In [ ]:
```
!pip install torch torchvision torchaudio --extra-index-url https://download.pytorc
```

In [1]:
```
import torch
import time

# Matrix dimension
N = 2048

# Create random matrices for CPU (float32)
A_cpu = torch.randn(N, N, dtype=torch.float32)
B_cpu = torch.randn(N, N, dtype=torch.float32)

# Transfer to GPU and convert to half precision for Tensor Core usage FP16 vs FP32
A_gpu = A_cpu.to('cuda').half()
B_gpu = B_cpu.to('cuda').half()

# Warm up GPU: perform a few preliminary multiplications
for _ in range(10):
    _ = torch.matmul(A_gpu, B_gpu)
torch.cuda.synchronize()

# --------------------------
# Benchmark on the CPU
# --------------------------
cpu_iterations = 10
cpu_times = []

for i in range(cpu_iterations):
    start_time = time.time()
    _ = torch.matmul(A_cpu, B_cpu)
    end_time = time.time()
    cpu_times.append(end_time - start_time)

avg_cpu_time = sum(cpu_times) / cpu_iterations

# --------------------------
# Benchmark on the GPU using CUDA events
# --------------------------
gpu_iterations = 100
start_event = torch.cuda.Event(enable_timing=True)
end_event = torch.cuda.Event(enable_timing=True)

# Another warm-up round for GPU kernels
for _ in range(10):
    _ = torch.matmul(A_gpu, B_gpu)
torch.cuda.synchronize()

# Record GPU time
start_event.record()
for i in range(gpu_iterations):
    _ = torch.matmul(A_gpu, B_gpu)
end_event.record()
torch.cuda.synchronize()  # Ensure all GPU ops have finished
```

```
# elapsed_time returns the time in milliseconds
total_gpu_time = start_event.elapsed_time(end_event)
avg_gpu_time = total_gpu_time / gpu_iterations

print(f"Average CPU time per multiplication (s): {avg_cpu_time:.6f}")
print(f"Average GPU time per multiplication (ms): {avg_gpu_time:.6f}")
```

```
Average CPU time per multiplication (s): 0.051557
Average GPU time per multiplication (ms): 0.742380
```

In [2]:
```
x = 0.49552 / 0.000743219
print (f"GPU was : {x:.6f} seconds faster")
x = x / 60
print (f"GPU was : {x:.6f} minutes faster")
```

```
GPU was : 666.721384 seconds faster
GPU was : 11.112023 minutes faster
```

Using the following 2 values Average CPU time per multiplication (s): 0.049552 Average GPU time per multiplication (ms): 0.743219, how much faster was the gpu time Let's calculate how much faster the GPU is compared to the CPU:

Given Values:

Average CPU time: 0.049552 seconds

Average GPU time: 0.743219 milliseconds

Convert GPU time to seconds:

0.743219 ms ÷ 1000 = 0.000743219 seconds Calculate the speedup factor: Divide the CPU time by the GPU time:

Speedup = CPU Time GPU Time = 0.049552 0.000743219 ≈ 66.68 Result: The GPU is approximately 66.68 times faster than the CPU for matrix multiplication in this program. Quite a performance boost, isn't it? Let me know if you'd like further clarification!

In [ ]:
```
# Values in seconds and milliseconds
time_seconds = 0.049552  # in seconds
time_milliseconds = 0.743219  # in milliseconds

# Convert milliseconds to seconds
time_milliseconds_in_seconds = time_milliseconds / 1000

# Calculate the difference
difference = time_seconds - time_milliseconds_in_seconds

# Display the difference
print("Difference between 0.049552 seconds and 0.743219 milliseconds is:")
print(f"{difference:.9f} seconds")
```

Defining the Input Values:

The program starts by defining two variables:

time_seconds, which holds the value 0.049552 (in seconds).

time_milliseconds, which holds the value 0.743219 (in milliseconds).

Converting Milliseconds to Seconds:

Since the two values are given in different units (seconds and milliseconds), we need to convert time_milliseconds into seconds. This is done by dividing the value by 1000, as there are 1000 milliseconds in a second.

The converted value is stored in the variable time_milliseconds_in_seconds.

Calculating the Difference:

The program then calculates the difference between time_seconds and time_milliseconds_in_seconds using a subtraction operation.

The result of this calculation is stored in the variable difference.

Displaying the Result:

Finally, the program prints the difference in a formatted manner using the print() function. The f"{difference:.9f}" part ensures the result is displayed with nine decimal places for precision.

In [ ]:
```
what is the difference between Difference between 0.049552 seconds and 0.743219 mil
Let's calculate the difference:

Start with the values provided:

0.049552 seconds

0.743219 milliseconds

Convert milliseconds to seconds:

Divide 0.743219 milliseconds by 1000 to get seconds: $$0.743219 \, \text{ms} \div 1

Subtract the two values (in seconds): $$0.049552 - 0.000743219 = 0.048808781 \, \te

Final Difference:
The difference is approximately 0.048808781 seconds.
```

GPUs are faster than CPUs for matrix multiplication and similar tasks because they are designed to handle parallelism and computational intensity efficiently. Here's why:

1. **Massive Parallelism:**

   - GPUs have thousands of smaller cores designed for parallel processing. Each core can handle a part of the matrix operation simultaneously. In contrast, CPUs have

fewer cores optimized for sequential, general-purpose tasks.

2. **Specialized Hardware (e.g., Tensor Cores)**:
    - Tensor Cores in modern GPUs, like those from NVIDIA, are specifically designed for matrix operations in half precision (FP16). This hardware provides accelerated performance for deep learning and other numerical tasks.
3. **High Memory Bandwidth**:
    - GPUs have much higher memory bandwidth than CPUs, allowing them to transfer large amounts of data between memory and processing units efficiently. This is critical for operations involving large matrices.
4. **Optimized for Compute-Intensive Tasks**:
    - GPUs are optimized for tasks with a high ratio of computations to memory access (compute-bound tasks). Matrix multiplication involves a lot of computations, making it ideal for GPUs.
5. **Pipeline and Architecture**:
    - CPUs are built for versatility and include features like large caches and advanced branch prediction, which aren't as useful for raw matrix computations. GPUs strip away these features, focusing instead on executing many floating-point operations simultaneously.
6. **Reduced Overhead in Batch Processing**:
    - Tasks like matrix multiplication benefit from batching. GPUs can handle large batches efficiently, minimizing the overhead associated with task scheduling and context switching.

Why Use the GPU's Half Precision?

By using half precision (FP16), as shown in your program, the GPU can process more data with the same resources, further speeding up computations while reducing memory consumption.

In summary, the GPU's architecture, specialized hardware, and optimized processing pipelines make it dramatically faster for matrix operations compared to the CPU, especially when leveraging half precision and Tensor Cores. Let me know if you'd like to dive deeper into any of these topics!

In [ ]: