Performance Analysis of Parallelization on Vision Transformers

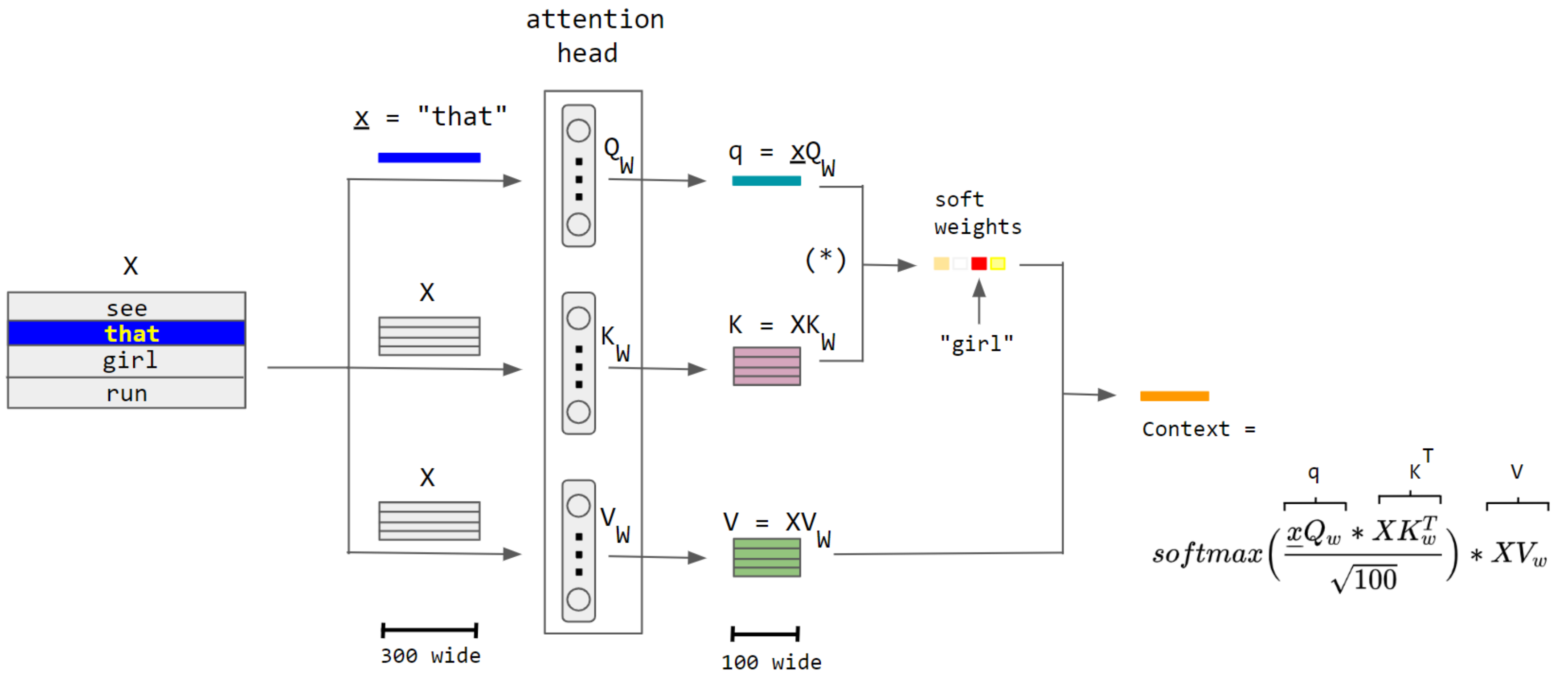
07.06.2023

Mustafa İzzet Muştu

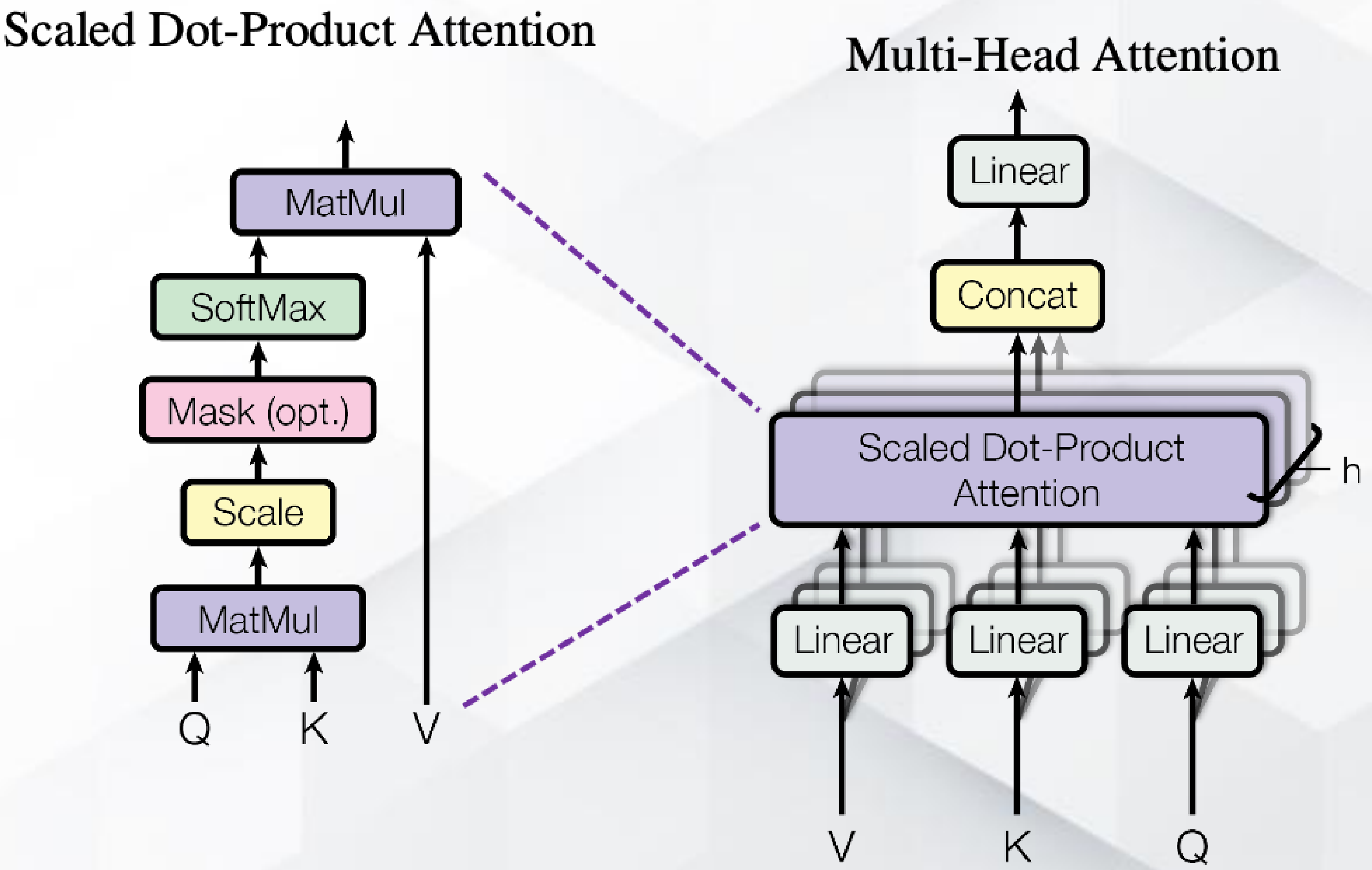
Contents

- Multi-Head Self-Attention
- Transformer
- Vision Transformer (ViT)
- Implementation Details
- Experiments & Results

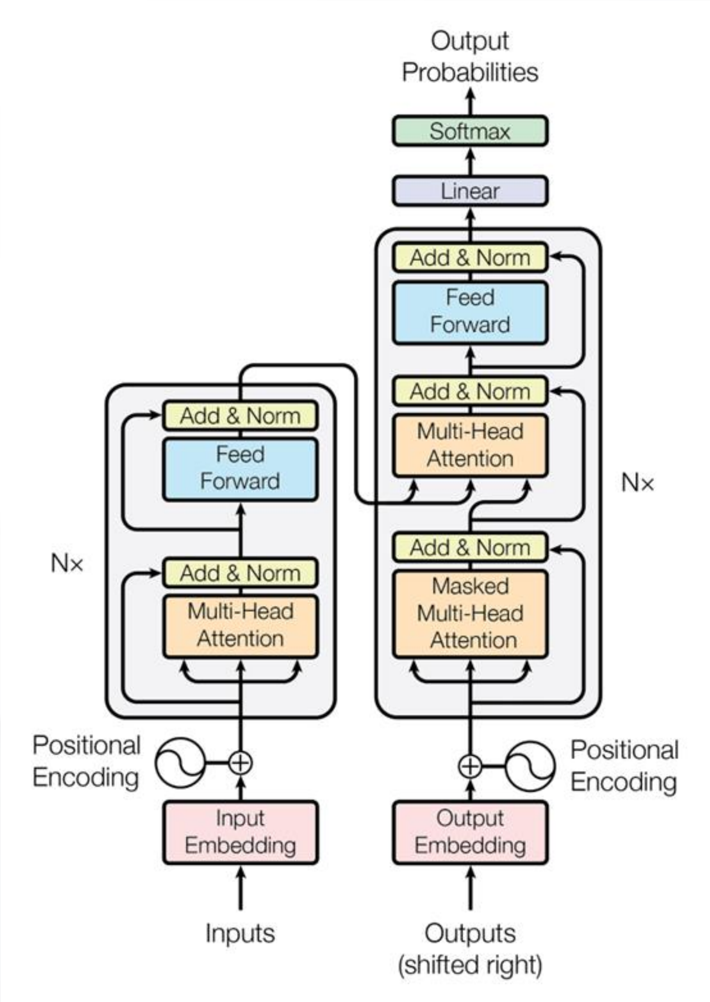
Multi-Head Self-Attention

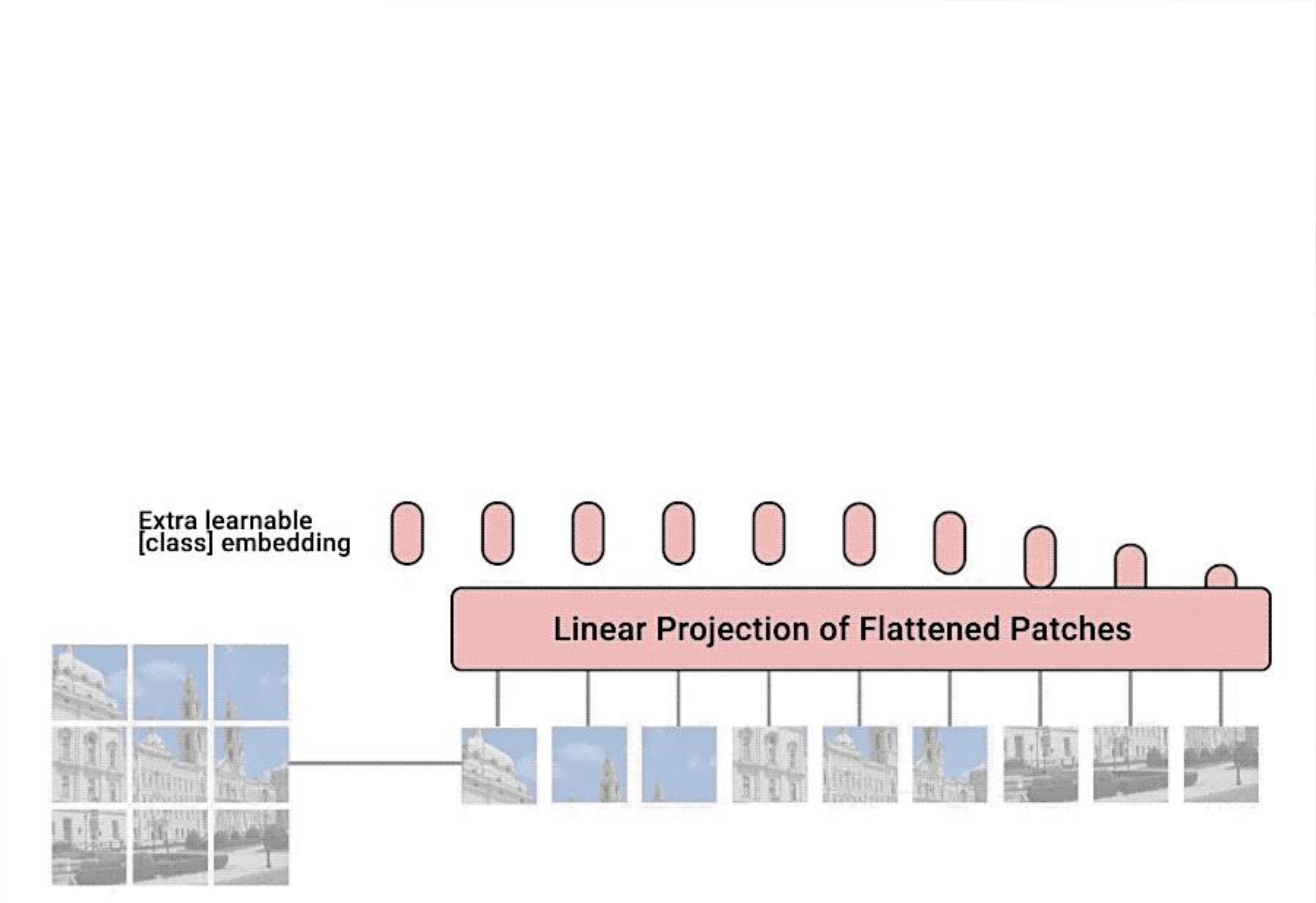


Multi-Head Self-Attention

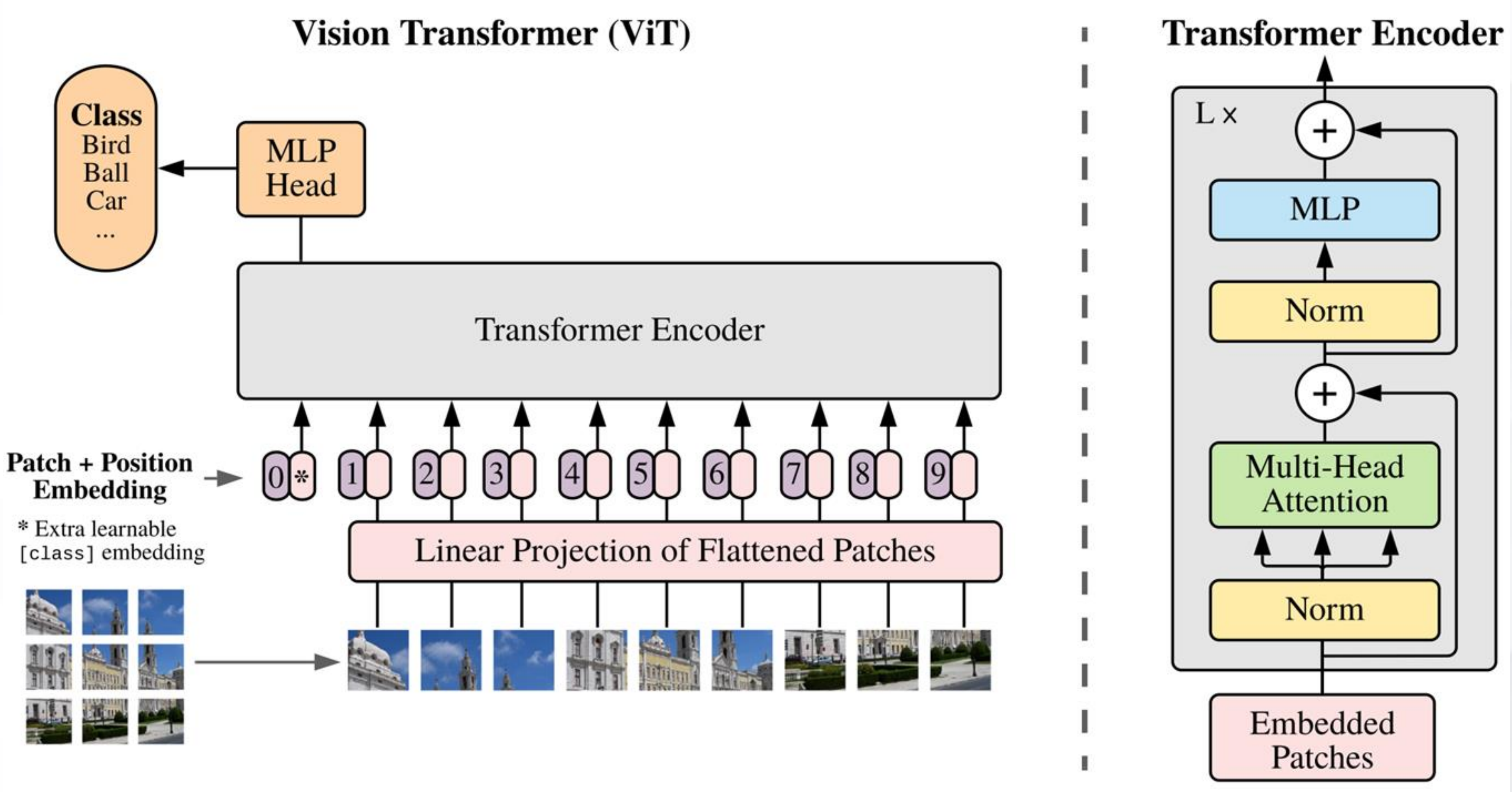


Transformer




Extra learnable
[class] embedding
Linear Projection of Flattened Patches

Vision Transformer (ViT)



Vision Transformer (ViT)

$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}};\, \mathbf{x}_p^1\mathbf{E};\, \mathbf{x}_p^2\mathbf{E};\, \cdots;\, \mathbf{x}_p^N\mathbf{E}] + \mathbf{E}_{pos}, \qquad \mathbf{E} \in \mathbb{R}^{(P^2\cdot C)\times D},\, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1)\times D} \tag{1}$$

$$\mathbf{z'}_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \qquad \ell = 1\ldots L \tag{2}$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z'}_\ell)) + \mathbf{z'}_\ell, \qquad \ell = 1\ldots L \tag{3}$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \tag{4}$$

$x \in \mathbb{R}^{HxWxC} \longrightarrow x_p \in \mathbb{R}^{Nx(P^2C)}$ where (H, W) is the resolution of the original image, C is the number of channels, (P, P) is the resolution of each image patch, and N = HW/P2 is the resulting number of patches.

Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Table 1: Details of Vision Transformer model variants.

Implementation Details

- ViT-Base model without dropout and GELU in MLP
- Eigen Library for matrix representation
- Shared memory and a tiling approach for matrix multiplication
- 12 heads in MSA
- 12 layers in ViT encoder
- 12 CPU thread calls
- 12 CUDA streams
- 12(layers)*12(heads)*5 + 12 multiplication kernel call for attention
- 12(layers)*2 multiplication kernel call for MLP
- 1 multiplication kernel call for embeddings

Implementation Details

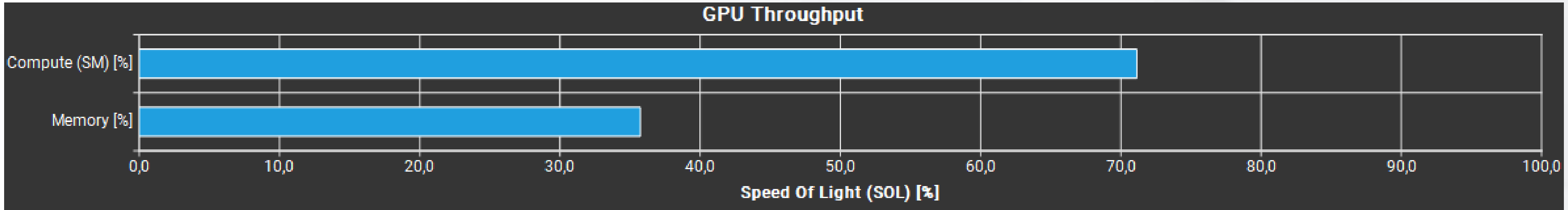
Algorithm 1 Tiled dense matrix-matrix multiplication kernel pseudocode

Require: $A, B, TILE\_WIDTH, Width$

Ensure: $C = AB$

```
sharedA[TILE_WIDTH][TILE_WIDTH]
sharedB[TILE_WIDTH][TILE_WIDTH]
bx ← blockIdx.x
by ← blockIdx.y
tx ← threadIdx.x
ty ← threadIdx.y
row ← ty + blockDim.y · by
column ← tx + blockDim.x · bx
result ← 0
for i = 0, i < (Width-1)/TILE_WIDTH + 1, i += 1 do
    sharedA[ty][tx] ← A[row · Width + i · TILE_WIDTH + tx]
    sharedB[ty][tx] ← B[(i · TILE_WIDTH + ty) · Width + column]
    __syncthreads()
    for j = 0, i < TILE_WIDTH, j += 1 do
        result+ = sharedA[ty][j] · sharedB[j][tx]
    end for
    __syncthreads()
end for
C[row · Width + column] ← result
```

Experiments & Results



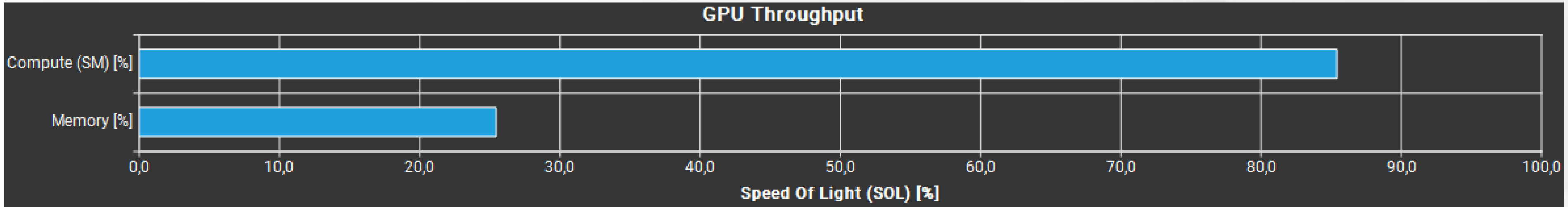
Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	66,67	Block Limit Registers [block]	24
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]	16
Achieved Occupancy [%]	66,06	Block Limit Warps [block]	24
Achieved Active Warps Per SM [warp]	31,71	Block Limit SM [block]	16

Tile width = 8

Experiments & Results

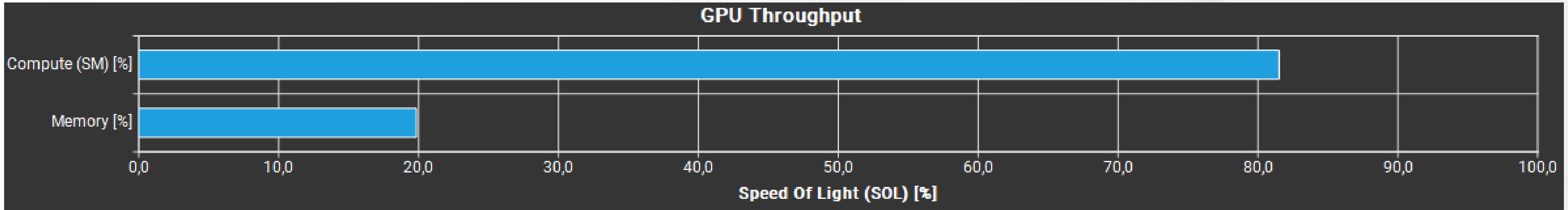


Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	100	Block Limit Registers [block]	6
Theoretical Active Warps per SM [warp]	48	Block Limit Shared Mem [block]	12
Achieved Occupancy [%]	97,38	Block Limit Warps [block]	6
Achieved Active Warps Per SM [warp]	46,74	Block Limit SM [block]	16

Tile width = 16



Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	66,67	Block Limit Registers [block]	1
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]	1
Achieved Occupancy [%]	66,44	Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	31,89	Block Limit SM [block]	16

Tile width = 32

Experiments & Results

TABLE I
EXECUTION TIMES IN 1 LAYER OF THE VIT-BASE ENCODER (CPU).

Operation	Time (s)
Embeddings (Only before the first layer)	1.776
Layer Normalization	0.025
MSA (Multithreaded)	32.626
Residual	0.001
Layer Normalization	0.026
MLP	14.339
Residual	0.002
Layer Normalization (Only after the last layer)	0.025

TABLE II
EXECUTION TIMES IN 1 LAYER OF THE VIT-BASE ENCODER (GPU-CPU).

Operation	Time (s)
Embeddings (Only before the first layer)	**0.011**
Layer Normalization	0.026
MSA (Multithreaded)	**0.037**
Residual	0.001
Layer Normalization	0.026
MLP	**0.007**
Residual	0.002
Layer Normalization (Only after the last layer)	0.026

Performance Analysis of Parallelization on Vision Transformers

Thank you

07.06.2023

Mustafa İzzet Muştu