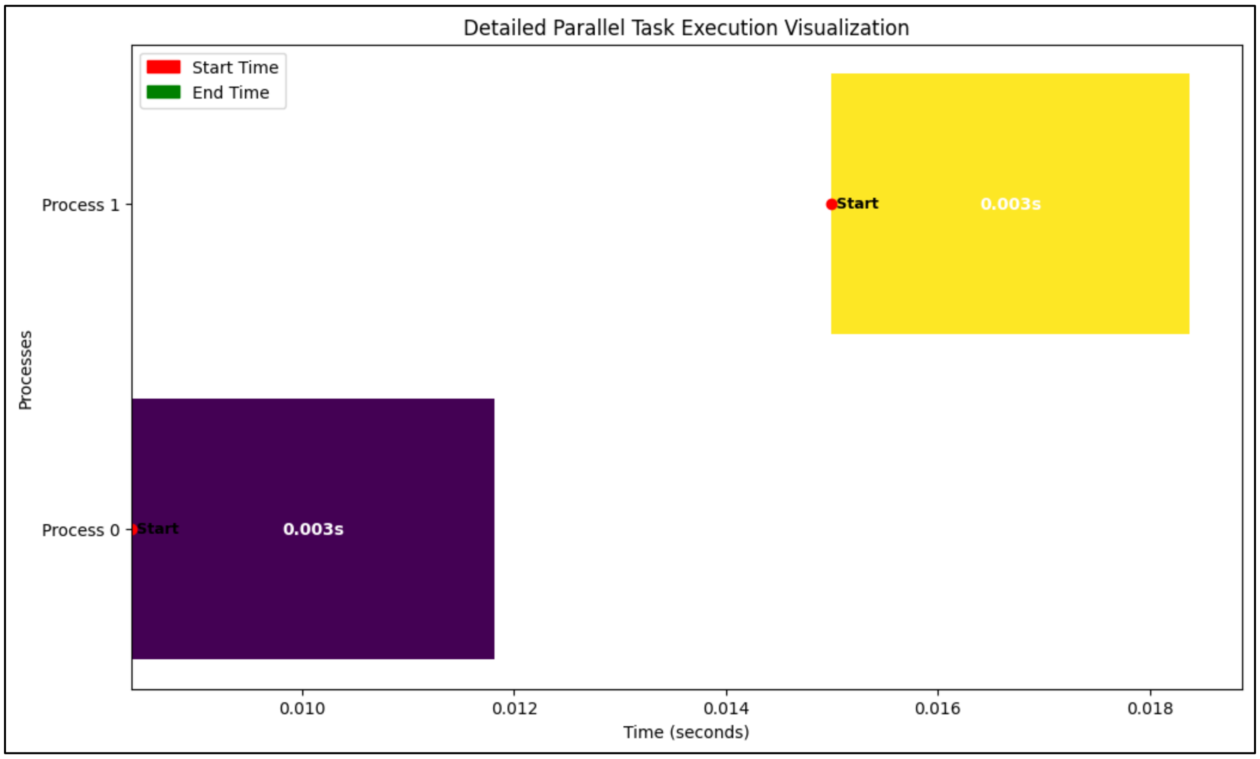
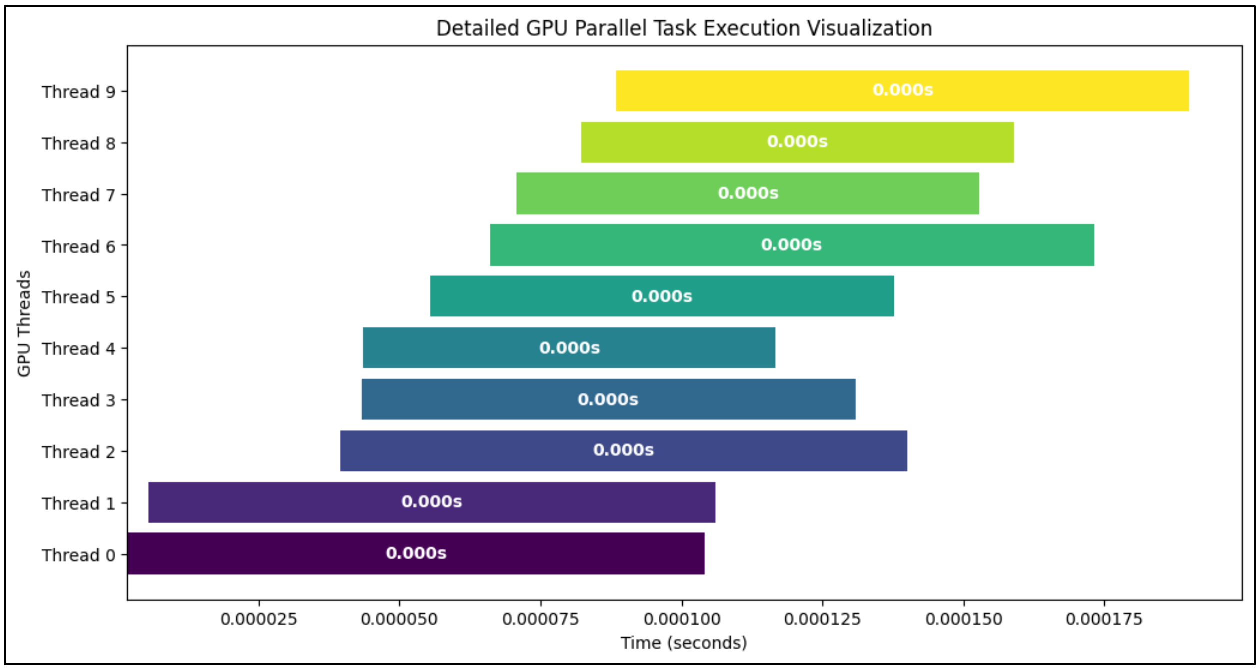
```
#Perform the Task Using CPU, You can see that 1 process is finished before
another starts as CPU perform tasks sequentially.

import multiprocessing as mp
import numpy as np
import time
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

#In this task a sum is calculated by distributing it into chunks of data
and perfrom in parallel
def compute_sum(chunk, process_id, results, timing, timestamps):
    start_time = time.time()
    timestamps[process_id] = [(start_time, "Start")]

    result = np.sum(chunk)  # Compute sum

    end_time = time.time()
    timestamps[process_id].append((end_time, "End"))

    results[process_id] = result
    timing[process_id] = (start_time, end_time)

# visualize parallel processing using CPU
def parallel_sum_visualization():
    NUM_PROCESSES = min(mp.cpu_count(), 8)  # Limit to 8 for better
visualization
    ARRAY_SIZE = 10_000_000  # Large array
    data = np.random.rand(ARRAY_SIZE)  # Generate random data

    chunk_size = ARRAY_SIZE // NUM_PROCESSES
    chunks = [data[i * chunk_size:(i + 1) * chunk_size] for i in
range(NUM_PROCESSES)]

    # Shared memory for results, timing, and timestamps
    manager = mp.Manager()
    results = manager.dict()
    timing = manager.dict()
    timestamps = manager.dict()

    processes = []

    start_global = time.time()
    for i in range(NUM_PROCESSES):
        p = mp.Process(target=compute_sum, args=(chunks[i], i, results,
timing, timestamps))
        processes.append(p)
        p.start()

    for p in processes:
        p.join()
    end_global = time.time()

    # Prepare visualization
    fig, ax = plt.subplots(figsize=(12, 7))
    colors = plt.cm.viridis(np.linspace(0, 1, NUM_PROCESSES))  # Generate
unique colors

    # Plot task execution timelines
    for process_id, (start, end) in timing.items():
        ax.barh(f"Process {process_id}", end - start, left=start -
start_global, color=colors[process_id])

        # Annotate durations
        duration = end - start
        ax.text((start - start_global) + duration / 2, process_id,
f"{duration:.3f}s",
                va='center', ha='center', fontsize=10, color='white',
fontweight='bold')

    # Scatter plot to show task start and end points
    for process_id, events in timestamps.items():
        for event_time, label in events:
            ax.scatter(event_time - start_global, process_id, marker="o",
color="red" if label == "Start" else "green")
            ax.text(event_time - start_global, process_id, f" {label}",
va='center', fontsize=9, fontweight="bold")

    # Labels and legend
    ax.set_xlabel("Time (seconds)")
    ax.set_ylabel("Processes")
    ax.set_title("Detailed Parallel Task Execution Visualization")

    start_patch = mpatches.Patch(color="red", label="Start Time")
    end_patch = mpatches.Patch(color="green", label="End Time")
    plt.legend(handles=[start_patch, end_patch])

    plt.show()
    # Print results
    total_sum = sum(results.values())
    print(f"Total Sum: {total_sum:.2f}")
    print(f"Total Execution Time: {end_global - start_global:.4f}
seconds")

# Run the visualization
if __name__ == "__main__":
    parallel_sum_visualization()
```



```
# Same task perform using GPU, and we can see the difference in execution
time

import torch
import time
import numpy as np
import matplotlib.pyplot as plt

# Check if CUDA is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Generate large random array
size = 10**7  # 10 million elements
arr = torch.randn(size, device=device)

# Measure execution time on GPU
start_time = time.time()
sum_gpu = torch.sum(arr)  # Summation on GPU
end_time = time.time()
gpu_time = end_time - start_time

# Simulate thread execution times (randomized for visualization purposes)
num_threads = 10  # Assuming 10 parallel threads
start_times = np.sort(np.random.uniform(0, gpu_time * 0.5, num_threads))
execution_times = np.random.uniform(gpu_time * 0.4, gpu_time * 0.6,
num_threads)

# Visualization of GPU parallel execution
fig, ax = plt.subplots(figsize=(12, 6))
colors = plt.cm.viridis(np.linspace(0, 1, num_threads))

for i in range(num_threads):
    ax.barh(f"Thread {i}", execution_times[i], left=start_times[i],
color=colors[i])
    ax.text(start_times[i] + execution_times[i] / 2, i,
f"{execution_times[i]:.3f}s",
            ha='center', va='center', color='white', fontsize=10,
fontweight='bold')

ax.set_xlabel("Time (seconds)")
ax.set_ylabel("GPU Threads")
ax.set_title("Detailed GPU Parallel Task Execution Visualization")
plt.show()

# Return GPU execution time
```

```
gpu_time
```


Detailed GPU Parallel Task Execution Visualization
GPU Threads
Thread 9
Thread 8
Thread 7
Thread 6
Thread 5
Thread 4
Thread 3
Thread 2
Thread 1
Thread 0
0.000s
0.000025
0.000050
0.000075
0.000100
0.000125
0.000150
0.000175
Time (seconds)