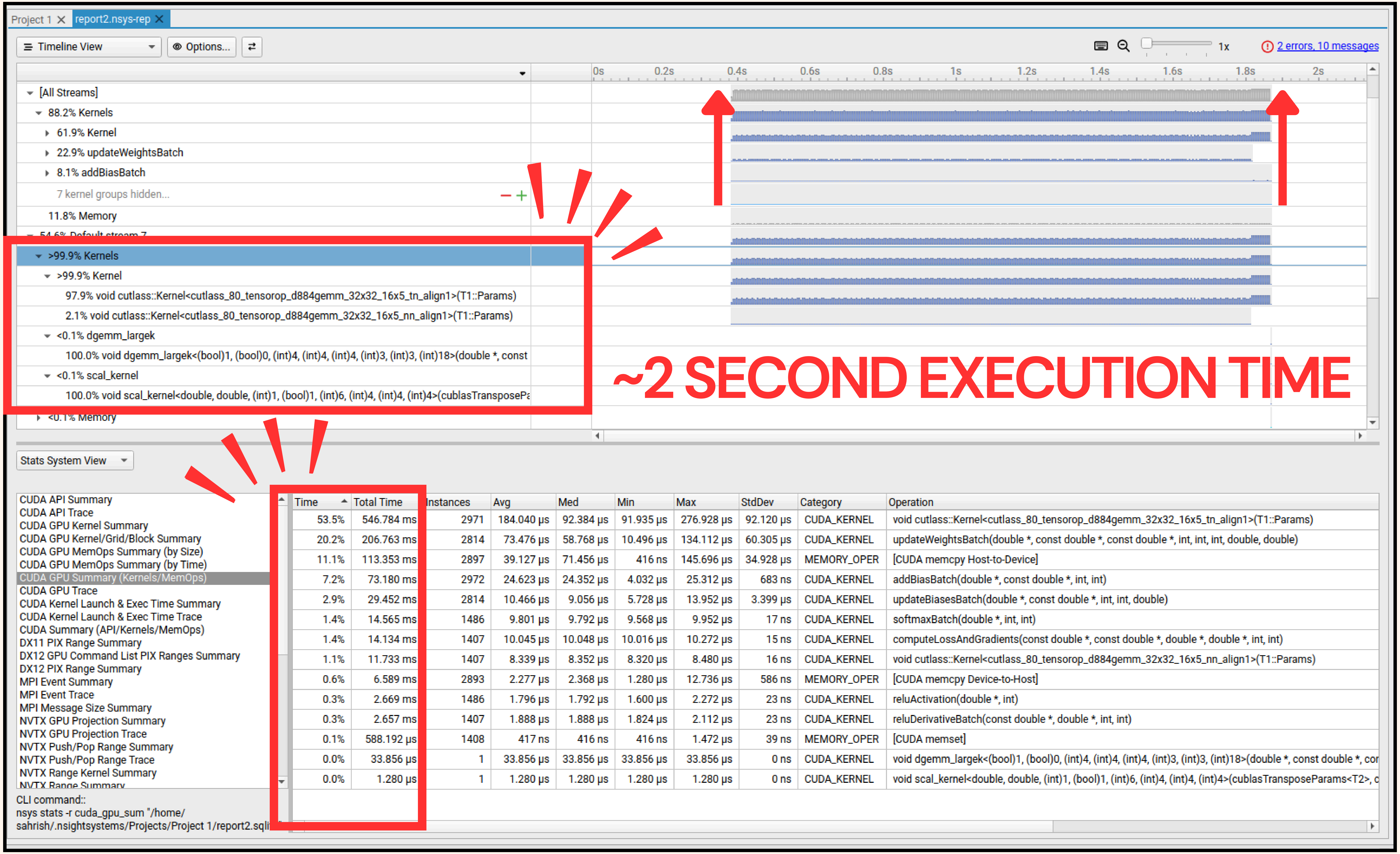


Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
63.1%	3.671 s	190000	19.321 μs	19.328 μs	18.784 μs	24.640 μs	246 ns	CUDA_KERNEL	forwardHidden(__half *, __half *, __half *, __half *)
10.5%	613.064 ms	380000	1.613 μs	1.600 μs	832 ns	20.897 μs	280 ns	MEMORY_OPER	[CUDA memcpy Device-to-Host]
9.9%	577.959 ms	190000	3.041 μs	3.040 μs	3.008 μs	3.552 μs	12 ns	CUDA_KERNEL	forwardOutput(__half *, __half *, __half *, __half *)
5.7%	331.752 ms	180000	1.843 μs	1.856 μs	1.760 μs	1.952 μs	20 ns	CUDA_KERNEL	updateHiddenLayer(__half *, __half *, __half *, __half *)
4.8%	277.718 ms	180000	1.542 μs	1.536 μs	1.504 μs	1.728 μs	13 ns	CUDA_KERNEL	computeHiddenGradients(__half *, __half *, __half *, __half *)
4.4%	258.368 ms	180000	1.435 μs	1.440 μs	1.408 μs	1.504 μs	13 ns	CUDA_KERNEL	updateOutputLayer(__half *, __half *, __half *, __half *)

ACCELERATION PLAN

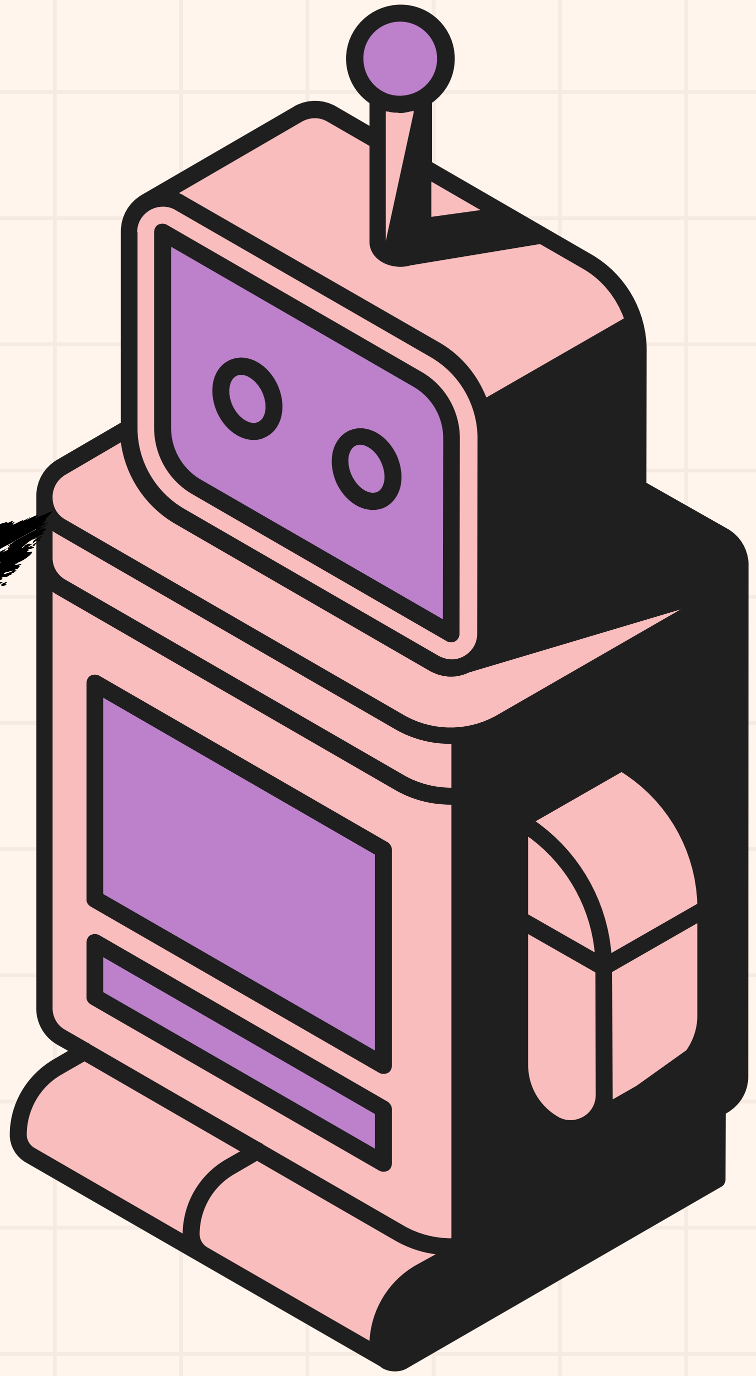
- BATCH PROCESSING
- TENSOR CORES


FORWARD PROPAGATION
1
cublasDgemm (cuBLAS) for W1 @ input_batch.
2
addBiasBatch adds biases to all samples
3
reluActivation kernel
4
cublasDgemm for W2 @ hidden_activations.
5
addBiasBatch adds biases to all samples
6
softmaxBatch kernel
*requires atomic operations
BACKWARD PROPAGATION
1
computeLossAndGradients
2
cublasDgemm computes W2^T @ output_gradients.
3
reluDerivativeBatch
4
updateWeightsBatch (for W1/W2)
5
updateWeightsBatch (for W1/W2)



Time	Total Time	Instances	Avg	Med	Min	Max	StdDev	Category	Operation
53.5%	546.784 ms	2971	184.040 μs	92.384 μs	91.935 μs	276.928 μs	92.120 μs	CUDA_KERNEL	void cutlass::Kernel<cutlass_80_tensorop_d884gemm_32x32_16x5_tn_align1>(T1::Params)
20.2%	206.763 ms	2814	73.476 μs	58.768 μs	10.496 μs	134.112 μs	60.305 μs	CUDA_KERNEL	updateWeightsBatch(double *, const double *, const double *, int, int, int, double, double)
11.1%	113.353 ms	2897	39.127 μs	71.456 μs	416 ns	145.696 μs	34.928 μs	MEMORY_OPER	[CUDA memcpy Host-to-Device]
7.2%	73.180 ms	2972	24.623 μs	24.352 μs	4.032 μs	25.312 μs	683 ns	CUDA_KERNEL	addBiasBatch(double *, const double *, int, int)
2.9%	29.452 ms	2814	10.466 μs	9.056 μs	5.728 μs	13.952 μs	3.399 μs	CUDA_KERNEL	updateBiasesBatch(double *, const double *, int, int, double)
1.4%	14.565 ms	1486	9.801 μs	9.792 μs	9.568 μs	9.952 μs	17 ns	CUDA_KERNEL	softmaxBatch(double *, int, int)
1.4%	14.134 ms	1407	10.045 μs	10.048 μs	10.016 μs	10.272 μs	15 ns	CUDA_KERNEL	computeLossAndGradients(const double *, const double *, double *, double *, int, int)
1.1%	11.733 ms	1407	8.339 μs	8.352 μs	8.320 μs	8.480 μs	16 ns	CUDA_KERNEL	void cutlass::Kernel<cutlass_80_tensorop_d884gemm_32x32_16x5_nn_align1>(T1::Params)
0.6%	6.589 ms	2893	2.277 μs	2.368 μs	1.280 μs	12.736 μs	586 ns	MEMORY_OPER	[CUDA memcpy Device-to-Host]
0.3%	2.669 ms	1486	1.796 μs	1.792 μs	1.600 μs	2.272 μs	23 ns	CUDA_KERNEL	reluActivation(double *, int)
0.3%	2.657 ms	1407	1.888 μs	1.888 μs	1.824 μs	2.112 μs	23 ns	CUDA_KERNEL	reluDerivativeBatch(const double *, double *, int, int)
0.1%	588.192 μs	1408	417 ns	416 ns	416 ns	1.472 μs	39 ns	MEMORY_OPER	[CUDA memset]
0.0%	33.856 μs	1	33.856 μs	33.856 μs	33.856 μs	33.856 μs	0 ns	CUDA_KERNEL	void dgemm_largek<(bool)1, (bool)0, (int)4, (int)4, (int)4, (int)3, (int)3, (int)18>(double *, const double *, cor
0.0%	1.280 μs	1	1.280 μs	1.280 μs	1.280 μs	1.280 μs	0 ns	CUDA_KERNEL	void scal_kernel<double, double, (int)1, (bool)1, (int)6, (int)4, (int)4, (int)4>(cublasTransposeParams<T2>, c

CONCLUSION



THANK YOU