|  |  |
| --- | --- |
|  |  |
| Montecarlo Minimization |  |
| **Assignment 1** |  |
|  | 07/08/2023CSC2002S |
|  | Nkhumeleni BestNKHBES001 |

Table of Contents

[Montecarlo Minimization 1](#_Toc142430218)

[07/08/2023 1](#_Toc142430219)

[CSC2002S 1](#_Toc142430220)

[Nkhumeleni Best 1](#_Toc142430221)

[NKHBES001 1](#_Toc142430222)

[Abstract 3](#_Toc142430223)

[Introduction 4](#_Toc142430224)

[Methodology 5](#_Toc142430225)

[1. Monte Carlo Approach Overview: 5](#_Toc142430226)

[2. Linearized Monte Carlo Approach: 5](#_Toc142430227)

[3. Parallelized Monte Carlo Approach 5](#_Toc142430228)

[4. Experiment Design: 6](#_Toc142430229)

[5. Data Collection: 6](#_Toc142430230)

[Results 7](#_Toc142430231)

[Benchmarking: 7](#_Toc142430232)

[Performance Evaluation: 8](#_Toc142430233)

[Comparison and Analysis: 8](#_Toc142430234)

[Discussion: 8](#_Toc142430235)

[Conclusion: 8](#_Toc142430236)

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | AbstractIn the rapidly evolving landscape of computational techniques, parallelism has emerged as a powerful approach to enhance the efficiency and speed of various algorithms. This report presents the parallelization of the "Montecarlo Minimization" program.  Through a series of experiments and performance evaluations, we demonstrate the superiority of the "Montecarlo Minimization" program's parallelized approach over its linearized counterpart. The results exhibit not only remarkable speedup in execution time but also emphasize the potential for substantial gains in efficiency when dealing with computationally intensive optimization tasks. | |  |

# Introduction

The Monte Carlo approach is a computational technique that relies on random sampling to estimate numerical results. While it's been a topic of debate, it can prove remarkably effective when dealing with systems or operations where our knowledge is limited.

In this report, we will employ the Monte Carlo approach to determine the lowest value within a defined region (terrain), characterized by a known mathematical function. Our goal is to accomplish this in two distinct ways, subsequently comparing the correctness and speed of each approach.

The first approach involves using a linearized version of the Monte Carlo method. In this scenario, we create a list of searches and systematically proceed through them, executing each search one after the other.

The second approach adopts a parallel strategy. Here, we assign each search to a separate thread, enabling multiple searches to be executed concurrently. This parallel execution holds the potential to expedite the process.

# Methodology

1. Monte Carlo Approach Overview:   
   The Monte Carlo approach essentially begins by selecting a random starting point. It then examines the values of nearby points in relation to this initial point. This process is particularly useful in our context, as we aim to identify the smallest value within the defined terrain.
2. Linearized Monte Carlo Approach:In the linearized version of the Monte Carlo approach, we start by crafting a list of searches. Each search is given a randomly selected row and column. The purpose of this process is to locate a valley, which signifies the nearest smaller value. As we detect these values, we store them in a designated variable. This variable essentially acts as a shared storage space to hold the smallest value we've encountered so far.

As we proceed, the approach involves inspecting each search entry in the array sequentially. If a given search uncovers a value smaller than what we currently have in the shared variable, we update the variable to hold this new, smaller value. Conversely, if the search doesn't yield a smaller value, we simply move on to the next search entry and repeat the comparison process.

1. Parallelized Monte Carlo Approach**:**

In my parallel implementation, I start by calculating the total number of searches required based on the given information about rows, columns, and search density. If the number of searches is greater than the sequential cutoff, I halve the number of searches and allocate two threads to handle this divided workload. Each thread checks if its assigned search count is still greater than the cutoff.

This iterative division process continues until the number of searches for each thread becomes lower than the sequential cutoff. When the number of searches falls below the sequential cutoff, I create individual instances of the search class, with a random row and column to start at. The find valley method is then called, which finds the smallest value from our start point. If any search instance uncovers a lower value than what's currently stored in a shared variable, that new lower value replaces the previous one.

The result that each thread finds is then compared to each other, and the lower value is then returned as the global minimum.

1. Experiment Design:

We will be comparing the performance in terms of time of the serial algorithm and the parallel algorithm, to do this we have created a standard set of inputs which will be given to both the parallel algorithm and the serial algorithm, to ensure fairness. The range of x and y values will be fixed at -1000 to 1000. Each of these algorithms will be searching for valleys in a terrain that is defined by the Rosebrock function, this is to ensure accuracy, as the function only has one global minimum.

The dataset we will be using is:

* + 10 rows;10 columns and 0.1 search density
  + 50 rows;50 columns and 0.2 search density
  + 100 rows;100 columns and 0.3 search density
  + 200 rows; 200 columns and 0.4 search density
  + 500 rows; 500 columns and 0.5 search density
  + 1000 rows; 1000 columns and 0.6 search density
  + 2000 rows; 2000 columns and 0.7 search density
  + 5000 rows; 5000 columns and 0.8 search density
  + 7500 rows; 7500 columns and 0.9 search density
  + 10000 rows; 10000 columns and 1 search density

1. Data Collection:

I have developed a Python script to facilitate data collection by evaluating our standard inputs using both the parallel and linear java files. The process involves supplying a standard input to one of the algorithms and running it five times. During each run, the execution time of the algorithm is recorded. These time measurements are then gathered, and the median value is calculated to yield the final result. This approach ensures a comprehensive and statistically reliable assessment of algorithm performance.

# Results

### Benchmarking:

We need a point of reference, to compare the performance of our parallel algorithm to. From running the linear algorithm 5 times each for a given standard input and taking the median, whilst the searches are being conducted on a terrain that is defined by the Rosebrook function.

The data collected from the linear algorithm:

**This benchmark was ran on a 4 core intel i510300H**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of rows | Number of columns | Search density | Time taken |
| 10 | 10 | 0,1 | 1 |
| 50 | 50 | 0,2 | 1 |
| 100 | 100 | 0,3 | 1 |
| 200 | 200 | 0,4 | 3 |
| 500 | 500 | 0,5 | 12 |
| 1000 | 1000 | 0,6 | 57 |
| 2000 | 2000 | 0,7 | 335 |
| 5000 | 5000 | 0,8 | 2538 |
| 7500 | 7500 | 0,9 | 6513 |
| 10000 | 10000 | 1 | 13857 |

**This benchmark was run on the UCT nightmare servers**

|  |  |  |  |
| --- | --- | --- | --- |
| Number of rows | Number of columns | Search density | Time taken |
| 10 | 10 | 0,1 | 4 |
| 50 | 50 | 0,2 | 5 |
| 100 | 100 | 0,3 | 6 |
| 200 | 200 | 0,4 | 11 |
| 500 | 500 | 0,5 | 34 |
| 1000 | 1000 | 0,6 | 119 |
| 2000 | 2000 | 0,7 | 624 |
| 5000 | 5000 | 0,8 | 5172 |
| 7500 | 7500 | 0,9 | 13588 |
| 10000 | 10000 | 1 | 32749 |

### Performance Evaluation:

* + Present the results obtained from running experiments for both approaches.
  + Use graphs, tables, or other visual aids to illustrate the performance metrics, such as execution time and efficiency.

### Comparison and Analysis:

* + Compare the execution times and efficiency of the linearized and parallelized approaches.
  + Discuss the advantages and drawbacks of each approach based on the obtained results.
  + Analyze how the number of searches or other parameters influences the performance.

# Discussion:

* + Interpret the findings and implications of the performance comparison.
  + Reflect on the trade-offs between the two approaches in terms of computational resources, complexity, and practicality.
  + Consider the applicability of parallelization to similar optimization problems.

# Conclusion:

* + Summarize the key findings of the comparison between the linearized and parallelized Monte Carlo approaches.
  + State which approach is more efficient for the given problem and discuss potential factors influencing the decision.