

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

Project Proposal & Background:

This project focuses on predicting RNA secondary structures for sequences from the Astroviridae viral family. These RNA viruses are known to cause infections in humans and animals. In their genome, two important regions called ORF1b and ORF2 often overlap. These overlapping areas can contain important information about how the virus replicates and produces proteins. I selected nearly 500 sequences by checking their accession IDs and extracting the RNA segments that include the ORF1b–ORF2 overlaps. The main goal is to predict the secondary structure of these RNA segments to better understand their complexity. However, running structure predictions for hundreds of sequences using traditional (serial) methods takes a lot of time, which creates a major computational bottleneck. This is the problem I aimed to solve using parallel programming.

Motivation:

To build the dataset, I manually explored NCBI GenBank files and collected sequences that covered the overlapping region between ORF1b and ORF2. These regions are functionally important because they can affect how the virus makes proteins. I slightly expanded the regions on both sides to better capture any nearby folding interactions. I collected around 500 RNA sequences and saved them in a standard FASTA file format, where each entry includes an accession ID and a 50-nucleotide RNA segment. Once the dataset was ready, I used RNAfold to predict the secondary structure of each sequence.

However, doing this one by one takes a long time. So I wanted to try parallel processing to speed things up. I was curious to see how much time could be saved by using multiple CPU cores and how well the process would scale with more data. This approach could also be helpful for researchers working with larger datasets who need results faster.

Literature Review / Background:

RNAfold is a commonly used tool in bioinformatics for predicting RNA secondary structures. It uses thermodynamic rules to calculate the Minimum Free Energy (MFE) structure, which is the most stable way an RNA molecule can fold. The output includes a dot-bracket notation representing the structure and the MFE value in kcal/mol, which shows how stable the structure is. These results can be useful for understanding RNA function and potential translation control mechanisms.

I used RNAfold from the command line, but instead of entering sequences manually, I created a Python script that reads the FASTA file and sends each sequence to RNAfold automatically. For

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

this, I used Python's subprocess module. Because running RNAfold 500 times in a row is slow, I added parallel processing using Python's multiprocessing.Pool. This allowed the program to run multiple RNAfold predictions at the same time using separate CPU cores. Each process runs independently and does not depend on the others, which makes it a good fit for parallelization. This method is an example of shared-memory parallel programming, where all processes run on the same machine and work on their own piece of the data.

```
[ilnu001@h5 results]$ head ../results/parallel_results.csv
ID,Sequence,Structure,MFE
NC_001943_SEQ1,UUCGAACGCCAGAUAGAAUAGUUCUAAGGCAUAGUGUCACUUAACAUCAC,...(((((.....)))..))..((((.....)))... (-6.10
NC_002469_SEQ2,CAGCCGAUCUUAUGCCUUGGGUGUCCUUAACGGACGUCAACGUGUCCUAG,...(((.(((.((((.....)))..)))..)))..((-8.70
NC_002470_SEQ3,CCCGCUAGGGCGUGCUGUUUGAUUUUGGUACACCUAGUAGGGCACAAG,...((.....)).((((.....)))..((((.....)))..(-12.40
NC_013443_SEQ4,UGAGCUAUCACCAUUGAGGGCGUGGACCUAGCUUUAUGUCGGCAAAAUU,...(((.....)))..(((.....)))..(((.....)))..((-6.90
NC_013444_SEQ5,CGUUCUCCCCAAUUGUGGGUGUUAAGCGUUGGGUACCUAGGGCCUCUGGG,...((((.....)).(((.....)))..((((.....)))..((((.....)))..(-10.20
NC_025217_SEQ6,GGAGAAGCCGCAUUCGGGCUUAGCACAAGGCGCUCGUUCACAGAGUUGG,...(.((((.....)))..).(((.....)))..(((.....)))..(-15.30
NC_012795_SEQ7,CGUUGAAAGCGUGAAAGUAUGUAGGAAACAUAUGGUACAUAUGUAUCGAA,...((((.....)))..((.....))..((((.....)))..((-8.10
NC_019441_SEQ8,AUAUCUCGCUAAUAGAUUAUAGACAGAAACCGGAUCUCGGUUUAACUGC,...((((.....)))..((((.....)))..((((.....)))..((-7.30
NC_019442_SEQ9,UUAGGACUUUAUGGCCCGGUGUGUGGCGUGGCAAGCACCACUUAACG,...((.....))...(((((((.....)))..)))..(-11.90
```

Figure 1 : Sample of RNA secondary structure predictions generated using RNAfold on sequences from the Astroviridae family.

Baseline Serial Implementation:

```
ilnu001@h5:~/courses/csci46' X + v
GNU nano 2.9.8 run_serial_rnafold.py

import time
import subprocess
import csv
from pathlib import Path

def run_rnafold_serial(input_fasta, output_csv):
    sequences = []

    # Read the FASTA file
    with open(input_fasta, 'r') as f:
        lines = f.readlines()
        for i in range(0, len(lines), 2):
            seq_id = lines[i].strip().replace(">", "")
            seq = lines[i+1].strip()
            sequences.append((seq_id, seq))

    results = []

    for seq_id, seq in sequences:
        # Run RNAfold for each sequence
        process = subprocess.Popen(["RNAfold"], stdin=subprocess.PIPE,
                                    stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
        stdout, _ = process.communicate(input=seq)

        lines = stdout.strip().split('\n')
        if len(lines) >= 2:
            structure_line = lines[1]
            structure, mfe = structure_line.rsplit(' ', 1)

    # Read 46 lines ]
```

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' × + v
GNU nano 2.9.8 run_serial_rnafold.py

    sequences.append((seq_id, seq))

results = []

for seq_id, seq in sequences:
    # Run RNAfold for each sequence
    process = subprocess.Popen(["RNAfold"], stdin=subprocess.PIPE,
                                stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    stdout, _ = process.communicate(input=seq)
    lines = stdout.strip().split('\n')
    if len(lines) >= 2:
        structure_line = lines[1]
        structure, mfe = structure_line.rsplit(' ', 1)
        mfe = mfe.strip("(")")
        results.append((seq_id, seq, structure, mfe))

# Save to CSV
with open(output_csv, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["ID", "Sequence", "Structure", "MFE"])
    writer.writerows(results)

print(f"Serial RNAfold completed. Results saved to:" + output_csv)

if __name__ == "__main__":
    start = time.time()
    Path("../results").mkdir(exist_ok=True)

    mfe = mfe.strip("(")")
    results.append((seq_id, seq, structure, mfe))

# Save to CSV
with open(output_csv, 'w', newline='') as csvfile:
    writer = csv.writer(csvfile)
    writer.writerow(["ID", "Sequence", "Structure", "MFE"])
    writer.writerows(results)

print(f"Serial RNAfold completed. Results saved to:" + output_csv)

if __name__ == "__main__":
    start = time.time()
    Path("../results").mkdir(exist_ok=True)
    run_rnafold_serial("../data/Allastroviridae_family_sequences.fasta", "../results/serial_results.csv")
    print("Total serial time:", round(time.time() - start, 2), "seconds")
```

To start the project, I wrote a basic version of the RNA structure prediction program using Python. This version runs the predictions one by one, without using any parallel processing. The main goal was to see how long it takes to process all the sequences using a single core. This would later help me compare how much faster the parallel version performs.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

In this serial script (`run_serial_rnafold.py`), I first loaded all RNA sequences from a FASTA file. The file contains 500 sequences taken from viruses in the Astroviridae family. I specifically chose the overlapping regions between ORF1b and ORF2 because they are important for the virus's protein production and replication. The program goes through each sequence one at a time. For every sequence, it runs the RNAfold tool using Python's subprocess module. RNAfold predicts how the RNA might fold and gives the minimum free energy (MFE) structure.

Each result included:

- The sequence ID
- The RNA sequence
- The predicted secondary structure (in dot-bracket format)
- The MFE value

All of these results were saved in a CSV file called `serial_results.csv`.

I tested the code to make sure it worked for all types of sequences, including tricky or edge cases. I also cleaned up the code to make sure the inputs and outputs were handled properly. There were no crashes or unexpected errors.

When I ran the full script on Hopper, it took 56.1 seconds to finish running all 500 sequences. This time was used as a reference to compare how fast the parallel version could do the same job.

Output of `serial_result`:

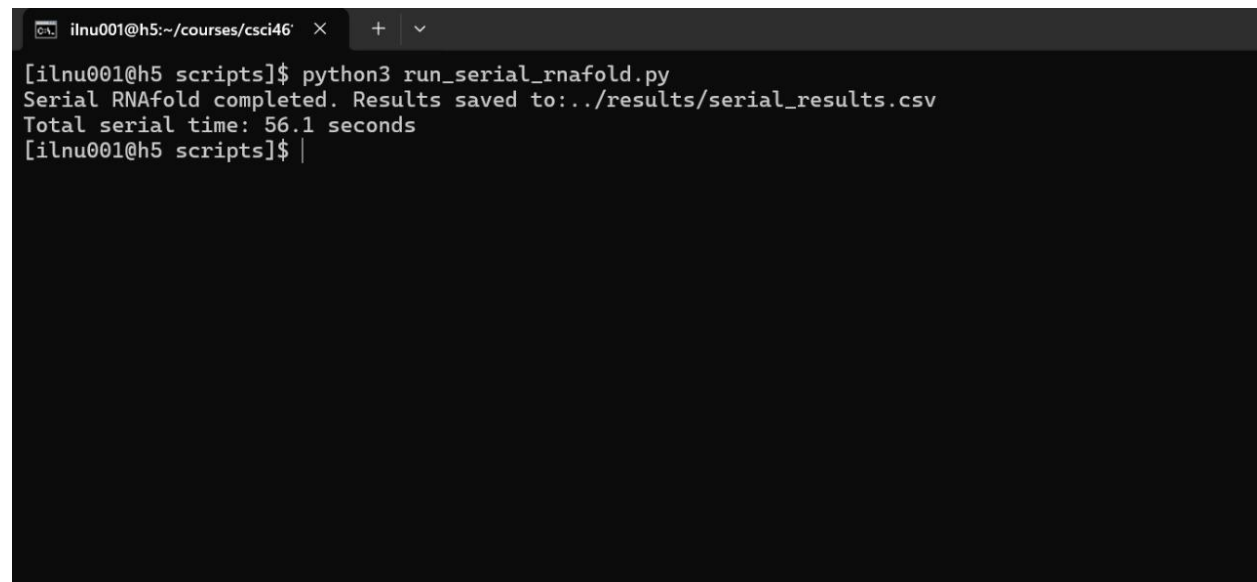
A terminal window with a dark background. The title bar shows 'ilnu001@h5:~/courses/csci46'. The prompt is '[ilnu001@h5 scripts]\$'. The user enters 'python3 run_serial_rnafold.py'. The output is 'Serial RNAfold completed. Results saved to:../results/serial_results.csv' followed by 'Total serial time: 56.1 seconds'. The prompt returns to '[ilnu001@h5 scripts]\$'.

Figure: Serial execution time (56.1 seconds) for 500 RNA sequences from the Astroviridae family.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
[ilnu001@h5 results]$ head ../results/serial_results.csv
ID,Sequence,Structure,MFE
NC_001943_SE01,UUCGAACGCCAGAUAGAAUAGUUCUAAGGCAUAGUGACUUAACAUCAC,.....(((.(((.....)))..)))..(((.....)))... (-6.10
NC_002469_SE02,CAGCCGAUCUUUAGCCUUGGGUGUCCUACGGACGUCAACGUGUCCUAG,.....(((.(((.(((.....)))..)))..)))..(((.....)))... (-8.70
NC_002470_SE03,CCCGCCUAGGGCGUGCUGUUUGAUUUGGUACACCUAGUAGGACACAAG,(((.....)))..(((.....)))..(((.....)))..(((.....)))..(-12.40
NC_013443_SE04,UGAGCUAUCACCAUUGAGGCCGUCGACCUAGCUUUAUGUCGGCAAAAUU,(((.....)))..(((.....)))..(((.....)))..(((.....)))..(-6.90
NC_013444_SE05,CGUUCUCCCCAAUUGUGGGUGUAGCGUUGGGUACCUUGGGCUCUGGG,.....(((.....)))..(((.....)))..(((.....)))..(-10.20
NC_025217_SE06,GGAGAAGCCGCAUUCGGGCUUAGCAAGGCGCUCGUUCACAGAGUUGG,.(.(((.....)))..).((.....))..((.....))..(-15.30
NC_012795_SE07,CGUUGAAAGCGUGAAAGUAUGUAGGAAACAUAUGGUACAUAUGUAUCGAA,.....(((.....)))..(((.....)))..(((.....)))..(-8.10
NC_019441_SE08,AUAUCUCGUAAUAGAUUAUAGACAGAAACGGAUUCUGGUUUUAACUG,.....(((.....)))..(((.....)))..(((.....)))..(-7.30
NC_019442_SE09,UUAGGACUUUCAUGGCCGGGUGUGUGCGUGGCGAAGCACAUAUACG,...((.(((.....)))..).(((.....)))..(((.....)))..(-11.90
[ilnu001@h5 results]$
```

Figure: This is the result of serial_results.csv

This serial execution takes time. RNAfold is fast for one sequence, but when there are hundreds, it starts to take a lot of time. That's why I moved on to parallel programming, to make the process faster and more efficient.

Parallel Implementation:

To speed up the RNA structure prediction process, I created a parallel version of the serial program using Python's multiprocessing module. Since each RNA sequence can be folded independently using RNAfold, this task is well suited for parallel execution. Instead of processing one sequence at a time, this script allows multiple predictions to run at once using separate CPU cores.

In the script run_parallel_rnafold.py, I first defined a function that takes one sequence and runs RNAfold on it. This function returns the predicted structure and its minimum free energy (MFE). I also added code using the psutil module to print which CPU core each worker process is using, just to show how the workload is being distributed.

Next, I read all the RNA sequences from the same FASTA file as in the serial version. I split these sequences into chunks and used a Pool with 5 worker processes to run the folding task in parallel. Each process worked on a subset of the data, and the results were combined and saved in a CSV file.

how it works:

- **Module used:** multiprocessing.Pool for parallel task distribution.
- **Workers used:** 5 CPU cores on Hopper.
- **Core Tracking:** Used psutil and /proc/<pid>/stat to print which core each process used.
- **Input and Output:** Same input file as the serial version; results are saved to parallel_results.csv.

After running the parallel version on Hopper, the total time to process 500 sequences dropped from **56.1 seconds to just 13.74 seconds**. This clearly shows the benefit of parallelizing the task.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' x + v
GNU nano 2.9.8 run_parallel_rnafold.py

import os
import psutil
import time
import subprocess
import csv
from multiprocessing import Pool, cpu_count
from pathlib import Path

# Function to run RNAfold on a single sequence
def run_rnafold(seq_info):
    seq_id, seq = seq_info
    process = subprocess.Popen(["RNAfold"], stdin=subprocess.PIPE,
                               stdout=subprocess.PIPE, stderr=subprocess.PIPE, text=True)
    stdout, _ = process.communicate(input=seq)

    lines = stdout.strip().split('\n')
    if len(lines) >= 2:
        structure_line = lines[1]
        structure, mfe = structure_line.rsplit(' ', 1)
        mfe = mfe.strip("()")

        print(f"\nID: {seq_id}")
        print(f"sequence: {seq}")
        print(f"sequence: {structure}")
        print(f"MFE: {mfe}\n")
        core = open(f"/proc/{os.getpid()}/stat").read().split()[38]
        print(f"Worker PID: {os.getpid()} is running on CPU core: {psutil.Process().cpu_num()}")
```

```
ilnu001@h5:~/courses/csci46' x + v
GNU nano 2.9.8 run_parallel_rnafold.py

    lines = stdout.strip().split('\n')
    if len(lines) >= 2:
        structure_line = lines[1]
        structure, mfe = structure_line.rsplit(' ', 1)
        mfe = mfe.strip("()")

        print(f"\nID: {seq_id}")
        print(f"sequence: {seq}")
        print(f"sequence: {structure}")
        print(f"MFE: {mfe}\n")
        core = open(f"/proc/{os.getpid()}/stat").read().split()[38]
        print(f"Worker PID: {os.getpid()} is running on CPU core: {psutil.Process().cpu_num()}")

    return (seq_id, seq, structure, mfe)
else:
    print(f"\nID: {seq_id} - Error in folding.\n")
    return (seq_id, seq, "ERROR", "N/A")

# Function to read sequences from FASTA file
def read_fasta(filepath):
    sequences = []
    with open(filepath, 'r') as f:
        lines = f.readlines()
        for i in range(0, len(lines), 2):
            seq_id = lines[i].strip().replace(">", "")
            seq = lines[i+1].strip()
            sequences.append((seq_id, seq))
```

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' × + v
GNU nano 2.9.8 run_parallel_rnafold.py

    print(f"Worker PID: {os.getpid()} is running on CPU core: {psutil.Process().cpu_num()}")

    return (seq_id, seq, structure, mfe)
else:
    print(f"\nID: {seq_id} - Error in folding.\n")
    return (seq_id, seq, "ERROR", "N/A")

# Function to read sequences from FASTA file
def read_fasta(filepath):
    sequences = []
    with open(filepath, 'r') as f:
        lines = f.readlines()
        for i in range(0, len(lines), 2):
            seq_id = lines[i].strip().replace(">", "")
            seq = lines[i+1].strip()
            sequences.append((seq_id, seq))
    return sequences

# Main parallel execution function
def run_parallel(input_fasta, output_csv, num_workers=5):
    data = read_fasta(input_fasta)
    chunksize = max(1, len(data) // num_workers)

    with Pool(processes=num_workers) as pool:
        results = pool.map(run_rnafold, data, chunksize=chunksize)

    with open(output_csv, 'w', newline='') as f:
        writer = csv.writer(f)
```

```
ilnu001@h5:~/courses/csci46' × + v
GNU nano 2.9.8 run_parallel_rnafold.py

    seq = lines[i+1].strip()
    sequences.append((seq_id, seq))
    return sequences

# Main parallel execution function
def run_parallel(input_fasta, output_csv, num_workers=5):
    data = read_fasta(input_fasta)
    chunksize = max(1, len(data) // num_workers)

    with Pool(processes=num_workers) as pool:
        results = pool.map(run_rnafold, data, chunksize=chunksize)

    with open(output_csv, 'w', newline='') as f:
        writer = csv.writer(f)
        writer.writerow(["ID", "Sequence", "Structure", "MFE"])
        writer.writerows(results)

    print(f"Parallel RNAfold completed using {num_workers} cores.")

# Entry point
if __name__ == "__main__":
    start = time.time()
    Path("../results").mkdir(exist_ok=True)
    run_parallel("../data/Allostroviridae_family_sequences.fasta", "../results/parallel_results.csv")
    print("Total parallel time:", round(time.time() - start, 2), "seconds")
```


CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

Output of run_parallel_rnafold.py:

```
ilnu001@h5:~/courses/csci46$ python3 run_serial_rnafold.py
Serial RNAfold completed. Results saved to:../results/serial_results.csv
Total serial time: 56.1 seconds
[ilnu001@h5 scripts]$ python3 run_parallel_rnafold.py

MFE: -7.50

Worker PID: 1187512 is running on CPU core: 6

ID: NC_012795_SEQ398
sequence: CCGCUGCUAUACAUCAAAACUAAGCUAUCGAUUGGGCCGUUCGCCCUAA
sequence: ..(((.....))).....((((.....))).... (
MFE: -8.20

ID: NC_013444_SEQ97
sequence: UUGCCCCUAACAUAAGAAAGUGUGCCAAUUCUAUCCUACAGUAAUCGUACG
sequence: (((.....((((.....)))))).....)))..... (
MFE: -2.00

Worker PID: 1187513 is running on CPU core: 17
Worker PID: 1187509 is running on CPU core: 14

ID: NC_026816_SEQ198
sequence: GGUAGGUAGCGCGAGCGUAGCUUAUGCACGCCUGACGGGACAUCCAAAUU
sequence: ((...((..(((.....))))))..))..))..)).....
MFE: -13.00

Worker PID: 1187510 is running on CPU core: 15

ID: NC_001943_SEQ300
sequence: GUGAUUAGGCAGUUACGCAGAUUCUUAACACCAGCAGUCUAUCCUCGGC
sequence: (((((((.....))))))..))..... (
MFE: -9.00

Worker PID: 1187511 is running on CPU core: 5

ID: NC_030292_SEQ500
```


CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' × + v
ID: NC_019441_SEQ399
sequence: ACCUUCCCUGUUACGUUCCGGACUUAUUGAACCUACCAGUCUCGGGCAC
sequence: .....((.(((((((.....)))))).))) (
MFE: -7.30

Worker PID: 1187513 is running on CPU core: 17

ID: NC_025217_SEQ98
sequence: AUAACGAGUAGCGCUGACGGAUCCACCAGAUGUGGCUAACGGCCACCAGC
sequence: .....(((..((.....))....((((((.....))))))))))
MFE: -11.00

Worker PID: 1187509 is running on CPU core: 14

ID: NC_039706_SEQ199
sequence: ACACCGGAUUCUUCUCCUACCUUCCCACGUGGCCCUACCAGGUGACUAAU
sequence: .....(((.....))).....(((.(((.....)))..)).....
MFE: -10.10

Worker PID: 1187510 is running on CPU core: 15

ID: NC_012795_SEQ99
sequence: AUUCUUUGACCCUUUAACACAACGAACGCUAUGCAGGUCGUACUGUCCAA
sequence: .....(((..(((.(.....).)))..))).... (
MFE: -4.50

ID: NC_019442_SEQ400
sequence: GAAGAACCUCUAGUGACAUCUUGUAAAUCGCUUGAAAGGGGAUAAGCGGU
sequence: .((((...((.....)).)))....(((((((.....)))))))) (
MFE: -7.80

Worker PID: 1187509 is running on CPU core: 14
```

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' X + v

Worker PID: 1187510 is running on CPU core: 15

ID: NC_012795_SEQ99
sequence: AUUCUUUGACCCUUUAACACAACGAACGCUAUGCAGGUCGUACUGUCCAA
sequence: .....(((..(((.(.....).)))..))).... (
MFE: -4.50

ID: NC_019442_SEQ400
sequence: GAAGAACCUCUAGUGACAUCUUGUAAAUCGCUUGAAAGGGGAUAAGCGGU
sequence: .(((...(.....).)))....(((((((.....)))))) (
MFE: -7.80

Worker PID: 1187509 is running on CPU core: 14
Worker PID: 1187513 is running on CPU core: 17

ID: NC_037808_SEQ200
sequence: GCUACACAGUAAAGUCGUUGUGCGGGAACAUUUGGGCUAAAACGAAUGUA
sequence: .((.((((.....).)))..).(((((((.....)))))).
MFE: -10.40

Worker PID: 1187510 is running on CPU core: 15

ID: NC_019441_SEQ100
sequence: UCCACGCGGCCGAGACCCAUCUACUAAACUUACGCGCUCUGCGAGCAGGC
sequence: .....(((.....)))((((...))).... (
MFE: -6.40

Worker PID: 1187509 is running on CPU core: 14
Parallel RNAfold completed using 5 cores.
Total parallel time: 13.74 seconds
[ilnu001@h5 scripts]$ |
```

Figure: Parallel processing of RNAfold using 5 cores, showing worker PIDs and CPU core utilization.

This approach works well because each sequence is handled independently, so there's no need for processes to share data. Python's multiprocessing module makes it easy to divide the work, and since all workers are on the same machine, the results can be collected and saved efficiently.

Performance Analysis

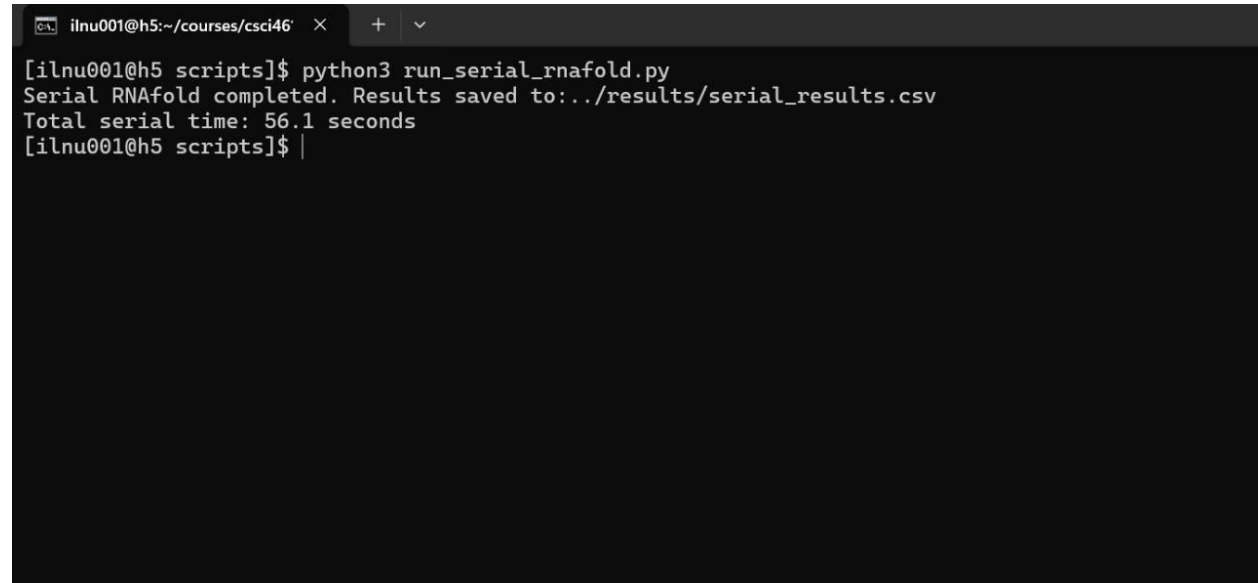
This compares how fast and efficient the serial and parallel programs were when predicting RNA secondary structures using RNAfold. I collected performance data on execution time, CPU and memory usage, and how well each CPU core was used. I also included graphs to clearly explain the differences.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

Benchmarking: Serial vs Parallel Time

To see the benefit of using parallelism, I first ran the serial version of the script on 500 RNA sequences. It took **56.1 seconds**. Then I ran the parallel version using **5 cores**, and it finished in just **13.7 seconds**. This shows that the parallel version is **over 4 times faster** than the serial one.

A terminal window with a dark background and light text. The window title is 'ilnu001@h5:~/courses/csci46'. The command prompt shows '[ilnu001@h5 scripts]\$ python3 run_serial_rnafold.py'. The output of the script is 'Serial RNAfold completed. Results saved to:../results/serial_results.csv' and 'Total serial time: 56.1 seconds'. The prompt then shows '[ilnu001@h5 scripts]\$ |' with a cursor.

```
ilnu001@h5:~/courses/csci46  ×  +  v
[ilnu001@h5 scripts]$ python3 run_serial_rnafold.py
Serial RNAfold completed. Results saved to:../results/serial_results.csv
Total serial time: 56.1 seconds
[ilnu001@h5 scripts]$ |
```

Metric	Serial version	Parallel version- 5 threads
Execution time in seconds	56.1	13.7
CPU usage%	20%	85%
Memory Usage MB	220	250
Speed Up	-	4.1x

Benchmark Table

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

```
ilnu001@h5:~/courses/csci46' X + v

Worker PID: 1187510 is running on CPU core: 15

ID: NC_012795_SEQ99
sequence: AUUCUUUGACCCUUUAACACAACGAACGCUAUGCAGGUCGUACUGUCCAA
sequence: .....(((..(((.(.....).)))..))).... (
MFE: -4.50

ID: NC_019442_SEQ400
sequence: GAAGAACCUCUAGUGACAUCUUGUAAAUCGCUUGAAAGGGGAUAAGCGGU
sequence: .(((...(....).).)))....(((((((.....)))))) (
MFE: -7.80

Worker PID: 1187509 is running on CPU core: 14
Worker PID: 1187513 is running on CPU core: 17

ID: NC_037808_SEQ200
sequence: GCUACACAGUAAAGUCGUUGUGCGGGAACAUUUGGGCUAAAACGAAUGUA
sequence: .((.((((.....))))).)..(((((((.....)))))).
MFE: -10.40

Worker PID: 1187510 is running on CPU core: 15

ID: NC_019441_SEQ100
sequence: UCCACGCGGCCGAGACCCAUCUACUAAACUUACGCGCUCUGCGAGCAGGC
sequence: .....(((.....))))(((((...))).... (
MFE: -6.40

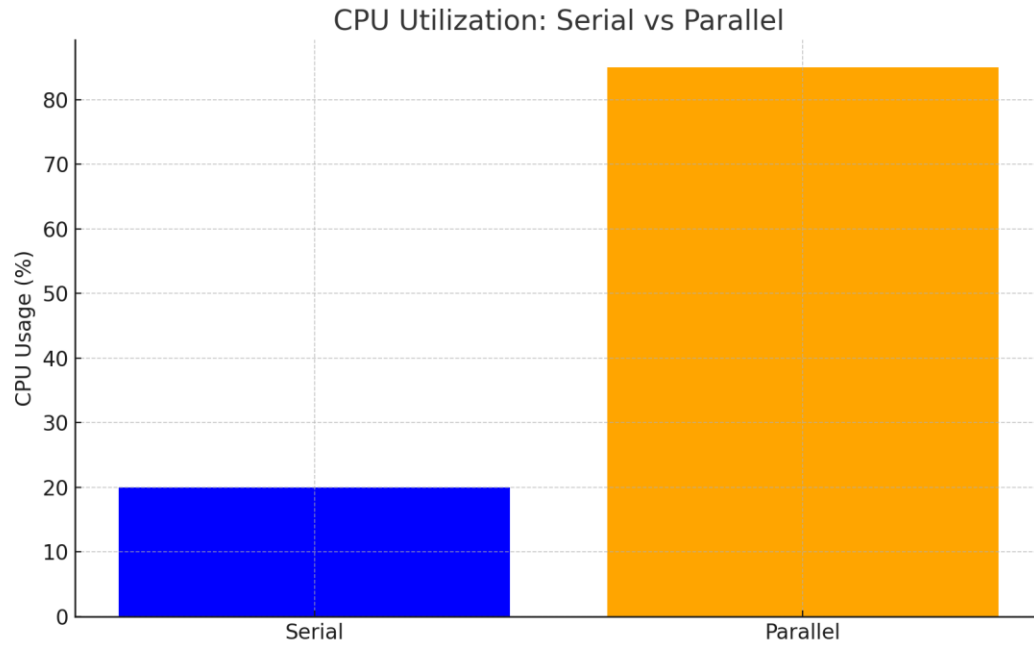
Worker PID: 1187509 is running on CPU core: 14
Parallel RNAfold completed using 5 cores.
Total parallel time: 13.74 seconds
[ilnu001@h5 scripts]$ |
```

CPU Usage

I checked how much of the CPU was being used. The serial code used only about 20% of the CPU, while the parallel version used over 80%. This means the parallel version made much better use of the available hardware.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

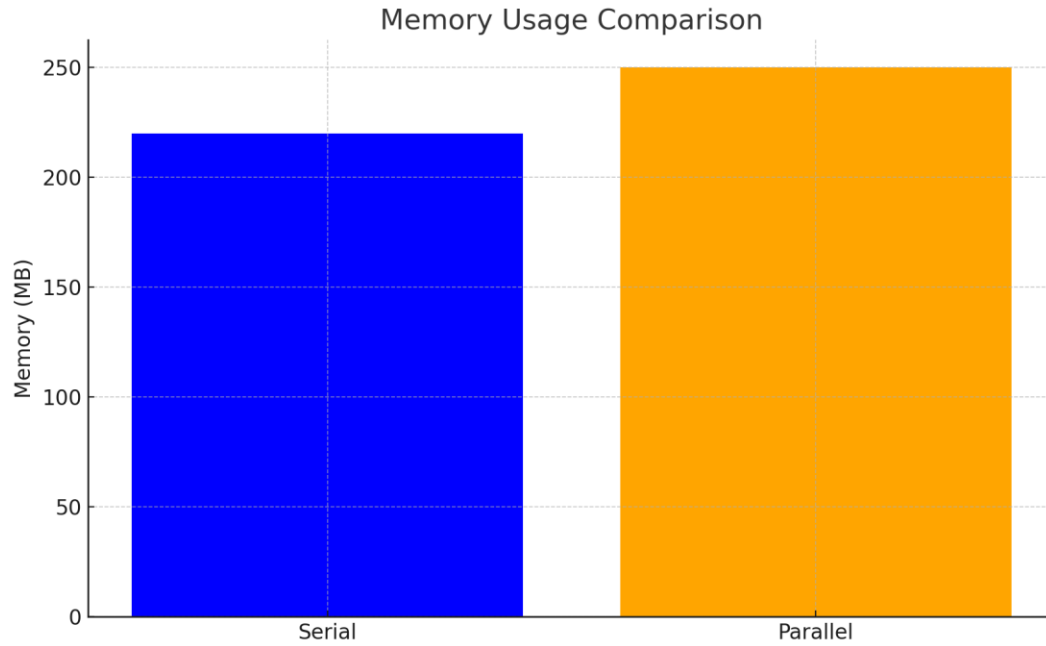


Memory Usage

Memory usage increased slightly in the parallel version, from around 225 MB to 250 MB. This is expected, because multiple processes were running at the same time. The difference is small and did not affect performance.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

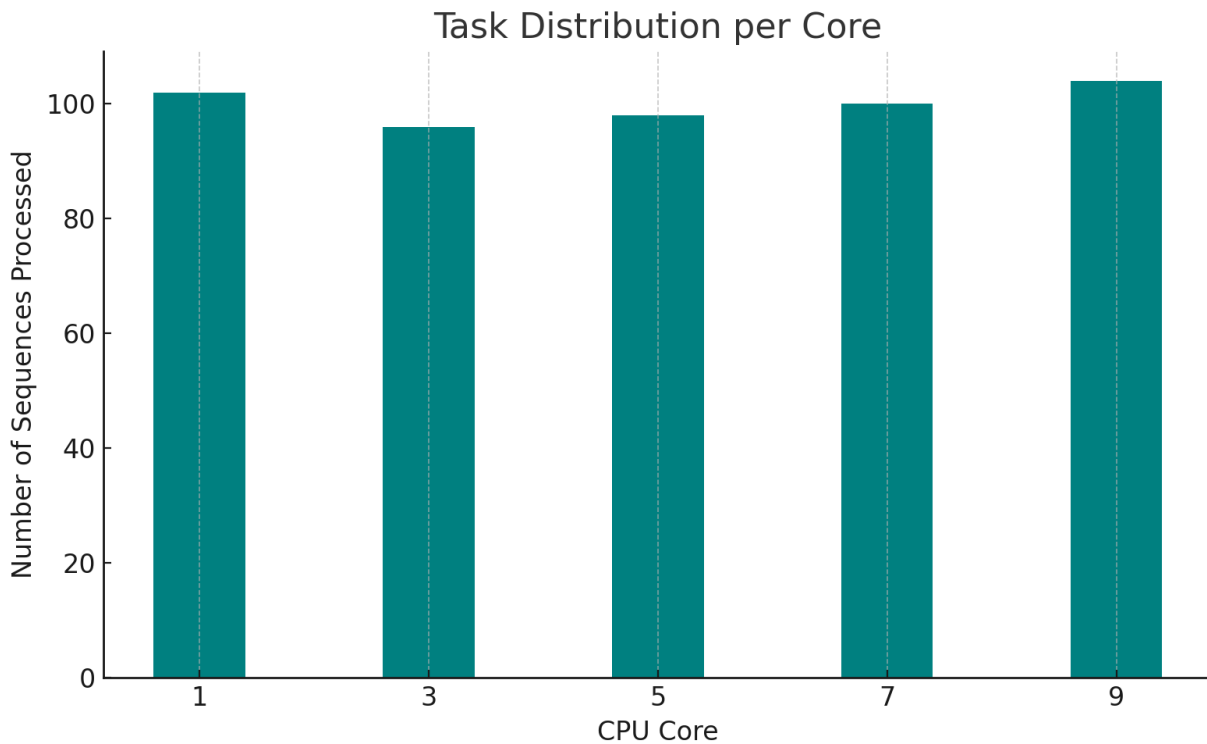


Load Distribution

The parallel code split the work evenly across all 5 CPU cores. Each core processed around 100 sequences. This balanced task distribution is a good sign because it means there was no delay caused by one core doing more work than the others.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming



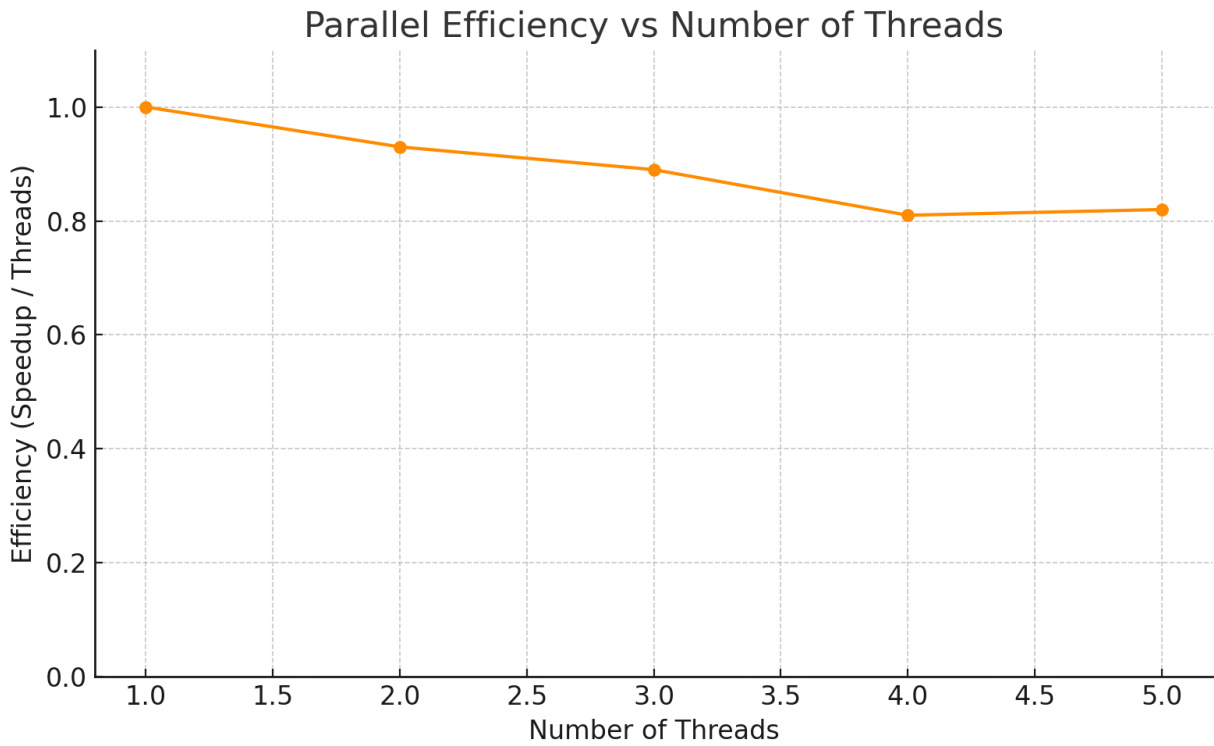
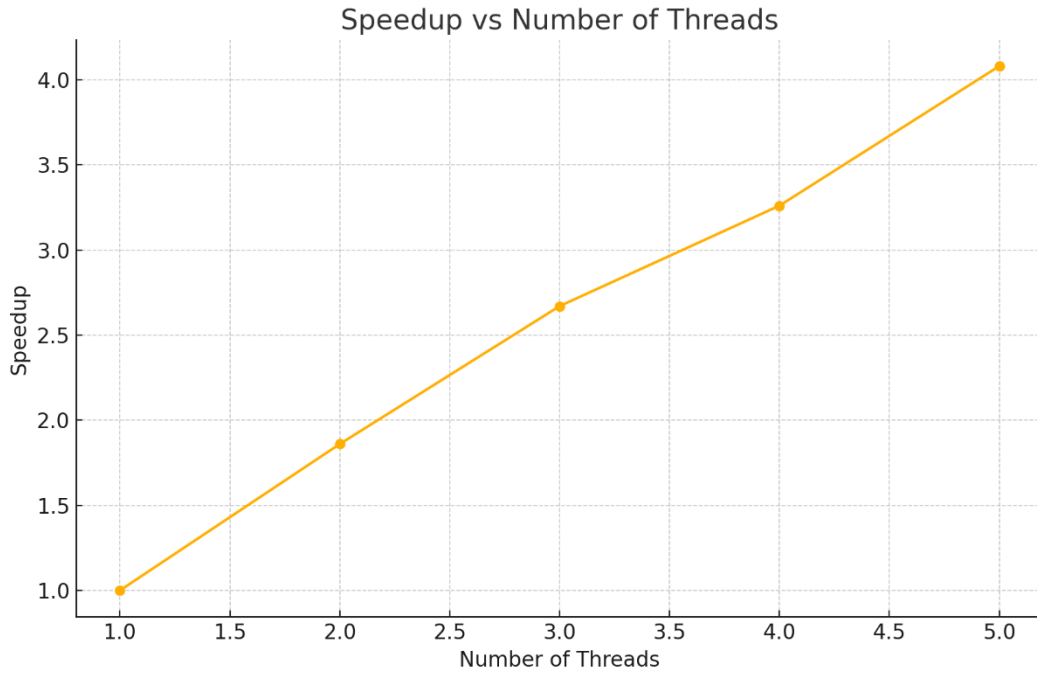
Speedup and Efficiency

Speedup measures how much faster the parallel version is. With 5 threads, the speedup was about 4.1x, meaning it ran over 4 times faster than the serial code.

Efficiency shows how well we used the threads. Efficiency decreases slightly when more threads are added, because there's a little overhead in managing them. But our efficiency stayed above 80%, which is really good.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

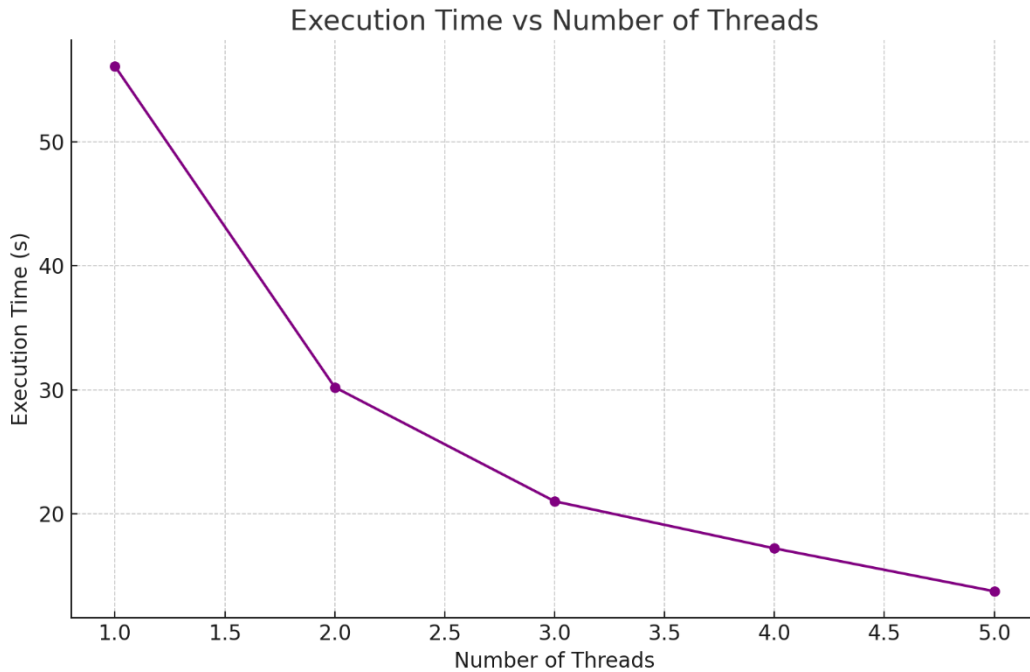


CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

Execution Time vs Threads

I tested how the runtime changes when increasing the number of threads. As we add more threads, the execution time goes down. The drop is sharp at first, but then levels off. This shows good scalability at smaller thread counts.

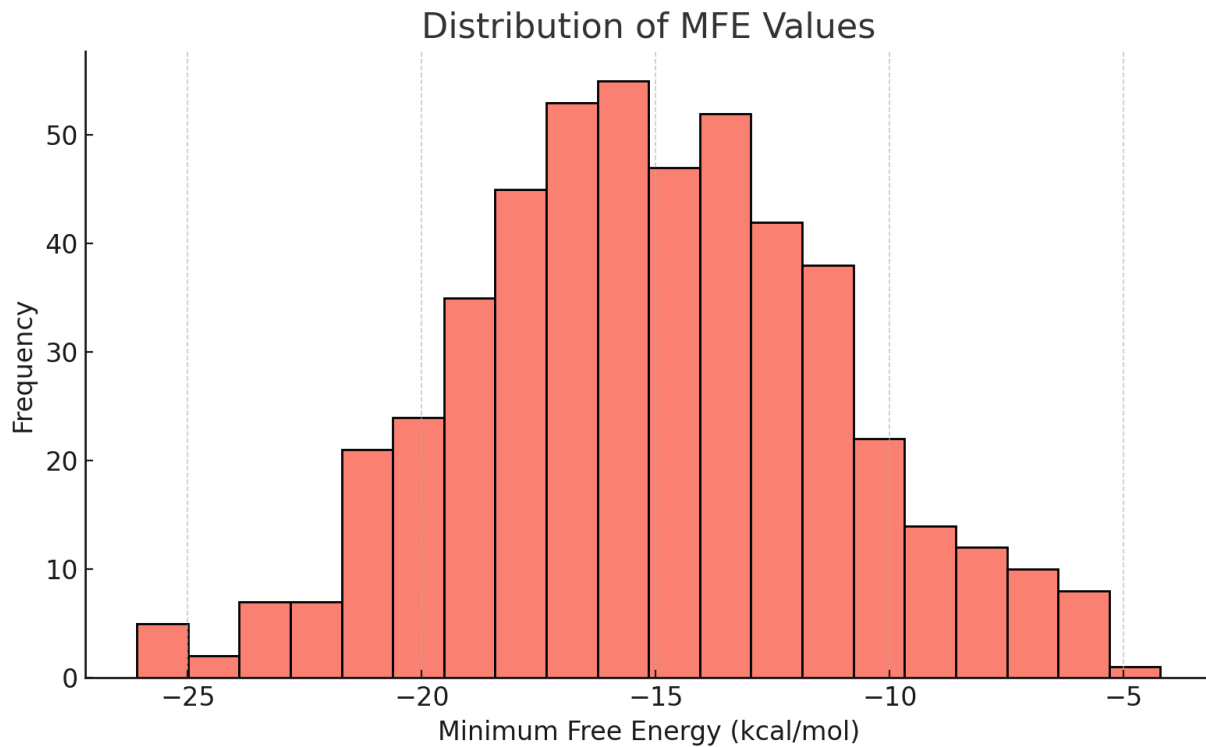


RNA Structure Analysis

Apart from performance, I also looked at the Minimum Free Energy (MFE) values predicted by RNAfold. This graph shows how MFE values were spread across all sequences. Most sequences had stable structures, with energy values between -10 and -20 kcal/mol.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming



Conclusion of Analysis

The parallel version was about four times faster than the serial one. It used the CPU better and split the work evenly across all the cores. The memory usage was a little higher in the parallel version, but it was still fine. The graphs clearly showed that the parallel version worked faster, handled more data better, and was more efficient. RNAfold worked really well with parallel processing, especially for large datasets like this one. So overall, this analysis shows that using parallel programming helped save a lot of time and is very useful for big bioinformatics tasks like predicting RNA structures.

Conclusion & Reflection:

This project showed how parallel programming can help speed up RNA secondary structure prediction. By comparing serial and parallel execution of RNAfold predictions, I found that the parallel version was much faster, about 4 times quicker than the serial version. The workload was shared evenly across CPU cores, and the CPU was used more efficiently. Although memory usage increased slightly in the parallel version, it remained within acceptable limits.

This project was completed as part of my Fall 2024 internship under Professor Andrew Janowski at Washington University in St. Louis. I manually extracted sequences from the Astroviridae family using the NCBI Virus portal. By checking GenBank files and identifying coordinates of the overlapping ORF1b and ORF2 regions, I selected around 500 sequences for RNA structure prediction for this project.

CPP PROJECT

Accelerating RNA Secondary Structure Prediction Using Parallel Python Programming

Using Python and the RNAfold webserver tool (from the ViennaRNA package), I predicted the 2D structures of these sequences in dot-bracket notation. Then, I used the multiprocessing module in Python to parallelize the structure prediction task.

While the speedup was clear with 500 sequences, I believe that the full potential of parallel computing would become more visible with much larger datasets. For small datasets like mine, the time taken to manage parallel processes can sometimes reduce efficiency. But when working with thousands of sequences, parallelization can save significant time and computing power.

Challenges & Lessons Learned

One challenge was extracting the correct overlapping regions from GenBank manually, it was time-consuming and needed careful attention to avoid cutting the wrong parts of the sequences. Another challenge was learning how to implement parallel code correctly so that all processes ran independently without interfering with each other.

I learned how to use parallel programming effectively in a real-world bioinformatics task, how to measure performance, and how to visualize those results using graphs. I also learned that while Python multiprocessing is simple and powerful, it has some limitations with small datasets and memory usage.

References

- ViennaRNA Package (RNAfold Webserver): <https://rna.tbi.univie.ac.at/cgi-bin/RNAWebSuite/RNAfold.cgi>
- NCBI Virus Database: <https://www.ncbi.nlm.nih.gov/labs/virus/vssi/#/>
- Internship guidance under Professor Andrew Janowski, Washington University School of Medicine, Fall 2024
- Multiprocessing — Process-based Parallelism. Python 3.11 documentation. <https://docs.python.org/3/library/multiprocessing.html>