GPU_basics

December 13, 2025

0.0.1 GPU

GPU(Graphics Processing Unit) - , , 3d .
, , , ,
(, ,).
, , , .
, , .

0.0.2 GPU (,):

-
-
-
- ( )

0.0.3 CPU (,):

-
-
- (,)
-
- (,)

0.0.4 :

- GPU:
- CPU: , ,

0.1

. , ,
. , CPU , ,
.

, GPU : , (,
, : , , GPU 100
). ( ).

, GPU : , .

: GPU – - CPU, ( ),
, .

Nvidia CUDA(Compute Unified Device Architecture) -
, , .

Nvidia A100:

SM(Streaming Multiprocessor) - ,
. : - CUDA - int32, float32/64 - Tensor Core -
- RT-cores - , (. .)

- , (SM), warp- – 32
. warp- . warp – , . . 32 int-a.

Warp Scheduler , warp- Dispatch Unit.(warp-
- – , warp,) Dispatch Unit
, warp. L0 warp-a.

warp- register file - .

: | | | , | |————|———-|—————————| | DRAM |
| 2000+ | | Global memory( ) | , GPU, |
200-400 | | L2 | L1 | 100-200 | | Shared memory(L1) | |
20-30 | | Register File | , warp | 1-2 | | L0 | | 1-2 |

0.2 Numba CUDA -

CUDA . numba:

[1]:
```
from numba import cuda
import numpy as np

cuda.detect()
```

```
Found 1 CUDA devices
id 0             b'Tesla T4'                              [SUPPORTED]
                      Compute Capability: 7.5
                           PCI Device ID: 4
                              PCI Bus ID: 0
                                    UUID:
GPU-517bb309-2f01-17b3-2703-6fa5306ee38c
                                Watchdog: Disabled
             FP32/FP64 Performance Ratio: 32
Summary:
        1/1 devices are supported
```

[1]:
```
True
```

0.2.1

[2]:
```
@cuda.jit
def add_kernel(x, y, out):
    idx = cuda.grid(1)
    if idx < x.size:
        out[idx] = x[idx] + y[idx]
```

[3]:
```
n = 1000000
x = np.ones(n)
y = np.ones(n) * 2
out = np.zeros(n)

threads_per_block = 256
blocks_per_grid = (n + threads_per_block - 1) // threads_per_block

add_kernel[blocks_per_grid, threads_per_block](x, y, out)
print(out[:10])
```

```
[3. 3. 3. 3. 3. 3. 3. 3. 3. 3.]
```

```
/usr/local/lib/python3.12/dist-
packages/numba_cuda/numba/cuda/cudadrv/devicearray.py:937:
NumbaPerformanceWarning: Host array used in CUDA kernel will incur copy overhead
to/from device.
  warn(NumbaPerformanceWarning(msg))
```

0.2.2

1.
2. (if/else)
3.
4.
5. shared memory

0.2.3

, , , .
:

[4]:
```
@cuda.jit
def test_func(out):
    idx = cuda.grid(1)
    tid = cuda.threadIdx.x
    bid = cuda.blockIdx.x
    bdim = cuda.blockDim.x
    gdim = cuda.gridDim.x

    if idx < out.size:
```

```
        out[idx] = tid * 1000 + bid * 100 + bdim * 10 + gdim
```

[5]:
```
out = np.zeros(10)
test_func[2, 4](out)
print(out)
```

```
[  42. 1042. 2042. 3042.  142. 1142. 2142. 3142.    0.    0.]
```

```
/usr/local/lib/python3.12/dist-packages/numba_cuda/numba/cuda/dispatcher.py:697:
NumbaPerformanceWarning: Grid size 2 will likely result in GPU under-utilization
due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
/usr/local/lib/python3.12/dist-
packages/numba_cuda/numba/cuda/cudadrv/devicearray.py:937:
NumbaPerformanceWarning: Host array used in CUDA kernel will incur copy overhead
to/from device.
  warn(NumbaPerformanceWarning(msg))
```

0.2.4 (2D)

[6]:
```
@cuda.jit
def matrix_op(matrix):
    x, y = cuda.grid(2)
    if x < matrix.shape[0] and y < matrix.shape[1]:
        matrix[x, y] = cuda.threadIdx.x * 100 + cuda.threadIdx.y
```

[7]:
```
matrix = np.zeros((8, 8))
matrix_op[(2, 2), (4, 4)](matrix)
print(matrix)
```

```
[[  0.   1.   2.   3.   0.   1.   2.   3.]
 [100. 101. 102. 103. 100. 101. 102. 103.]
 [200. 201. 202. 203. 200. 201. 202. 203.]
 [300. 301. 302. 303. 300. 301. 302. 303.]
 [  0.   1.   2.   3.   0.   1.   2.   3.]
 [100. 101. 102. 103. 100. 101. 102. 103.]
 [200. 201. 202. 203. 200. 201. 202. 203.]
 [300. 301. 302. 303. 300. 301. 302. 303.]]
```

```
/usr/local/lib/python3.12/dist-packages/numba_cuda/numba/cuda/dispatcher.py:697:
NumbaPerformanceWarning: Grid size 4 will likely result in GPU under-utilization
due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
/usr/local/lib/python3.12/dist-
packages/numba_cuda/numba/cuda/cudadrv/devicearray.py:937:
NumbaPerformanceWarning: Host array used in CUDA kernel will incur copy overhead
to/from device.
  warn(NumbaPerformanceWarning(msg))
```

0.2.5

- `cuda.grid(1)` - (1D)
- `cuda.grid(2)` - (x, y) 2D
- `cuda.threadIdx.x`, `cuda.threadIdx.y` -
- `cuda.blockIdx.x`, `cuda.blockIdx.y` -
- `cuda.blockDim.x`, `cuda.blockDim.y` -
- `cuda.gridDim.x`, `cuda.gridDim.y` -

0.3

, – :
GPU, GPU.

0.3.1 Shared memory

(64-256) , .

[8]:
```
@cuda.jit
def sum_reduce(arr, out):
    #      shared memory       256
    shared = cuda.shared.array(256, dtype=np.float32)

    #
    idx = cuda.grid(1)
    tid = cuda.threadIdx.x

    #                          shared memory
    #    :    arr = [1,2,3,4],    shared = [1,2,3,4,0,0,...]
    if idx < arr.size:
        shared[tid] = arr[idx]
    else:
        shared[tid] = 0

    #
    #        :                ,     ,
    cuda.syncthreads()

    #
    #             [1,2,3,4]:
    #    1: [3,2,7,4,0,0,...] (1+2, 3+4)
    #    2: [10,2,7,4,0,0,...] (3+7)
    s = 1
    while s < cuda.blockDim.x:
        if tid % (2 * s) == 0:
            shared[tid] += shared[tid + s]
        s *= 2
        cuda.syncthreads()
```

```
    #
    #       = 10 (            )
    if tid == 0: #     ,
        # out[0] = out[0] + shared[0] -                    -
        cuda.atomic.add(out, 0, shared[0])


n = 1000000
arr = cuda.to_device(np.ones(n, dtype=np.float32))
out = cuda.to_device(np.zeros(1, dtype=np.float32))


threads = 256
blocks = (n + threads - 1) // threads
sum_reduce[blocks, threads](arr, out)

result = out.copy_to_host()
print(f"Sum: {result[0]}")
print(f"Expected: {n}")
```

```
Sum: 1000000.0
Expected: 1000000
```

0.3.2 CPU – GPU

[9]:
```
@cuda.jit
def multiply_by_two(arr):
    idx = cuda.grid(1)
    if idx < arr.size:
        arr[idx] *= 2


x_host = np.ones(1000) #       RAM
x_device = cuda.to_device(x_host) #         GPU

nulls_device = cuda.device_array(1000) #           GPU


threadsperblock = 256
blockspergrid = (x_device.size + (threadsperblock - 1)) // threadsperblock


multiply_by_two[blockspergrid, threadsperblock](x_device)

result = x_device.copy_to_host() #             RAM

print(result[:5])
```

```
[2. 2. 2. 2. 2.]
```

```
/usr/local/lib/python3.12/dist-packages/numba_cuda/numba/cuda/dispatcher.py:697:
NumbaPerformanceWarning: Grid size 4 will likely result in GPU under-utilization
due to low occupancy.
  warn(NumbaPerformanceWarning(msg))
```

, (numba.cuda.cudadrv.devicearray.DeviceNDArray), np.NDArray:

```
arr = np.zeros(5)
func[1, 5](arr)
```

numba - :

```
arr = np.zeros(5)
gpu_zeros = cuda.to_device(arr)
func[1, 5](gpu_zeros)
```

0.4

! , cuda.steam . :

[10]:
```
@cuda.jit
def kernel(x, y):
    idx = cuda.grid(1)
    if idx < x.size:
```

```
        y[idx] = x[idx] * 2

#
stream1 = cuda.stream()
stream2 = cuda.stream()

n = 1000000
x1 = cuda.to_device(np.ones(n))
y1 = cuda.device_array(n)
x2 = cuda.to_device(np.ones(n) * 2)
y2 = cuda.device_array(n)

#
threads = 256
blocks = (n + threads - 1) // threads
kernel[blocks, threads, stream1](x1, y1)
kernel[blocks, threads, stream2](x2, y2)

#
stream1.synchronize()  #          stream1
stream2.synchronize()  #          stream2

#
result1 = y1.copy_to_host()
result2 = y2.copy_to_host()

print("Stream1 result:", result1[:5])
print("Stream2 result:", result2[:5])
```

```
Stream1 result: [2. 2. 2. 2. 2.]
Stream2 result: [4. 4. 4. 4. 4.]
```

: , ,
stream. ,
, - .

0.4.1 CPU vs GPU

:

[11]:
```
@cuda.jit
def add_gpu(x, y, out):
    idx = cuda.grid(1)
    if idx < x.size:
        out[idx] = x[idx] + y[idx]

n = 10000000
x = np.random.rand(n)
```

```
y = np.random.rand(n)

print('CPU (NumPy):')
%timeit x + y

x_d = cuda.to_device(x)
y_d = cuda.to_device(y)
out_d = cuda.device_array(n)
threads = 256
blocks = (n + threads - 1) // threads

print('GPU (CUDA):')
%timeit add_gpu[blocks, threads](x_d, y_d, out_d); cuda.synchronize()
```

```
CPU (NumPy):
28.2 ms ± 568 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
GPU (CUDA):
1.02 ms ± 17.5 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

:

[ ]:
```
n = 100000000
arr = np.random.rand(n)

print('CPU (NumPy sum):')
%timeit np.sum(arr)

arr_d = cuda.to_device(arr.astype(np.float32))
out_d = cuda.to_device(np.zeros(1, dtype=np.float32))
threads = 256
blocks = (n + threads - 1) // threads

print('GPU:')
%timeit sum_reduce[blocks, threads](arr_d, out_d); cuda.synchronize()
```

```
CPU (NumPy sum):
67.9 ms ± 2.24 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
GPU:
9.69 ms ± 26.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

:

[13]:
```
@cuda.jit
def sin_gpu(x, out):
    idx = cuda.grid(1)
    if idx < x.size:
        out[idx] = cuda.libdevice.sin(x[idx])

n = 10000000
```

```
x = np.random.rand(n)

print('CPU (NumPy sin):')
%timeit np.sin(x)

x_d = cuda.to_device(x)
out_d = cuda.device_array(n)
threads = 256
blocks = (n + threads - 1) // threads

print('GPU (CUDA sin):')
%timeit sin_gpu[blocks, threads](x_d, out_d); cuda.synchronize()
```

```
CPU (NumPy sin):
134 ms ± 23.6 ms per loop (mean ± std. dev. of 7 runs, 10 loops each)
GPU (CUDA sin):
1.64 ms ± 25.6 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

[14]:
```
@cuda.jit
def matmul_gpu(A, B, C):
    i, j = cuda.grid(2)
    if i < C.shape[0] and j < C.shape[1]:
        tmp = 0.0
        for k in range(A.shape[1]):
            tmp += A[i, k] * B[k, j]
        C[i, j] = tmp

n = 4096
A = np.random.rand(n, n).astype(np.float32)
B = np.random.rand(n, n).astype(np.float32)

print('CPU (NumPy):')
%timeit np.dot(A, B)

A_d = cuda.to_device(A)
B_d = cuda.to_device(B)
C_d = cuda.device_array((n, n), dtype=np.float32)



print('GPU (CUDA):')
%timeit matmul_gpu[(16, 16), (32, 32)](A_d, B_d, C_d); cuda.synchronize()
```

```
CPU (NumPy):
1.22 s ± 222 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
GPU (CUDA):
52.9 ms ± 18.4 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```