UNIVERSITÀ
DEGLI STUDI
FIRENZE

Scuola di Ingegneria

Corso di Laurea Magistrale in
Intelligenza Artificiale

Evaluating the performance of Gradient Descent and Nesterov Accelerated Gradient on Logistic Regression Problems

Parallel Programming for Machine Learning
Project work

Candidate
Loric TONGO GUIMTSA

Professor
Marco BERTINI

Academic year: 2023/2024

1- Introduction

This report presents the comparative performance analysis between the sequential and parallel programs to resolve the Regression Logistic problem using the **Gradient Descent method** and the **Nesterov Accelerate Gradient**.

2- Context

Logistic regression models are trained to solve a nonlinear optimization problem. This problem can be solved using any of the most popular solvers for nonlinear unconstrained optimization. In particular, Gradient Descent method and Nesterov's method are both viable options. The Nesterov Accelerated Gradient (NAG) algorithm is an optimization technique that improves upon gradient descent by using a momentum term. It computes a "lookahead" update by combining the current gradient with a fraction of the previous update, which helps converge faster, especially in scenarios with high curvature or many local minima.

3- Methodology

a. Description of the problem

Logistic regression is a statistical and machine learning method used to model the probability of a certain event occurring. It is primarily used in binary classification problems, where the objective is to predict one of two possible classes.
The project aims to evaluate the performance of Gradient Descent and Nesterov Accelerated Gradient on L2-regularized logistic regression problems minimizing the objective function below:

$$\min_{w \in R^n} \frac{1}{2}\|w\|^2 + \sum_{i=1}^{N} \log\left(1 + \exp\left(-y_i(w^T x_i + b)\right)\right)$$

Where x_i and y_i are respectively the *input* and the *output* of the dataset represented as follow:

$$D = \left\{(x_i, y_i) \mid x_i \in R^n, y_i \in \{-1, 1\}\right\}$$

b. Description of the optimizers

The Nesterov and Gradient Descent methods use a combination of current and previous gradients to accelerate convergence. The algorithm operates by iterating the following steps:

- Calculating the gradients of the objective function with respect to the parameters.
- Updating the parameters using a combination of current and previous gradients.
- Adjusting the gradient step size **"alpha"** (by Armijo-type line search) and the acceleration parameter **"beta Nesterov"** (for Nesterov method).

The Nesterov Accelerate Gradient method will use a Nesterov's rule represented as follow:

$$\beta_{n+1} = (1 + \sqrt{4\beta_n^2 + 1})/2$$

$$y_n = x_n - \alpha_n \nabla f(x_n)$$

$$x_{n+1} = y_n + \beta_n(y_n - y_{n-1})$$

Where $\boldsymbol{\alpha_n}$ is the step size compute by the Armijo-type line search and $\boldsymbol{\beta_n}$ is the Nesterovs' rule.

c. Parallelization

To parallelize the optimization, we split the dataset into fragments, with each fragment processed by a separate process. Python's ***"Multiprocessing"*** and ***"concurrent.futures"*** libraries were used to manage the processes in parallel.

4. Implementation

a. Sequential program

The sequential program is implemented, and we used in the first time six (06) small sets of real-world instances (6 datasets) to carry out experiments. And then, to experiment big quantities of data, we generate random data in process.

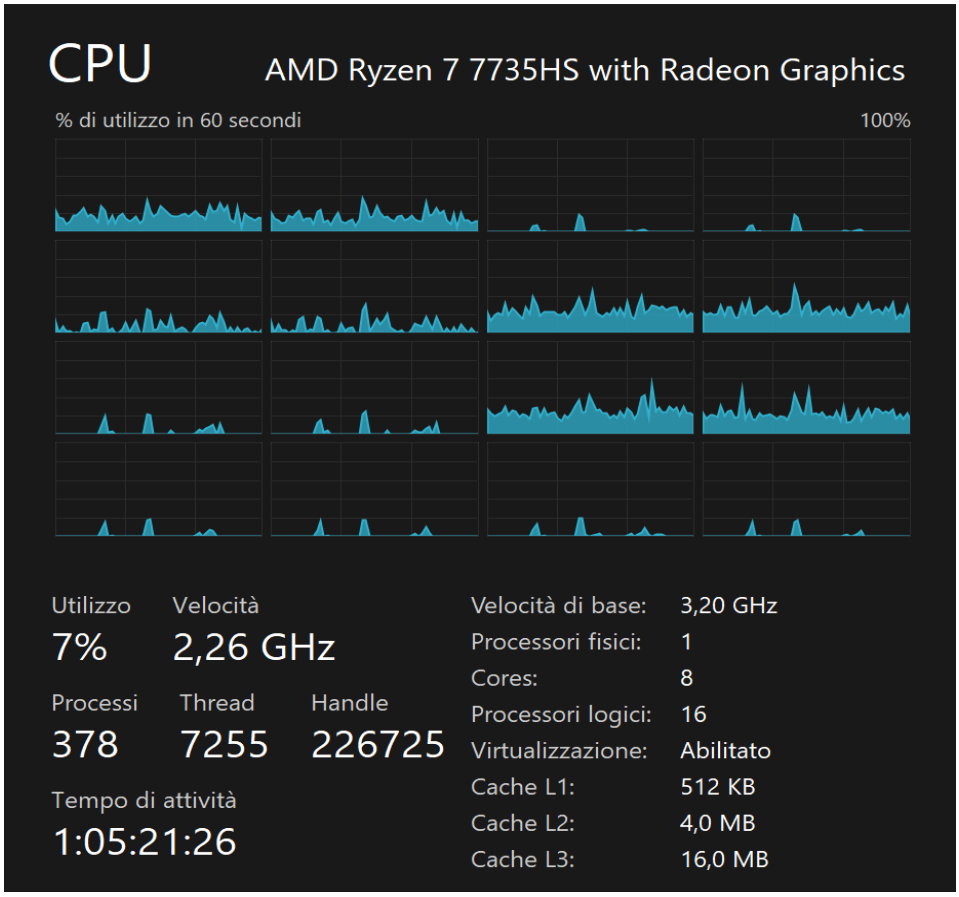


Figure 1: CPU utilization

As evidenced by Figure 1, which depicts CPU utilization during the execution of the sequential program, it's apparent that the sequential program does not utilize all CPU cores. The figure clearly illustrates that only about 7% of the CPU power is utilized during the execution of the sequential program.

b. Parallel Programs

We used **"Multiprocessing"** and **"Concurrent.futures"** libraries to implement the parallel program for this project. We also used six (06) small real-world datasets to carry out experiments and generated data in process to experiment with large quantities of data.

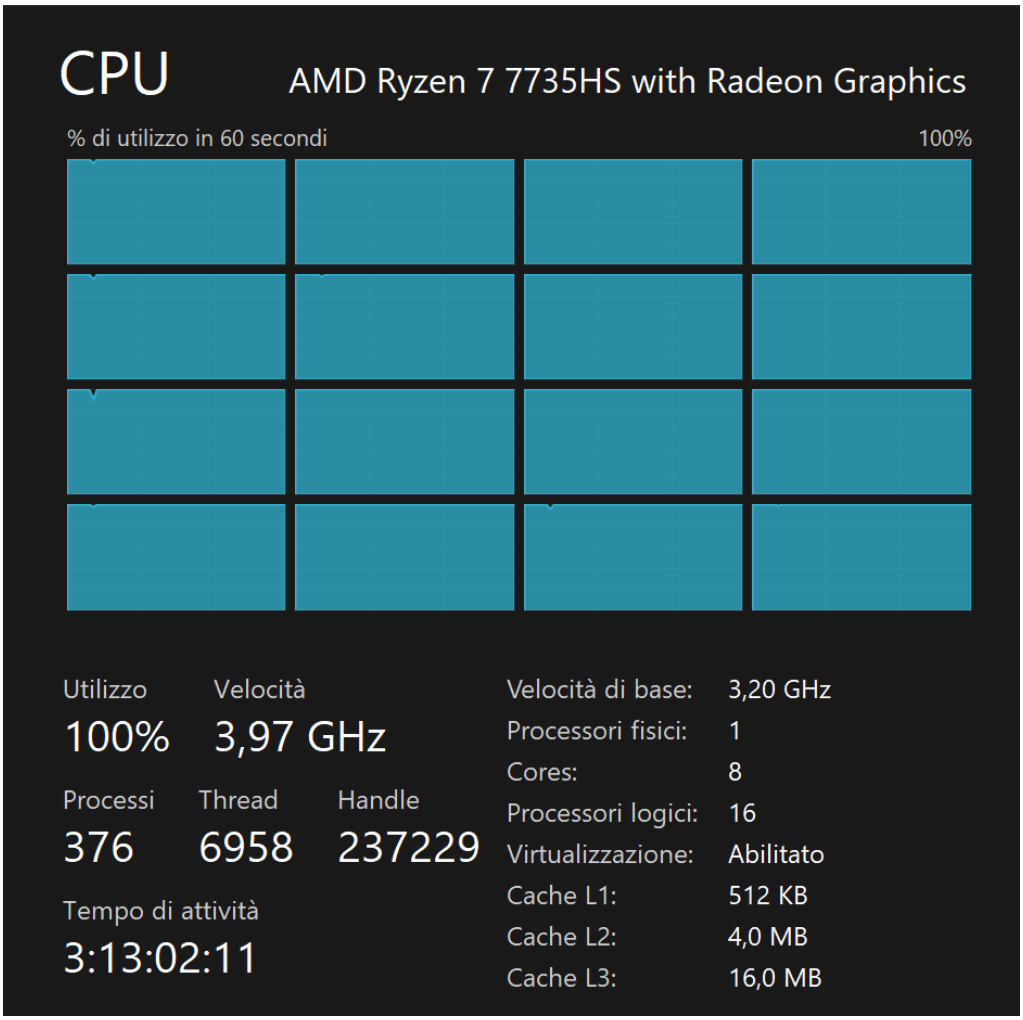


Figure 2: CPU utilization

As illustrated in Figure 2, which depicts CPU utilization during the execution of the parallel program, it's evident that the parallel program utilizes all CPU cores. The figure clearly shows that the parallel program efficiently utilizes the entire processing power of the CPU.

5. Results

a. Sequential Program

The execution times for the sequential implementations of the Gradient Descent method and the Nesterov Accelerate Gradient method with the real-world datasets and the data generated are as follows:

i. Small real-world datasets

	Social network	Indian liver patient	Cancer data	Diabete	Mushroom	Taxi fare
Samples	400	500	568	768	10.000	50.000
Features	3	9	30	8	8	7
Gradient Descent	0.0456	0.0626	1.0439	0.3056	8.4261	27.7293
Nesterov Accelerate Gradient	0.175	0.1773	0.3068	0.0665	2.9593	2.7616

ii. Data Generated in Process

	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5
Samples	100.000	200.000	300.000	400.000	500.000
Features	10	13	15	19	8
Gradient Descent	23.5306	90.4669	201.4517	909.946	1034.6381
Nesterov Accelerate Gradient	5.9476	74.2551	143.2239	179.4508	170.6421

b. Parallel Program

i. Small real-world datasets

- **Multiprocessing:**
  - **Gradient Descent:**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Social network	1.48	1.66	2.04	2.27	2.69	3.1	3.37	2.37
Indian liver patient	1.46	1.78	1.99	2.4	2.82	2.84	3.44	2.39
Cancer data	1.49	1.92	2.15	2.34	2.67	2.86	3.21	2.37
Diabete	1.57	1.75	2.18	2.38	2.52	2.8	3.14	2.33
Mushroom	5.42	5.71	5.62	2.97	3.33	3.4	3.86	4.19
Taxi fare	8.41	9.53	9.15	8.78	9.35	10.16	10.94	9.47

❖ **Nesterov Accelerate Gradient**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Social network	**1.59**	**1.87**	**2.24**	**2.3**	**2.8**	**3.05**	**3.58**	**2.34**
Indian liver patient	**1.6**	**1.88**	**2.19**	**2.4**	**2.69**	**2.92**	**3.57**	**2.32**
Cancer data	**1.62**	**1.86**	**2.24**	**2.47**	**2.66**	**2.91**	**3.35**	**2.3**
Diabete	**1.62**	**2.08**	**2.28**	**2.8**	**2.71**	**2.95**	**3.32**	**2.53**
Mushroom	**3.24**	**3.56**	**3.8**	**2.8**	**3**	**3.18**	**3.58**	**3**
Taxi fare	**5.02**	**5.17**	**5.35**	**5.58**	**6.04**	**6.22**	**7.27**	**5.8**

➢ **Concurrent.futures:**

❖ **Gradient Descent**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Social network	**1.4285**	**1.8458**	**2.0181**	**2.4088**	**2.6502**	**2.9423**	**3.3007**	**2.37**
Indian liver patient	**1.4979**	**1.994**	**2.1506**	**2.397**	**2.7687**	**2.8449**	**3.2398**	**2.41**
Cancer data	**1.479**	**1.826**	**2.0119**	**2.3196**	**2.7714**	**2.8753**	**3.333**	**2.51**
Diabete	**1.5775**	**1.7577**	**2.0402**	**2.302**	**2.6237**	**2.9309**	**3.2814**	**2.5**
Mushroom	**5.4715**	**5.646**	**5.4445**	**3.0769**	**3.4757**	**3.4839**	**4.001**	**4.22**
Taxi fare	**8.5509**	**9.3186**	**9.0119**	**9.0761**	**9.625**	**10.3036**	**11.391**	**9.46**

❖ **Nesterov Accelerate Gradient**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Social network	**1.5793**	**2.2167**	**2.0841**	**2.4361**	**2.6541**	**3.0911**	**3.4485**	**2.5**
Indian liver patient	**1.5969**	**2.0946**	**2.227**	**2.5859**	**2.7885**	**2.8592**	**3.305**	**2.49**
Cancer data	**1.6189**	**2.0086**	**2.2833**	**2.3982**	**2.7424**	**2.9449**	**3.2682**	**2.46**
Diabete	**1.6353**	**1.9028**	**2.2497**	**2.5387**	**2.9626**	**2.9998**	**3.3723**	**2.66**
Mushroom	**3.2148**	**3.4198**	**3.7304**	**2.727**	**3.0383**	**3.2703**	**3.6261**	**3.28**
Taxi fare	**4.849**	**4.9856**	**5.3893**	**5.6169**	**5.9394**	**6.2709**	**7.0492**	**5.87**

ii. Data Generated in Process

- **Multiprocessing:**
  - **Gradient Descent**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Dataset 1	**10.40**	**5.6**	**8.93**	**5.18**	**5.64**	**10.55**	**11.89**	**8.32**
Dataset 2	**29.59**	**15.25**	**20.35**	**24.31**	**13.87**	**9.04**	**31.97**	**20.63**
Dataset 3	**76.67**	**22.46**	**73.72**	**33.62**	**25.05**	**60.56**	**59.21**	**50.19**
Dataset 4	**124.55**	**155.93**	**141.3**	**78.21**	**153.3**	**121.31**	**71.74**	**120.92**
Dataset 5	**338.4**	**108.86**	**46.61**	**25.95**	**114.47**	**33.64**	**41.55**	**101.36**

  - **Nesterov Accelerate Gradient**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Dataset 1	**9.09**	**10.44**	**8.22**	**8.67**	**3.78**	**4.77**	**5.37**	**7.19**
Dataset 2	**13.75**	**20.24**	**16.79**	**19.67**	**18.6**	**17.5**	**21.53**	**18.3**
Dataset 3	**30.34**	**31.32**	**31.02**	**30.93**	**35.77**	**31.54**	**36.87**	**32.55**
Dataset 4	**71.11**	**52.53**	**52.85**	**61.52**	**58.65**	**48.24**	**50.55**	**56.5**
Dataset 5	**20.95**	**36.04**	**42.87**	**35.9**	**40.94**	**43.41**	**40.14**	**37.18**

- **Concurrent.futures:**
  - **Gradient Descent**

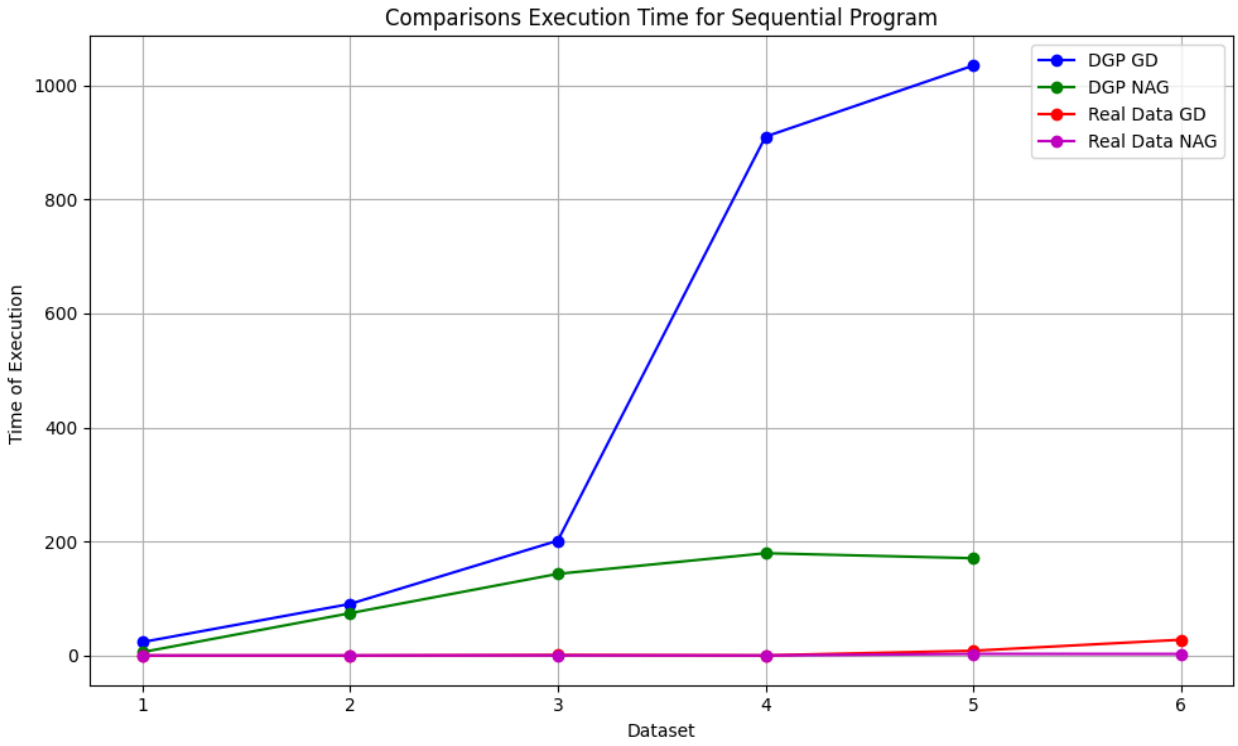
	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Dataset 1	**15.4**	**5.55**	**8.78**	**5.33**	**6.24**	**10.58**	**10.58**	**8.78**
Dataset 2	**33.02**	**16.25**	**21**	**27.88**	**13.98**	**9.26**	**23.21**	**20.94**
Dataset 3	**83.91**	**23.65**	**68.25**	**34.12**	**20.77**	**59.02**	**55.74**	**49.92**
Dataset 4	**109.27**	**152.99**	**116.68**	**67.48**	**133.9**	**105.45**	**66.44**	**107.46**
Dataset 5	**286.25**	**120.14**	**56.82**	**22.54**	**70.9**	**26.58**	**38.74**	**88.71**

  - **Nesterov Accelerate Gradient**

	4 CPU	6 CPU	8 CPU	10 CPU	12 CPU	14 CPU	16 CPU	Means
Dataset 1	**13.75**	**9.8**	**8.71**	**8.38**	**3.87**	**4.02**	**4.84**	**7.48**
Dataset 2	**17.55**	**21.75**	**18.1**	**19.08**	**18.29**	**17.97**	**18**	**18.68**
Dataset 3	**33.34**	**32.75**	**33.74**	**37.41**	**32.65**	**30.68**	**34.32**	**33.7**
Dataset 4	**64.78**	**53.74**	**50.37**	**51.52**	**48.1**	**49.18**	**45.93**	**51.37**
Dataset 5	**18.04**	**41.09**	**42.75**	**28.93**	**33.28**	**35.97**	**36.09**	**33.16**

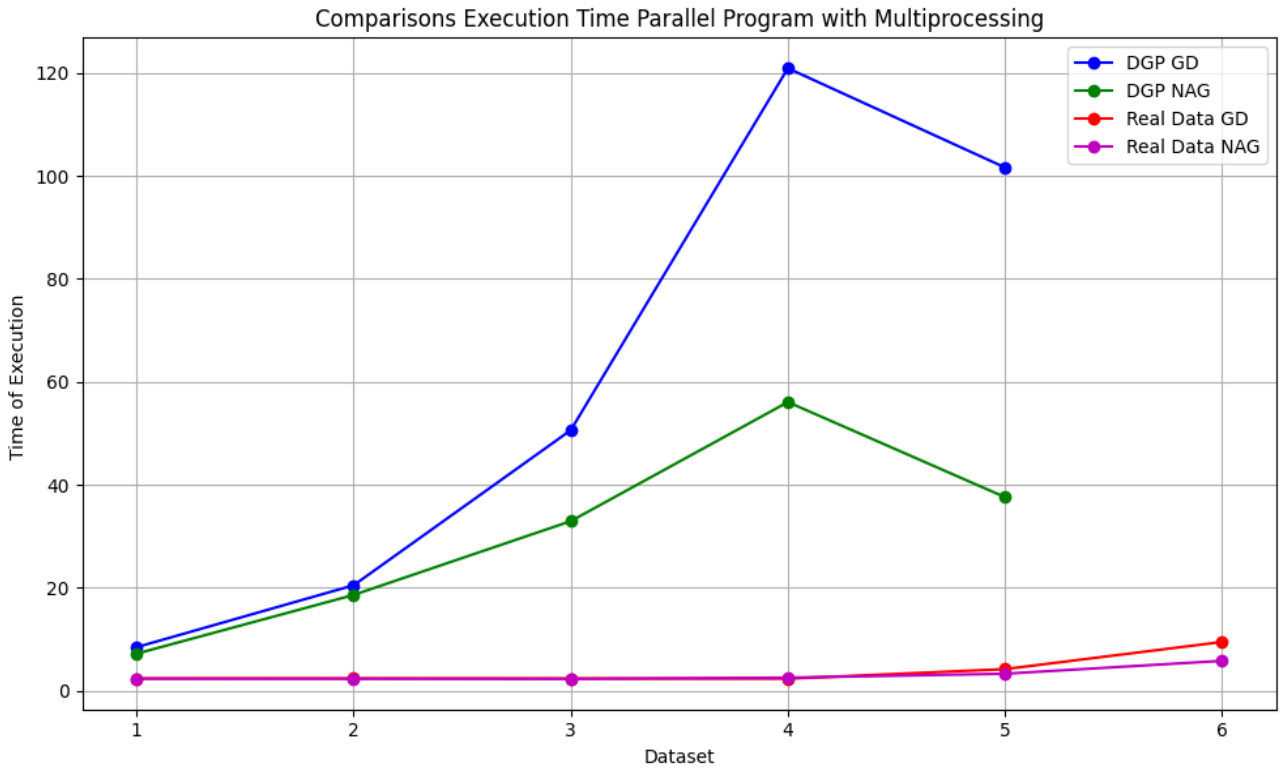
6. Performance Analysis

a. Execution Time Sequential Program

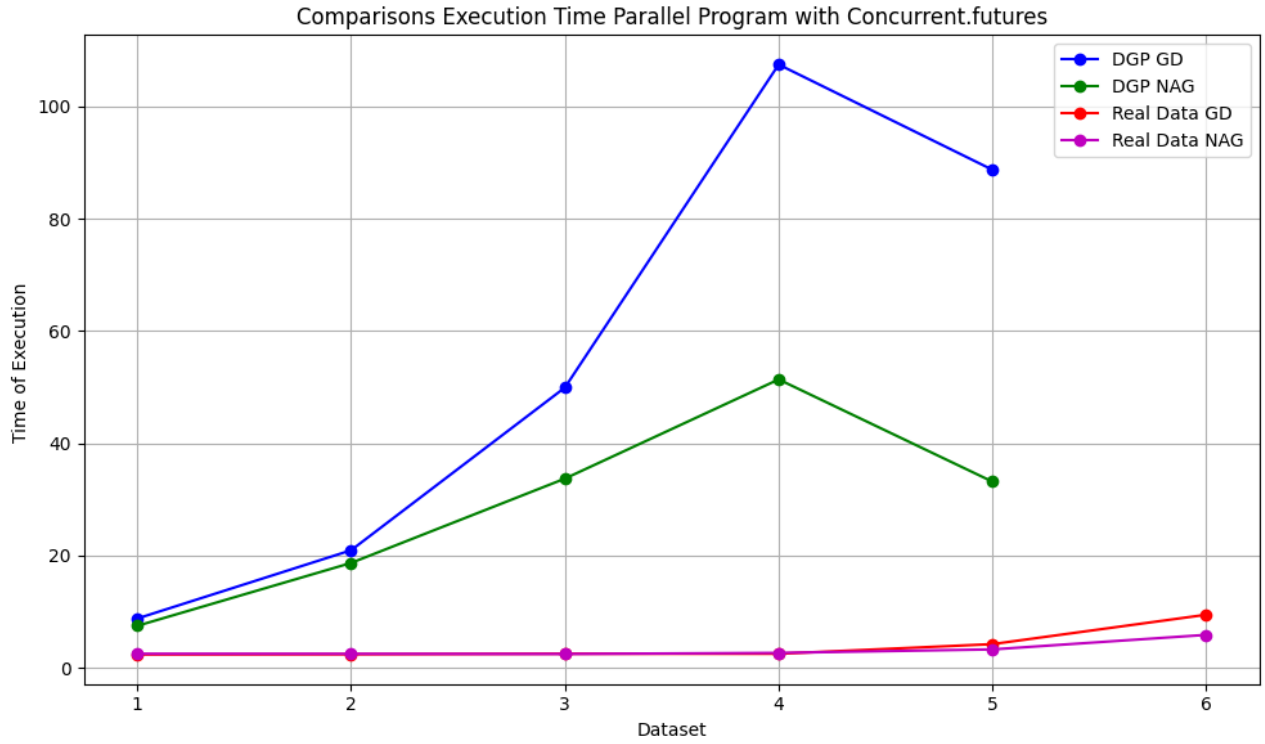


b. Average Execution Time for Parallel Program

i. Multiprocessing

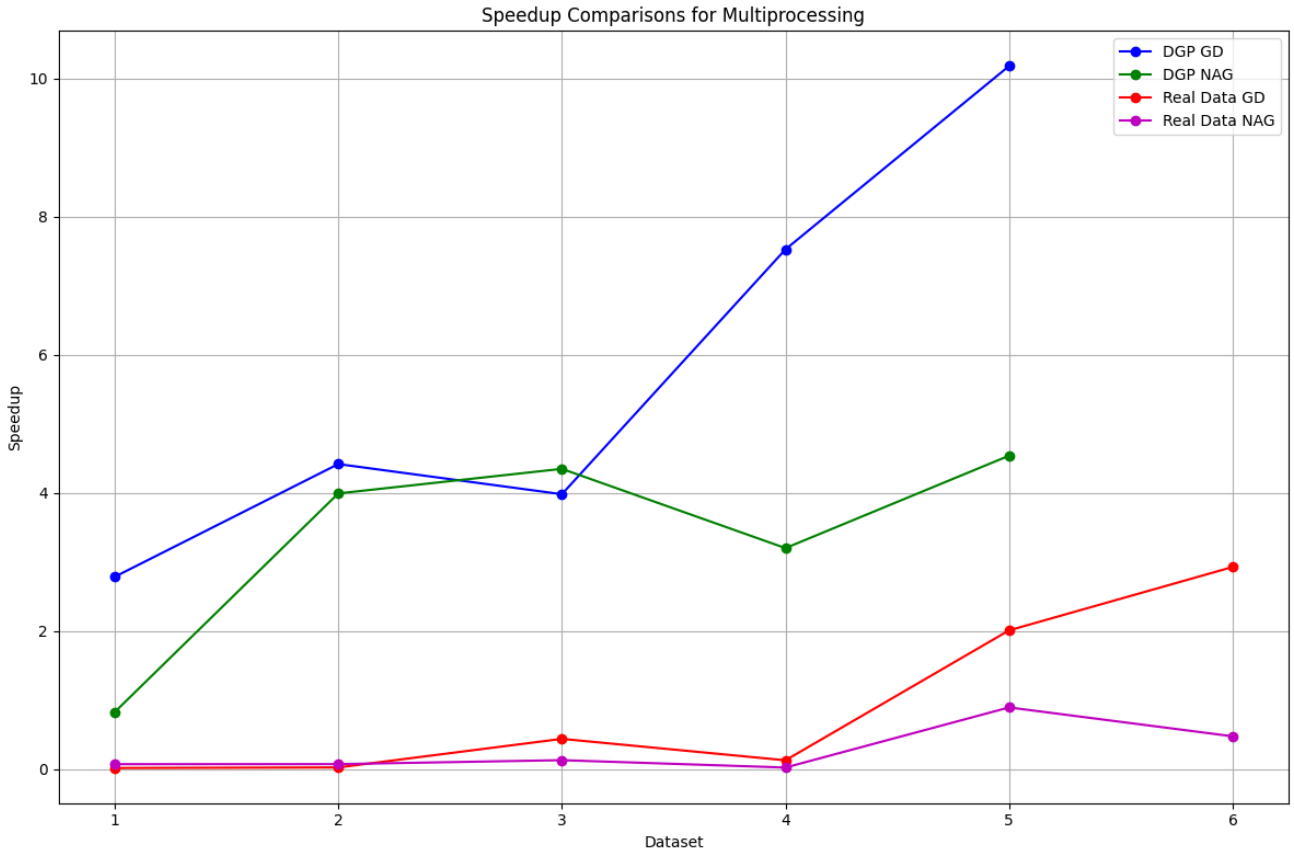


ii. Concurrent.futures

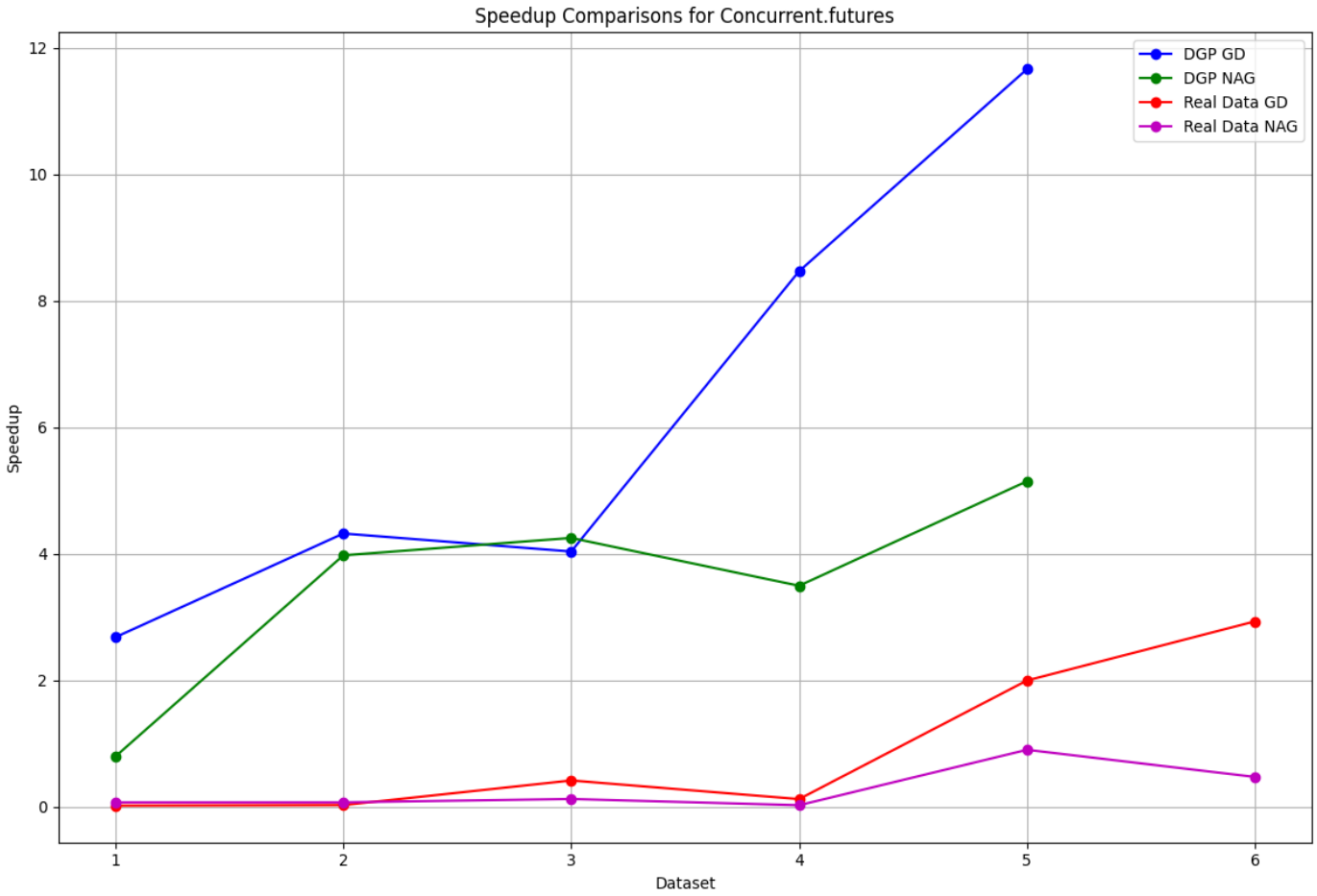


c. Speedup comparisons

i. Multiprocessing



ii. Concurrent.futures



7. Conclusion

The results clearly show the performance improvements achieved through parallel processing. Both Gradient Descent and Nesterov Accelerate Gradient algorithms benefit significantly from using multiple CPU cores. Nesterov Accelerate Gradient consistently outperforms Gradient Descent in terms of execution time, both in sequential and parallel implementations. This study demonstrates the effectiveness of parallel computing in optimizing computationally intensive tasks, making it a valuable approach for handling large datasets.