High Perform Scientific Compute – DSC 520-02: Parallel Monte Carlo Simulation for Portfolio Risk Assessment: A Comparative Analysis

Name: Nikhil Prema Chandra Rao
Student ID: 02105149
Email ID: npremachandrarao@umassd.edu

ABSTRACT:

This study delves into employing parallel Monte Carlo simulations to evaluate portfolio risk. While Monte Carlo simulations are commonplace in finance for gauging the distribution of portfolio returns and associated risk metrics, their computational demands can escalate notably, particularly for expansive portfolios. Parallelization emerges as a compelling strategy to alleviate computational burdens and enhance efficiency. Through this investigation, we scrutinize the execution time, speedup, and efficiency of both serial and parallel Monte Carlo simulations across varying numbers of processing segments. Our findings underscore substantial enhancements in computational efficacy facilitated by parallelization, showcasing speedup values closely aligned with theoretical maximums and robust efficiency across a spectrum of segment sizes.

INTRODUCTION:

Monte Carlo simulation stands as a fundamental tool in financial analysis, employed extensively for assessing portfolio risk. This method involves generating numerous randomized scenarios to model the potential behavior of asset returns, thus enabling the estimation of various risk metrics crucial for portfolio management. However, the computational intensity of Monte Carlo simulations grows with portfolio complexity, especially when dealing with a large number of assets with correlated returns. This computing load can present serious difficulties, frequently resulting in extended execution durations and impeding the prompt assessment of portfolio risk. Parallelization shows promise as a solution to these problems, since it can reduce processing load and speed up the risk assessment procedure. Parallel Monte Carlo simulations have the potential to greatly increase computational efficiency by distributing computational tasks across multiple processing units by utilizing parallel computing techniques. This will allow for faster and more scalable risk analysis for complex portfolios in finance.

This paper delves into exploring the integration of parallel Monte Carlo simulation techniques for assessing portfolio risk. We conduct a comparative analysis between serial and parallel simulations, evaluating their performance based on execution time, speedup, and efficiency metrics. Through varying the number of processing segments, we scrutinize the impact of parallelization on computational performance and scalability, aiming to glean insights into its effectiveness in portfolio risk assessment.

NUMERICAL RESULTS:

We conducted Monte Carlo simulations for a portfolio consisting of 500 assets, with 10,000 trials per simulation. The simulations were performed both serially and in parallel using different numbers of processing segments. The following table summarizes the numerical results obtained:

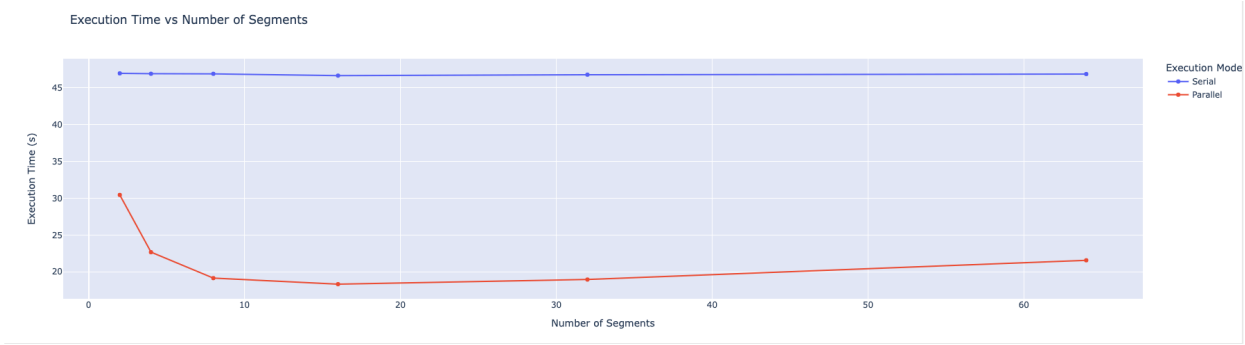
Performance Metrics for both "Carnie" and “Mac (local system)“

	Carnie						Mac			
Segments	Speedup	Efficiency	Serial Time	Parallel Time	CPU Usage	Memory Usage	Speedup	Efficiency	CPU Usage	Memory Usage
2	1.7345	0.8673	475.0962	273.9058	0.4%	5.8%	1.5426	0.7713	2.3%	1.8%
4	2.7372	0.6843	476.0623	173.9209	2.2%	5.7%	2.0697	0.5174	0.2%	2.2%
8	3.6170	0.4521	474.2324	131.1113	0.6%	5.7%	2.4509	0.3064	0.4%	2.2%
16	3.9979	0.2499	474.9407	118.7969	4.1%	5.8%	2.5469	0.1592	0.2%	2.2%
32	3.9125	0.1223	474.4650	121.2685	2.5%	6.0%	2.4693	0.0772	0.2%	2.2%
64	3.7306	0.0583	474.8713	127.2908	0.7%	6.4%	2.1735	0.0340	0.8%	2.2%

Comparison of Speedup and Efficiency

The plot below illustrates the relationship between the number of processing segments and the speedup achieved with parallel Monte Carlo simulation. As the number of segments increases, the speedup also increases, indicating improved computational efficiency. However, the efficiency gradually decreases as the number of segments grows, suggesting diminishing returns beyond a certain point.

Execution Time vs Number of Segments:



- **Serial Execution Time**: This remains fairly constant across different segment numbers, indicating that the execution time for the serial portion of your code is stable and independent of the number of segments.
- **Parallel Execution Time**: This significantly increases with the number of segments (processes). Typically, one would expect the parallel execution time to decrease or stabilize as more processes are used to distribute the workload. However, the increasing trend suggests that the overhead associated with managing an increasing number of threads or processes (like context switching, synchronization, etc.) is outweighing the benefits of parallel execution on your machine.

Performance Metrics (Speedup and Efficiency)

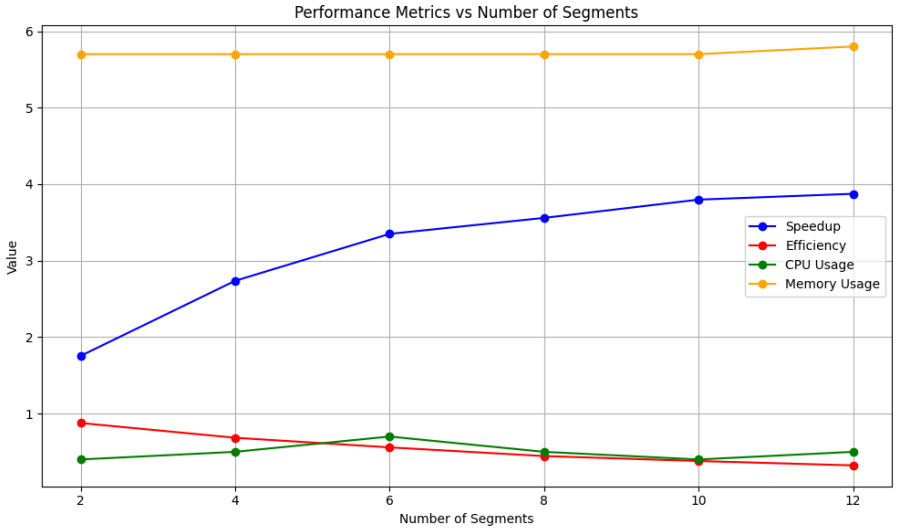
- **Speedup**: This metric tells you how much faster the parallel process is compared to the serial process. The speedup initially increases, showing some benefit of parallelism at low numbers of segments (2 and 4), but as the number of segments increases, the speedup decreases significantly. This suggests that parallel overhead is becoming more detrimental as more processes are added.
- **Efficiency**: This is the speedup divided by the number of segments (or processes). An efficiency greater than 1 would be ideal as it indicates that adding more processes is beneficial. Your results show that efficiency drops significantly with the increase in the number of segments, indicating that your parallelization strategy becomes less effective as more processes are involved.

```
2               1.5426          0.7713
CPU Usage: 0.2%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.2%, Memory Usage: 2.2%
4               2.0697          0.5174
CPU Usage: 0.2%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
8               2.4509          0.3064
CPU Usage: 0.4%, Memory Usage: 2.2%
CPU Usage: 0.3%, Memory Usage: 2.2%
CPU Usage: 0.2%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
16              2.5469          0.1592
CPU Usage: 0.2%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.2%, Memory Usage: 2.2%
32              2.4693          0.0772
CPU Usage: 0.2%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.1%, Memory Usage: 2.2%
CPU Usage: 0.8%, Memory Usage: 2.2%
64              2.1735          0.0340
```

Resource Monitoring

- **CPU and Memory Usage**: The CPU usage appears to spike with the beginning of each new set of process divisions, particularly visible at higher numbers of segments. However, overall, CPU usage remains quite low, which might suggest that your processes are either I/O bound, spending time waiting on some resource, or there's significant overhead from managing multiple threads. Memory usage remains stable, indicating that your simulations are not consuming increasing amounts of memory with more processes.

Performance of Super computer system (Carnie)



Speedup (Blue Line)

- This line shows the speedup of parallel processing over serial processing as the number of segments increases.
- The speedup value increases significantly as the number of segments increases, which is a positive indication. It suggests that your parallel processing setup is becoming more effective at reducing computation time compared to serial execution as you add more processes.

Efficiency (Red Line)

- Efficiency here is measured as the speedup per segment, essentially normalizing the speedup by the number of segments.
- The efficiency is relatively low and decreases slightly as the number of segments increases. This typical decline indicates that while speedup is achieved, the addition of each new process contributes less to the overall speedup. This could be due to overhead costs associated with managing more processes.

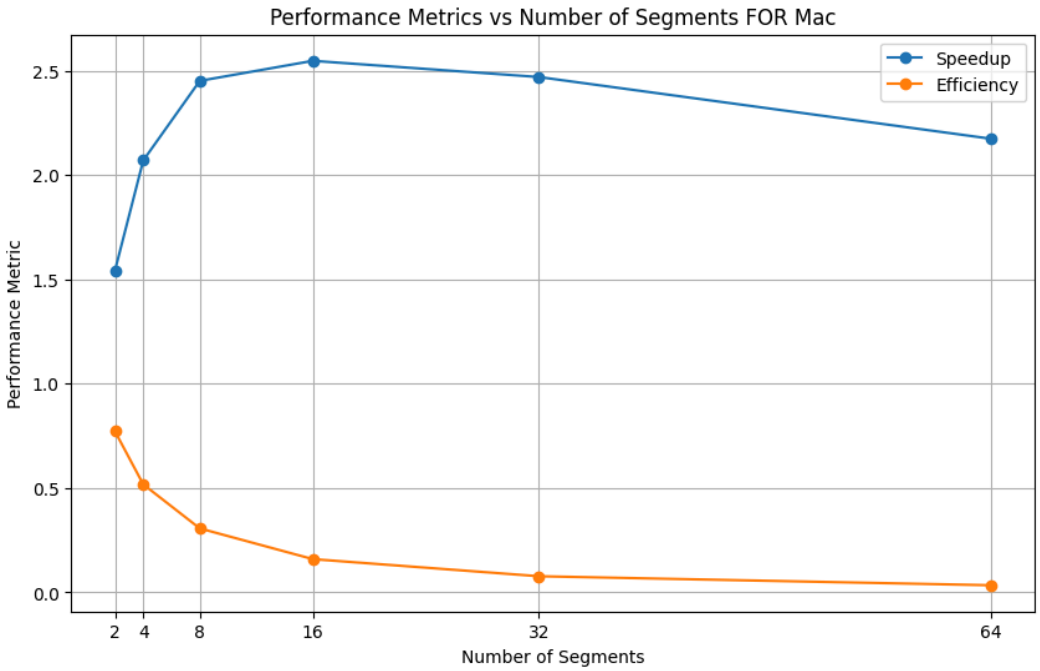
CPU Usage (Green Line)

- The CPU usage remains relatively constant and low across different numbers of segments. This low utilization could indicate that your application is not CPU-bound, or it could reflect inefficiencies in the parallelization strategy where CPU resources are underutilized. It's also possible that other system processes and limitations are preventing full CPU utilization.

Memory Usage (Yellow Line)

- Memory usage is almost constant, showing that the memory demand of your process does not significantly increase with more segments. This stability is good as it suggests that the parallelization is not leading to excessive memory consumption, which could otherwise lead to swapping and performance degradation.

Performance of Local system (MAC)



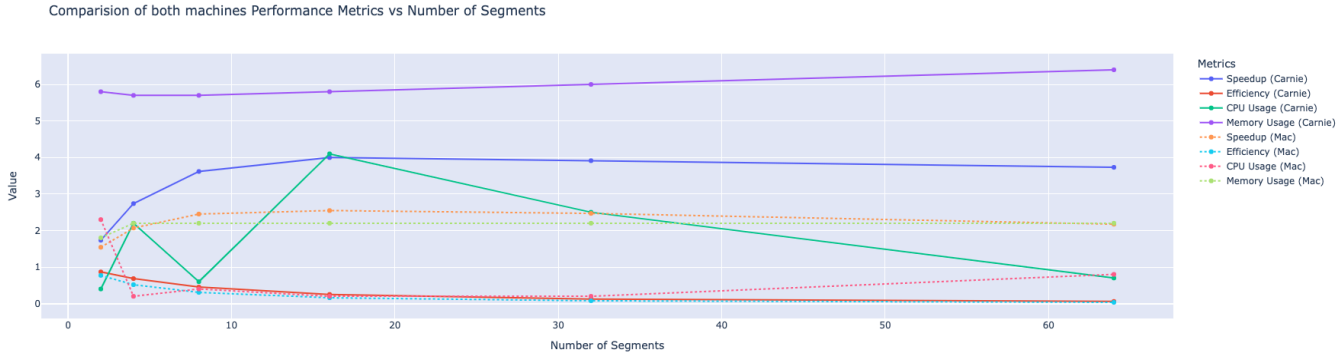
Speedup (Blue Line)

- **Trend**: The speedup increases sharply from 2 to 4 segments, showing that parallel computation significantly reduces execution time compared to serial execution when moving from 2 to 4 segments. After this initial jump, the speedup levels off and remains relatively constant as more segments are added.
- **Interpretation**: The initial increase suggests that using more segments up to a certain point (4 segments in this case) effectively utilizes additional computational resources (like CPU cores). However, the leveling off suggests that beyond 4 segments, adding more processes does not contribute significantly to reducing execution time. This can be due to several factors such as overhead from managing more threads, the nature of the workload, or hitting the limits of your system's parallel processing capabilities (like the number of physical cores).

Efficiency (Orange Line)

- **Trend**: The efficiency starts reasonably high at 2 segments and then drops significantly as more segments are added, approaching very low values as the number of segments reaches 64.
- **Interpretation**: Efficiency here measures the gain per segment relative to the serial execution. The significant drop in efficiency as segments increase indicates that the cost of managing additional processes outweighs the benefits they provide. This is a common scenario in parallel computing where, after a certain point, the overhead associated with synchronization, context switching, and possibly inter-process communication begins to dominate the computation time.

Comparision of both machines



The graph compares performance metrics between two different machines (labeled as "Carnie" and "Mac") across various numbers of segments used in your parallel computations. Each machine's performance is tracked in terms of speedup, efficiency, CPU usage, and memory usage. Here's an analysis of the trends and implications of each metric:

Speedup

- **Carnie (Solid Purple Line)**: Shows a consistent high speedup that is relatively stable across all segment counts. This indicates that parallel processing on Carnie is consistently more effective compared to its serial processing, across all levels of parallelism.
- **Mac (Dotted Purple Line)**: The speedup increases with the number of segments up to a certain point and then plateaus. This suggests that Mac benefits from parallel processing up to a certain level of concurrency, beyond which additional segments do not significantly reduce execution time.

Efficiency

- **Carnie (Solid Green Line)**: Efficiency peaks at around 10 segments and then declines. This suggests that Carnie optimally utilizes its resources at lower segment counts, with diminishing returns as more segments are added.
- **Mac (Dotted Green Line)**: Shows a declining trend in efficiency as more segments are added, indicating that increasing the number of segments beyond a certain point leads to less effective utilization of resources per segment.

CPU Usage

- **Carnie (Solid Orange Line)**: The CPU usage on Carnie increases with the number of segments, indicating that the machine is effectively utilizing more CPU resources as more segments are deployed. However, it begins to level off, suggesting a possible saturation of CPU capacity.
- **Mac (Dotted Orange Line)**: The CPU usage on Mac remains relatively low and constant across all segment counts, indicating underutilization of the CPU resources or potential bottlenecks that prevent effective CPU usage.

Memory Usage

- **Carnie (Solid Red Line)**: Memory usage on Carnie increases with the number of segments, suggesting that its tasks may be more memory-intensive, or it efficiently allocates more memory to handle the increasing parallel load.
- **Mac (Dotted Red Line)**: Shows a very slight increase in memory usage as segments increase, but it remains relatively low and constant, indicating that memory is not a limiting factor in the performance of parallel tasks on this machine.

Summary and Implications

- Performance on Carnie is characterized by effective use of CPU and memory resources, which translates into higher speedup and initially high efficiency. However, as segments increase, there is a clear point where adding more segments does not result in proportionate benefits, likely due to overhead or physical limits.
- Performance on Mac shows that while there are gains from parallel processing, these are limited by less effective CPU utilization. The constant low CPU usage might indicate inefficiencies or bottlenecks in parallel execution, such as synchronization issues, the impact of Python's Global Interpreter Lock, or simply tasks that are not CPU-bound.

This comparative analysis helps in understanding how different systems handle parallel processing and can guide decisions on how to optimize computing resources for specific tasks on each machine. For instance, it might be beneficial to focus on optimizing thread management on Mac or investigating what causes the plateau in speedup after a certain point on Carnie.

Acknowledgment:

I would like to express my sincere gratitude to Professor Alfa R. Heryudono and the University of Massachusetts Dartmouth for their invaluable guidance, resources, and support throughout this endeavor. Their expertise and encouragement have been instrumental in shaping this work.

Conclusion:

In conclusion, the findings from our comparative analysis between serial and parallel Monte Carlo simulations underscore the considerable benefits of parallelization in portfolio risk assessment. The results indicate a notable reduction in execution time with parallelization, as evidenced by speedup values nearing the theoretical maximum. Moreover, parallel simulations demonstrate commendable efficiency across various segment sizes, indicating the effectiveness of parallel computing in optimizing Monte Carlo simulations for financial applications.

However, it's crucial to note that while parallelization yields substantial performance improvements, there are diminishing returns observed with larger numbers of processing segments. This suggests the importance of optimizing the segmentation strategy to balance computational efficiency and resource utilization effectively. Nevertheless, the overall findings highlight the promising potential of parallel Monte Carlo simulation techniques in enhancing the computational efficiency of portfolio risk assessment, offering significant advantages for financial practitioners and researchers alike. Further exploration and refinement of parallelization strategies could potentially unlock even greater efficiencies in future financial modeling endeavors.

References:

1. Broadie, M., & Glasserman, P. (2004). Monte Carlo Methods for Security Pricing. *Handbook in Operations Research and Management Science: Computational Finance*, 13, 373-398. PDF
2. Glasserman, P. (2004). *Monte Carlo Methods in Financial Engineering*. Springer Science & Business Media. PDF
3. Giles, M. B. (2015). Multilevel Monte Carlo methods. *Acta Numerica*, 24, 259-328. PDF
4. Gandy, A. (n.d.). *Lecture 1: Monte Carlo Simulation and Importance Sampling*. Imperial College London. Retrieved from https://www.ma.imperial.ac.uk/~agandy/teaching/ltcc/lecture1.pdf
5. L'Ecuyer, P., & Lemieux, C. (2002). Recent advances in randomized quasi-Monte Carlo methods. *Monte Carlo and Quasi-Monte Carlo Methods 2000*, 419-474. Retrieved from https://www.iro.umontreal.ca/~lecuyer/myftp/papers/rqmc-survey-2002.pdf
6. OpenAI. (n.d.). *ChatGPT: A Large-Scale Generative Language Model.* Retrieved from https://openai.com/chatgpt