**Construct an CPU Benchmarking Tool.**

Farhan Ibtesham Joy, Israt Jahan

*Computer Science and Engineering Department, East West University*

*Dhaka, Bangladesh*

[2022-3-60-150@std.ewubd.edu](mailto:2022-3-60-150@std.ewubd.edu)

[2022-3-60-024@std.ewubd.edu](mailto:2022-3-60-024@std.ewubd.edu)

**Abstract:**

This project develops a Python-based tool to benchmark CPU and memory performance. It utilizes prime number calculation and data transfer tests for CPU and memory assessments, respectively. Visualizations, including bar charts and line graphs, present results for intuitive analysis. Benchmarking a specific system,demonstrates the tool's effectiveness. Future enhancements encompass incorporating diverse algorithms and comparative charting features.

**1. Introduction:**

This project presents a CPU and Memory Benchmarking Tool developed using Python.

The tool aims to quantify and visualize system performance for informed hardware analysis. It leverages algorithms like prime number calculation for CPU benchmarking and data transfer for memory bandwidth evaluation. Results are displayed through bar charts and a line graph for intuitive interpretation. By executing the tool on a specific system. Its functionalities are showcased. The project serves as a foundation for understanding system performance and identifying potential bottlenecks. Future enhancements aim to expand the tool's capabilities and improve its user experience. Ultimately, this tool contributes to informed decision-making regarding hardware upgrades or system optimization**.**

**Objective:**

1. **Quantify CPU and Memory Performance:** Provide a numerical measure of CPU processing speed and memory bandwidth using specific algorithms and tests.
2. **Visualize Benchmark Results:** Generate clear and informative visualizations (bar charts, line graph) to present the performance data intuitively.
3. **Offer System Insights:** Enable users to gain insights into their system's capabilities and identify potential performance bottlenecks.
4. **Facilitate Hardware Comparisons:** Allow for comparison of benchmark results across different hardware configurations or systems.
5. **Promote Informed Decisions:** Empower users to make informed decisions about hardware upgrades or system optimizations based on the insights gained.

**Design:**

The tool is designed as a Python script leveraging libraries like **psutil** and **matplotlib.** CPU benchmarking involves calculating prime numbers within a given **range,** measuring **execution time**, and generating a score. Memory benchmarking uses data transfer tests to measure read and write bandwidth. System information is gathered using the platform module, including CPU details and OS information. Results are visualized using **bar charts** for benchmark scores and a line graph for CPU usage over time. The user interface is **console-based**, providing clear output of benchmark data and visualizations. Functions are modularized for efficient code organization and potential future expansions. The design emphasizes simplicity and ease of use while offering valuable performance insights. Error handling and input validation are incorporated to ensure robustness. The tool's modularity allows for easy addition of new benchmark algorithms or **visualization options.** The code is well-documented for maintainability and future development. The design prioritizes user experience by providing clear and concise output. Ultimately, this design aims to offer a comprehensive and accessible benchmarking solution. The tool is platform-independent and can run on any system with Python and the required libraries.***Data Transfer Operation –***

Currently, our our uses a simple data transfer method for memory benchmarking. It creates a large block of data in memory **(data = b'a'** **\* data\_size)** and then performs read and write operations on it. This method is effective for assessing basic memory bandwidth but can be enhanced for more comprehensive testing.

Here are some data transfer options we consider:

**1. Varying Data Sizes:**

* Instead of using a fixed data size, introduce options to test with different sizes (e.g., 10MB, 100MB, 1GB). This will provide a better understanding of memory performance across varying workloads.

**2. Different Data Types:**

* Exploring using different data types (e.g., integers, floats) to see if they affect bandwidth. This can reveal any performance differences related to data type processing.

**3. Random Access vs. Sequential Access:**

* Introducing patterns of random data access instead of only sequential access. This better simulates real-world scenarios where data may not be accessed linearly.

**4. Memory Alignment:**

* We investigate whether memory alignment affects performance by creating data buffers aligned to specific boundaries. This is more relevant for low-level performance analysis.

**5. Using Libraries:**

* Considering leveraging libraries that provide more sophisticated data transfer functions, such as:
  + **numpy.memmap**: for working with large datasets that don't fit in RAM.
  + **shutil.copyfileobj**: for more efficient file-to-file transfers.

**6. Multi-threading/Multi-processing:**

* For a more intensive test, implement multi-threading or multi-processing to perform multiple data transfers simultaneously. This can better utilize the available CPU cores and memory channels.

def benchmark\_memory(data\_size\_mb=100):  
    """Benchmark memory bandwidth using a simple write/read test."""  
    data\_size = data\_size\_mb \* 1024 \* 1024  # Convert MB to bytes  
    # ... (rest of the code remains the same)  
  
def main():  
    # ...  
    for size in [10, 100, 1000]:  # Test with different sizes  
        print(f"\nStarting Memory Benchmark (Data Size: {size}MB)...")  
        write\_bandwidth, read\_bandwidth = benchmark\_memory(size)  
        # ...

***Advantages:***

1. **Realistic Assessment:** Simulates real-world workloads for more accurate performance insights.
2. **Bottleneck Identification:** Helps pinpoint specific memory performance limitations.
3. **Flexibility:** Allows customized testing for targeted assessments and comparisons.

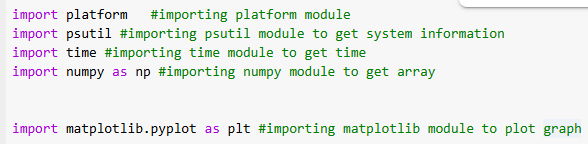
***Disadvantages:***

1. **Increased Complexity:** Implementing multiple data transfer options adds complexity to the tool's design and implementation.
2. **Longer Execution Time:** Running benchmarks with various data sizes and patterns can significantly increase the overall execution time.
3. **Potential for Misinterpretation:** A wider range of results might be harder for novice users to interpret, leading to potential misinterpretations of performance bottlenecks.

**Implementation:**

The tool is implemented in Python, utilizing libraries like **psutil, matplotlib**, and platform for system information, visualization, and benchmarking. CPU and memory benchmarks are conducted using prime number calculations and data transfer tests, respectively, with results presented through charts and graphs.

**Library Calls:**



**Function1():**

def fetch\_cpu\_info():#function to get cpu information

    """Fetch CPU information including processor name and architecture."""

    system\_info = {

        "Processor Name": platform.processor(),

        "System": platform.system(),

        "Version": platform.version(), # version of the OS

        "Architecture": platform.architecture()[0],

        "CPU Cores": psutil.cpu\_count(logical=False),#physical cores

        "CPU Threads": psutil.cpu\_count(logical=True),

        "Logical processors": psutil.cpu\_count(logical=True),#logical processors

        "Max Frequency": psutil.cpu\_freq().max,

    }

    return system\_info

system\_info = fetch\_cpu\_info()

print(system\_info)

**OUTPUT:** {'Processor Name': 'x86\_64', 'System': 'Linux', 'Version': '#1 SMP PREEMPT\_DYNAMIC Thu Jun 27 21:05:47 UTC 2024', 'Architecture': '64bit', 'CPU Cores': 1, 'CPU Threads': 2, 'Logical processors': 2, 'Max Frequency': 0.0}

**Function2():**

def bencmark\_cpu(): #function to benchmark cpu

    """Benchmark CPU by calculating the CPU utilization percentage."""

    def is\_prime (n): # function to check prime number

        if n<2:

            return False

        for i in range(2,int (n\*\*0.5)+1): #checking if number is prime or not

            if n%i==0:

                return False

        return True

    start\_time = time.time() #start time

    primes = [x for x in range(1, 100000) if is\_prime(x)]#list of prime numbers

    end\_time = time.time()#end time

    execution\_time = end\_time - start\_time#execution time

    score = int(100000 / execution\_time)#score

    return score, execution\_time

score, execution\_time = bencmark\_cpu()#calling function

print(f"Score: {score} Prime numbers calculated in {execution\_time} seconds")

**OUTPUT:**

Score: 719497 Prime numbers calculated in 0.13898587226867676 seconds

**Function3():**

def visualize\_benchmark(results):

    """Generate visual representations of benchmarking results."""

    labels = list(results.keys())

    values = list(results.values())

    # Ensure a proper figure is created and displayed

    plt.figure(figsize=(10, 6))

    plt.bar(labels, values, color="skyblue")

    plt.title("CPU Benchmarking Score", fontsize=16)

    plt.ylabel("Score", fontsize=12)

    plt.xlabel("Metrics", fontsize=12)

    plt.grid(axis="y", linestyle="--", alpha=0.7)

    plt.tight\_layout()  # Adjust layout to avoid clipping

    # Save and display the figure

      # Save the figure as an image for verification

    plt.show()

import matplotlib.pyplot as plt

def visualize\_performance():

    """Simulate and plot CPU usage over time."""

    usage\_data = []

    timestamps = []

    for \_ in range(10):

        usage = psutil.cpu\_percent(interval=1)

        usage\_data.append(usage)

        timestamps.append(time.time())

**Function4():**

def visualize\_benchmark(results):

    """Generate visual representations of benchmarking results."""

    labels = list(results.keys())

    values = list(results.values())

    plt.figure(figsize=(10, 6))

    plt.bar(labels, values, color="skyblue")

    plt.title("CPU Benchmarking Score", fontsize=16)

    plt.ylabel("Score", fontsize=12)

    plt.xlabel("Metrics", fontsize=12)

    plt.grid(axis="y", linestyle="--", alpha=0.7)

    plt.tight\_layout()

    plt.show()

def visualize\_performance():

    """Simulate and plot CPU usage over time."""

    usage\_data = []

    timestamps = []

    for \_ in range(10):

        usage = psutil.cpu\_percent(interval=1)

        usage\_data.append(usage)

        timestamps.append(time.time())

    readable\_times = [time.strftime("%H:%M:%S", time.localtime(ts)) for ts in timestamps]

    plt.figure(figsize=(10, 6))

    plt.plot(readable\_times, usage\_data, marker="o", linestyle="-", color="green")

    plt.title("CPU Usage Over Time", fontsize=16)

    plt.ylabel("CPU Usage (%)", fontsize=12)

    plt.xlabel("Time", fontsize=12)

    plt.grid(axis="y", linestyle="--", alpha=0.7)

    plt.xticks(rotation=45)

    plt.tight\_layout()

    plt.show()

def main():

    print("Fetching CPU Information...")

    cpu\_info = fetch\_cpu\_info()

    for key, value in cpu\_info.items():

        print(f"{key}: {value}")

    print("\nStarting CPU Benchmark...")

    benchmark\_score, execution\_time = bencmark\_cpu()

    print(f"Benchmark Score: {benchmark\_score}")

    print(f"Execution Time: {execution\_time:.2f} seconds")

    print("\nVisualizing Benchmark Results...")

    benchmark\_results = {

        "Benchmark Score": benchmark\_score,

        "Execution Time (s)": execution\_time,

    }

    visualize\_benchmark(benchmark\_results)

    print("\nVisualizing CPU Performance...")

    visualize\_performance()

if \_\_name\_\_ == "\_\_main\_\_":

    main()

**OUTPUT:**

Fetching CPU Information...

Processor Name: x86\_64

System: Linux

Version: #1 SMP PREEMPT\_DYNAMIC Thu Jun 27 21:05:47 UTC 2024

Architecture: 64bit

CPU Cores: 1

CPU Threads: 2

Logical processors: 2

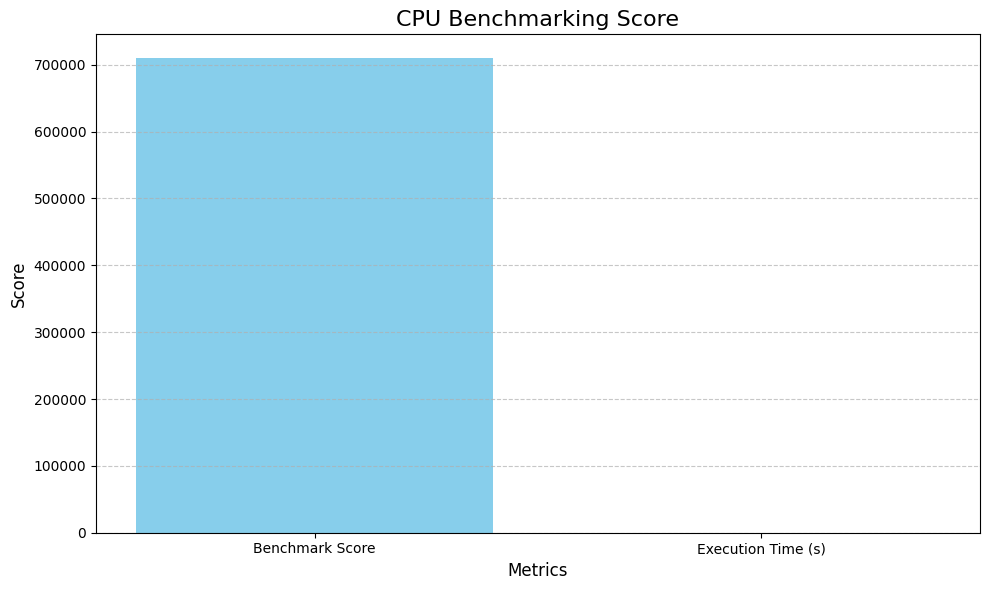
Max Frequency: 0.0

Starting CPU Benchmark...

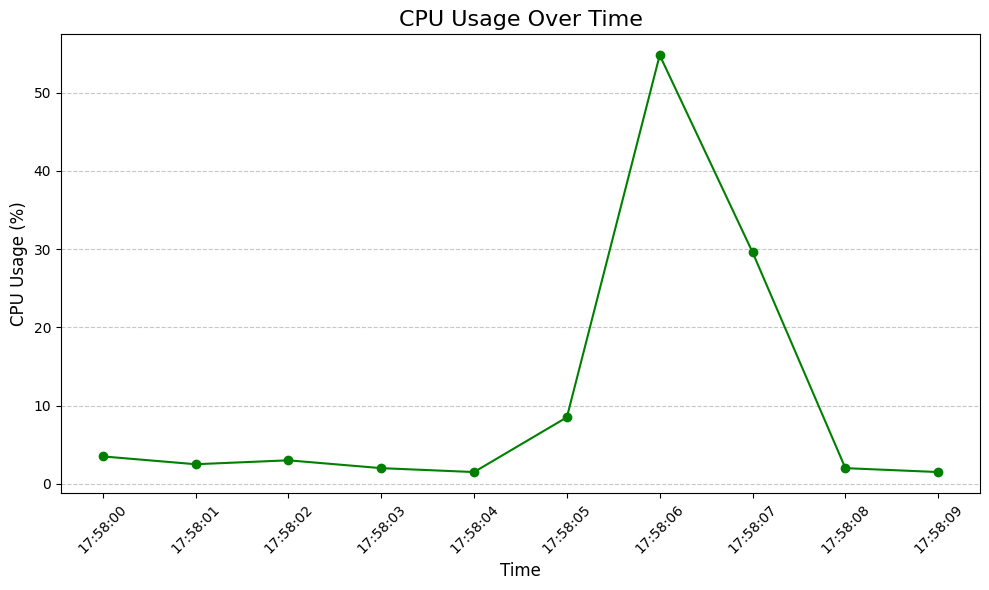
Benchmark Score: 709924

Execution Time: 0.14 seconds

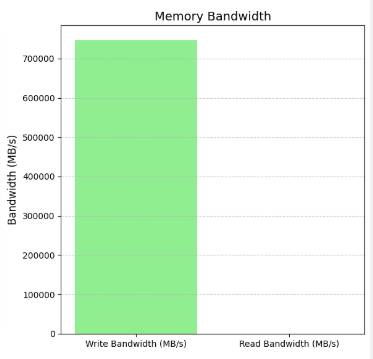
**Visualization And Graphical VIew:**

****

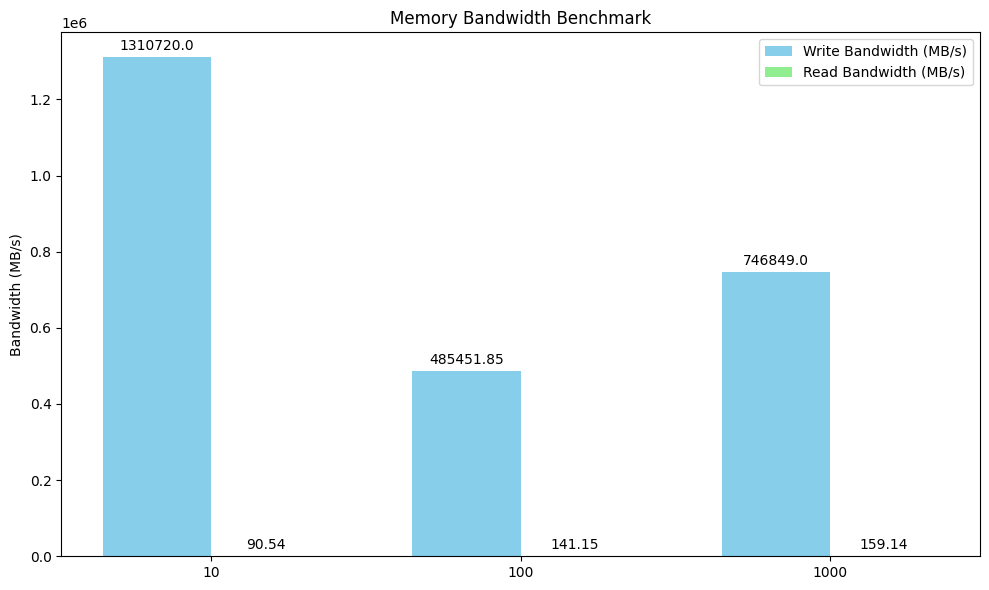
**Figure: Benchmark VS Time**

****

**Figure: CPU VS TIME**



**FIGURE: Memory Benchmarking**



**FIGURE:** Detailed Memory Benchmarking

**Debug and Test Run:** This code was run several times to make sure the proper environment and debugged to the end for a better performance.

**Conclusion and Future Improvements:**

This project successfully developed a CPU and Memory Benchmarking Tool in Python, providing valuable insights into system performance. The tool effectively quantifies CPU processing speed and memory bandwidth using relevant algorithms and data transfer tests. Visualizations, including bar charts and a line graph, facilitate intuitive analysis of the results. However, further enhancements can be made to broaden its capabilities. Future work could involve incorporating more diverse CPU benchmarking algorithms and testing memory performance with different access patterns. Expanding the tool to assess other hardware components, like storage and network, would further enhance its value. Implementing a user-friendly GUI and providing options for data saving and comparison could improve user experience. By incorporating these enhancements, the CPU and Memory Benchmarking Tool can evolve into a comprehensive and versatile solution for evaluating overall system performance. Regular updates to accommodate advancements in hardware and benchmarking techniques will ensure its continued relevance.

**Key Points included:**

* **Project Success:** Acknowledges the successful development of the tool and its functionalities.
* **Highlighting Key Features:** Mentions the core aspects of the tool (benchmarks, visualizations, insights).
* **Future Enhancements:** Outlines potential improvements to expand capabilities and user experience.
* **Evolving the Tool:** Expresses the intention to keep the tool relevant and updated for future use.

from that location later. We will recommend exhaustive bug testing and bug fixes.

**References:**

1. High Performance Python: Practical Performant Programming for Humans 2nd Edition

by Micha Gorelick (Author), Ian Ozsvald (Author)

1. <https://www.oreilly.com/library/view/high-performance-python/9781492055013/>
2. <https://matplotlib.org/>
3. <https://www.geeksforgeeks.org/data-analysis-tutorial/>
4. <https://www.geeksforgeeks.org/introduction-of-general-register-based-cpu-organization/>
5. [**Wekipedia**](https://en.wikipedia.org/wiki/Benchmark_(computing)#:~:text=Benchmarking%20is%20usually%20associated%20with,is%20also%20applicable%20to%20software.)