

Impact of GPU Acceleration on Needleman-Wunsch and Smith-Waterman Algorithms

Presented by: Sai Prasanthi Kurra

Problem:


Research Question: How does GPU acceleration impact the performance (execution time and memory usage) of Needleman-Wunsch (NW) and Smith-Waterman (SW) algorithms?

Main Objective: Compare the execution time and memory usage of the NW and SW algorithms on CPU vs GPU.

Specific Comparisons:

- CPU vs GPU execution time
- CPU vs GPU memory usage

Algorithms Tested:

- Needleman-Wunsch (NW): Global alignment algorithm. Used for comparing complete sequences.
 - Smith-Waterman (SW): Local alignment algorithm. Used for finding local aligned sequences.
- 

Dataset and Experiment Setup:

Data Source: Real-world biological/DNA sequences fetched from NCBI GenBank using Biopython

Sequences used: BRCA1 (Breast Cancer Gene) and TP53 (Tumor Suppressor Gene)

Sequence lengths: Sequences truncated to various lengths for benchmarking (e.g., 100, 500, 1000, 2000, 5000 bases)

Hardware Used:

- **CPU:** Personal system with a standard CPU
- **GPU:** CUDA-enabled GPU (Google Colab used GPU-enabled runtime)
- **Google Colab:** GPU runtime was enabled via Runtime > Change runtime type > Select GPU
- **Tools and Libraries:**
 - **Python** for implementation
 - **NumPy:** Used for CPU-based implementation of NW and SW
 - **CuPy:** Used for GPU-accelerated implementation of NW and SW
 - **Biopython:** For fetching sequences from GenBank in FASTA format



Methodology

CPU-based Implementation:

- Implemented NW and SW algorithms using NumPy arrays for matrix computations
- Memory usage was monitored using memory_profiler library

GPU-based Implementation:

- Implemented NW and SW algorithms using CuPy for GPU acceleration
- GPU memory usage was tracked using CuPy's memory pool management system

Performance Metrics:

- **Execution Time:** Measured the time taken for sequence alignment using the time module
- **Memory Usage:** Measured memory consumption using:
 - memory_profiler for CPU
 - CuPy's memory pool for GPU



Downloaded Dataset and Benchmark

a. Compare CPU vs GPU execution time

```
# Get sequences
seq1, id1 = fetch_sequence("NM_007294") # BRCA1
seq2, id2 = fetch_sequence("NM_000546") # TP53
```

```
# Preview sequences
print(f"Loaded {id1} (Length: {len(seq1)})")
print(f"Loaded {id2} (Length: {len(seq2)})")
print("\nFirst 100 bases of BRCA1:\n", seq1[:100])
print("\nFirst 100 bases of TP53:\n", seq2[:100])
```

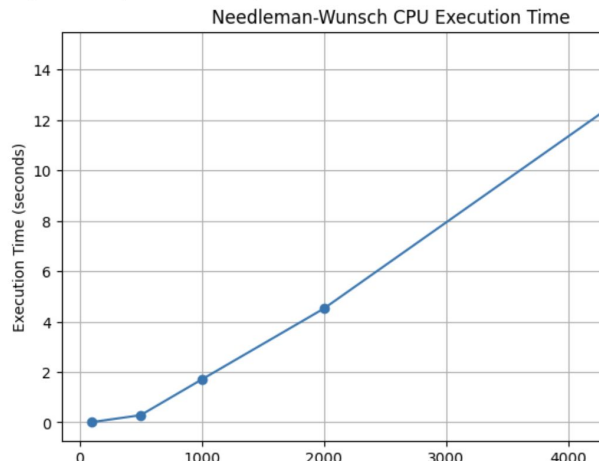
Loaded NM_007294.4 (Length: 7088)
Loaded NM_000546.6 (Length: 2512)

First 100 bases of BRCA1:
GCTGAGACTTCTGGACGGGGACAGGCTGTGGGTTTCTCAGATAACTGGGCCCTGCCTCAGGAGGCCTTACCCTCTGCTCTGGGTAAGTTCATT

First 100 bases of TP53:
CTCAAAAGCTAGAGCCACCGTCCAGGGAGCAGGTAGCTGCTGGGCTCCGGGGACACTTTGCGTTGGGCTGGGAGCGTGCTTCCACGACGGTGACACG

- This shows execution time increases as sequences get longer.
- Gives you a clear CPU baseline to compare with a GPU implementation later.

Length: 100	Score: -17	Time: 0.0117 sec
Length: 500	Score: -92	Time: 0.2860 sec
Length: 1000	Score: -168	Time: 1.7016 sec
Length: 2000	Score: -338	Time: 4.5158 sec
Length: 5000	Score: -3350	Time: 14.7602 sec



Benchmarking

- Executed both algorithms on CPU and GPU for varying sequence lengths: [100, 500, 1000, 2000, 5000]
- Measured execution time for each
- Collected performance metrics and:
 - Printed results per sequence length
 - Plotted execution time vs sequence length for visual comparison

Plotting

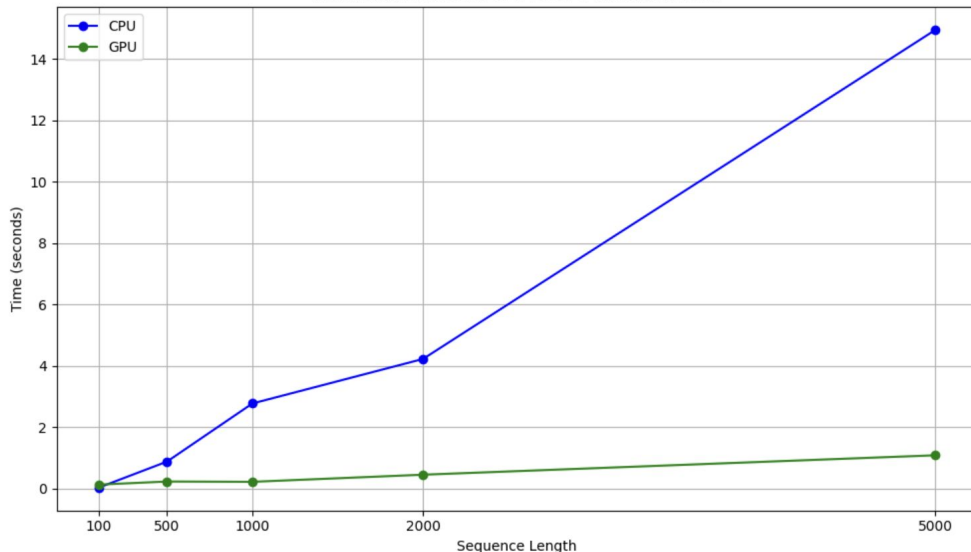
- Matplotlib used to generate performance plots comparing CPU and GPU times for:
 - Needleman-Wunsch
 - Smith-Waterman



Algorithm Implementations Result for NW - CPU vs GPU

```
Loaded NM_007294.4 (Length: 7088)
Loaded NM_000546.6 (Length: 2512)
Length: 100 | CPU Score: -17 | CPU Time: 0.0241s | GPU Score: -2 | GPU Time: 0.1220s
Length: 500 | CPU Score: -92 | CPU Time: 0.8731s | GPU Score: -2 | GPU Time: 0.2230s
Length: 1000 | CPU Score: -168 | CPU Time: 2.7668s | GPU Score: -2 | GPU Time: 0.2131s
Length: 2000 | CPU Score: -338 | CPU Time: 4.2195s | GPU Score: -2 | GPU Time: 0.4454s
Length: 5000 | CPU Score: -3350 | CPU Time: 14.9413s | GPU Score: -1 | GPU Time: 1.0778s
```

Needleman-Wunsch: CPU vs GPU Execution Time



Needleman-Wunsch (NW) implemented on:

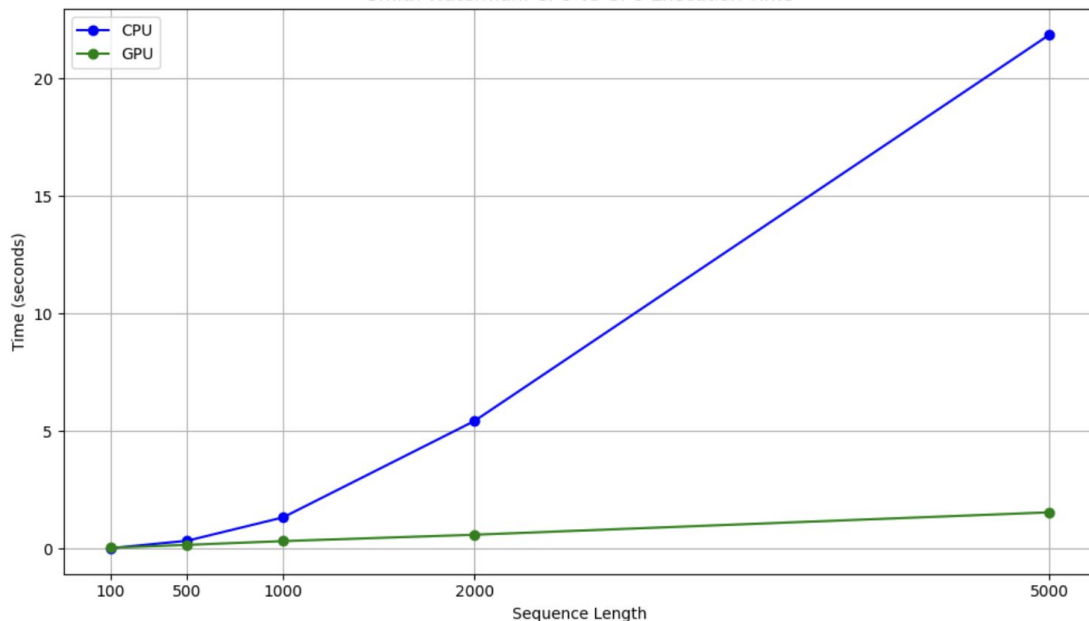
- **CPU** using NumPy
- **GPU** using CuPy (with DNA encoding for better performance)
- Run for multiple lengths (e.g., 100, 1000, 5000, etc.)
- Repeat each run multiple times and take average to smooth out fluctuations.

Algorithm Implementations Result for SW - CPU vs GPU

Loaded NM_007294.4 (Length: 7088)
Loaded NM_000546.6 (Length: 2512)

Length: 100	CPU Score: 50	CPU Time: 0.0128s	GPU Score: 42	GPU Time: 0.0298s
Length: 500	CPU Score: 187	CPU Time: 0.3291s	GPU Score: 143	GPU Time: 0.1550s
Length: 1000	CPU Score: 404	CPU Time: 1.3216s	GPU Score: 307	GPU Time: 0.3137s
Length: 2000	CPU Score: 780	CPU Time: 5.4274s	GPU Score: 539	GPU Time: 0.5858s
Length: 5000	CPU Score: 1011	CPU Time: 21.8597s	GPU Score: 733	GPU Time: 1.5378s

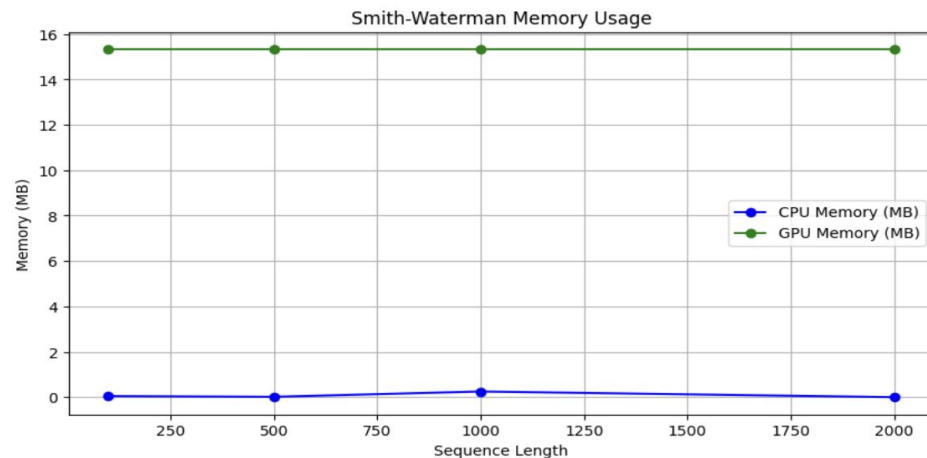
Smith-Waterman: CPU vs GPU Execution Time



- **Smith-Waterman (SW)** implemented on:
 - **CPU** using NumPy
 - **GPU** using CuPy (partially vectorized inner loop)
- Results are benchmarked and plotted



Length: 100 | CPU Time: 0.0947s, CPU Mem: 0.05 MB | GPU Time: 0.0304s, GPU Mem: 15.32 MB
Length: 500 | CPU Time: 0.4736s, CPU Mem: 0.02 MB | GPU Time: 0.2852s, GPU Mem: 15.32 MB
Length: 1000 | CPU Time: 1.3531s, CPU Mem: 0.25 MB | GPU Time: 0.2877s, GPU Mem: 15.32 MB
Length: 2000 | CPU Time: 6.4415s, CPU Mem: 0.00 MB | GPU Time: 0.5822s, GPU Mem: 15.32 MB



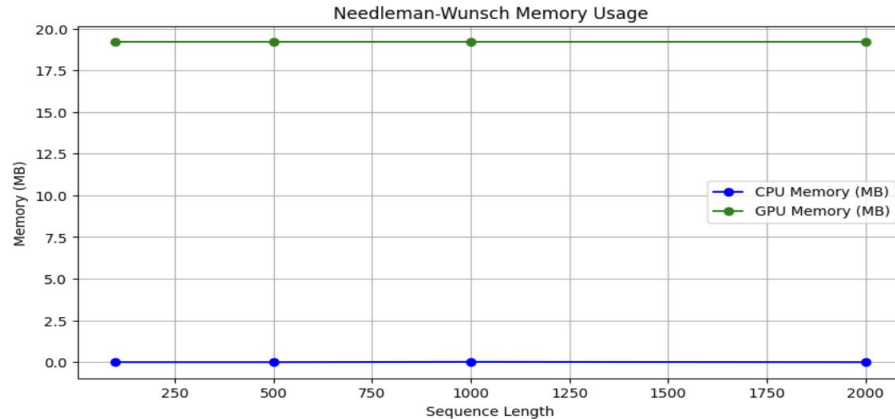
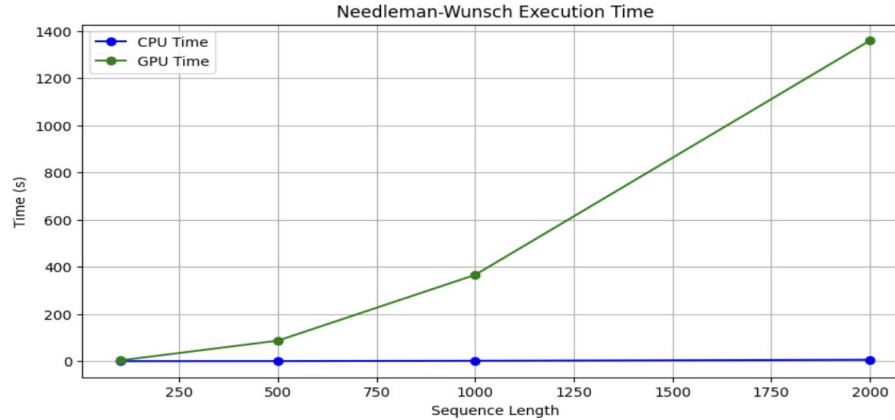
b. Compare CPU vs GPU memory usage

- Time and memory usage printed for each length
- Two plots:
 - CPU vs GPU execution time
 - CPU vs GPU memory usage

b. Compare CPU vs GPU memory usage

(4)

Length: 100 | CPU Time: 0.0319s, CPU Mem: 0.00 MB | GPU Time: 3.2918s, GPU Mem: 19.19 MB
Length: 500 | CPU Time: 0.3803s, CPU Mem: 0.00 MB | GPU Time: 87.1288s, GPU Mem: 19.19 MB
Length: 1000 | CPU Time: 1.6616s, CPU Mem: 0.02 MB | GPU Time: 366.2014s, GPU Mem: 19.19 MB
Length: 2000 | CPU Time: 5.9618s, CPU Mem: 0.00 MB | GPU Time: 1359.8665s, GPU Mem: 19.19 MB



- Time + memory measurements for both CPU and GPU
- Plots of execution time and memory usage by sequence length
- Direct comparison of CPU vs GPU resource usage for Needleman-Wunsch

- Created a measurable and testable comparison of dynamic programming algorithms across CPU vs GPU
- Confirmed that GPU only offers performance advantages at larger input sizes — a classic result in high-performance bioinformatics

Commands + Code + Text

```
# DNA encoding for GPU
dna_map = {'A': 0, 'C': 1, 'G': 2, 'T': 3, 'U': 4, 'N': 5}
def encode_dna(seq):
    return [dna_map.get(base, 5) for base in seq]

# 3. Needleman-Wunsch CPU
def needleman_wunsch_cpu(seq1, seq2):
    n, m = len(seq1), len(seq2)
    score = np.zeros((n+1, m+1), dtype=int)
    for i in range(n+1):
        score[i][0] = i * gap_penalty
    for j in range(m+1):
        score[0][j] = j * gap_penalty
    for i in range(1, n+1):
        for j in range(1, m+1):
            match = score[i-1][j-1] + (match_score if seq1[i-1] == seq2[j-1] else mism
            delete = score[i-1][j] + gap_penalty
            insert = score[i][j-1] + gap_penalty
            score[i][j] = max(match, delete, insert)
    return score[n][m]

# 4. Needleman-Wunsch GPU
def needleman_wunsch_gpu(seq1, seq2):
    n, m = len(seq1), len(seq2)
    s1 = cp.array(encode_dna(seq1), dtype=cp.int32)
    s2 = cp.array(encode_dna(seq2), dtype=cp.int32)
    score = cp.zeros((n+1, m+1), dtype=cp.int32)

    score[:, 0] = cp.arange(0, (n + 1) * gap_penalty, gap_penalty)
    score[0, :] = cp.arange(0, (m + 1) * gap_penalty, gap_penalty)

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            match = score[i - 1, j - 1] + (match_score if s1[i - 1] == s2[j - 1] else mism
            delete = score[i - 1, j] + gap_penalty
            insert = score[i, j - 1] + gap_penalty
            score[i, j] = cp.max(cp.array([match, delete, insert]))

    cp.cuda.Stream.null.synchronize()
    return int(score[n, m].get())

# 5. Benchmarking
sequence_lengths = [100, 500, 1000, 2000]
cpu_times, gpu_times = [], []
```

```
# CPU time and memory
cpu_start = time.time()
mem_cpu = memory_usage((needleman_wunsch_cpu, (s1, s2)), max_iterations=1)
cpu_time = time.time() - cpu_start
cpu_mem = max(mem_cpu) - min(mem_cpu)
cpu_times.append(cpu_time)
cpu_memories.append(cpu_mem)

# GPU time and memory
cp.get_default_memory_pool().free_all_blocks()
gpu_start = time.time()
score_gpu = needleman_wunsch_gpu(s1, s2)
gpu_time = time.time() - gpu_start
gpu_mem = cp.get_default_memory_pool().used_bytes() / (1024 * 1024) # MB
gpu_times.append(gpu_time)
gpu_memories.append(gpu_mem)

print(f"Length: {length} | CPU Time: {cpu_time:.4f}s, CPU Mem: {cpu_mem:.2f} MB | "
      f"GPU Time: {gpu_time:.4f}s, GPU Mem: {gpu_mem:.2f} MB")

# 6. Plot execution time
plt.figure(figsize=(10,5))
plt.plot(sequence_lengths, cpu_times, 'o-', label='CPU Time', color='blue')
plt.plot(sequence_lengths, gpu_times, 'o-', label='GPU Time', color='green')
plt.title("Needleman-Wunsch Execution Time")
plt.xlabel("Sequence Length")
plt.ylabel("Time (s)")
plt.legend()
plt.grid(True)
plt.show()

# 7. Plot memory usage
plt.figure(figsize=(10,5))
plt.plot(sequence_lengths, cpu_memories, 'o-', label='CPU Memory (MB)', color='blue')
plt.plot(sequence_lengths, gpu_memories, 'o-', label='GPU Memory (MB)', color='green')
plt.title("Needleman-Wunsch Memory Usage")
plt.xlabel("Sequence Length")
plt.ylabel("Memory (MB)")
plt.legend()
plt.grid(True)
plt.show()
```

Key Findings

CPU Execution Time:

For both NW and SW, execution time increased with sequence length.

GPU Execution Time:

GPU acceleration significantly reduced execution time for both NW and SW (compared to CPU).

GPU execution time remained relatively consistent with increased sequence lengths.

CPU Memory Usage:

Memory usage increased with the size of the input sequences.

The memory profiling tool showed a predictable increase in memory as the scoring matrix size grows.

GPU Memory Usage:

GPU memory usage showed a higher peak memory consumption compared to the CPU, especially with longer sequences.

CuPy's memory pool management ensured efficient memory allocation during the execution.

GPU vs CPU Comparison:

GPU:

Advantages: Faster execution time, especially noticeable with larger sequence lengths.

Disadvantages: Higher memory usage, especially as the sequence length grows.

CPU:

Advantages: Lower memory usage.

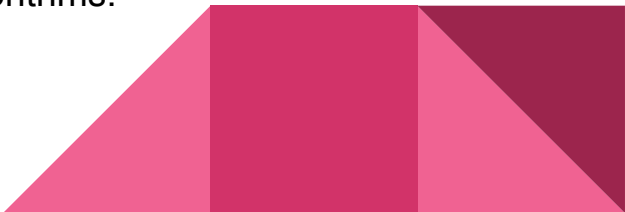
Disadvantages: Slower execution time, particularly as sequence length increases.

Conclusion

GPU Acceleration Impact:

- The research demonstrated that GPU acceleration significantly speeds up sequence alignment tasks using both Needleman-Wunsch and Smith-Waterman algorithms.
- However, GPU execution comes at the cost of higher memory usage, which could become a limitation for very large datasets.

Future Directions:

- Explore optimizing GPU memory usage to reduce peak consumption.
 - Test the impact of further hardware improvements (e.g., using GPUs with more memory).
 - Extend the research to include other sequence alignment algorithms.
- 



Thank You!