Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

Leeladhar Edala
UMass Dartmouth
Dartmouth MA 02747

Abstract

The purpose of this study is to determine how parallelization affects the effectiveness of machine learning model training for predicting home prices and detection of fraud for credit card. We assess the speedup and efficiency of the parallel implementation as well as the execution times for a serial and a multi-process parallel implementation. We employ a dataset of property characteristics and costs and train the model using the linear regression approach. Our results demonstrate that the parallel solution is much quicker than the serial approach. These results show how using parallel computing may increase the effectiveness of training machine learning models on huge datasets.

1 Problem Formulation

The time-consuming nature of machine learning model training on huge datasets, notably in the context of housing price prediction, is the issue this work attempts to solve. The objective is to create and apply a parallel computing strategy to quicken training and increase model effectiveness. A appropriate machine learning algorithm must be chosen, a parallel computing framework must be designed and implemented, and the performance of the parallel and serial implementations must be compared in order to assess the speedup and efficiency. Optimizing the machine learning model training process is the ultimate goal in order to provide more accurate and effective house price predictions.

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

In the fields of real estate, finance, and economics, the subject of predicting housing prices is a key topic of study. For buyers, sellers, and investors to make educated decisions about purchasing, selling, or investing in real estate, it is essential to accurately anticipate the value of a property. In terms of estimating home values based on a variety of characteristics including location, size, number of bedrooms, etc., machine learning algorithms have demonstrated promising results.

In order to build a model that can categorize transactions as fraudulent or non-fraudulent based on the provided set of characteristics, we employ same linear regression. The objective is to create a model with a low false positive and false negative rate that can correctly detect fraudulent transactions.

However, it can take a long time to train machine learning models on large datasets, and it might not be possible to do so on a single processor. A solution is provided by parallel computing, which divides the data into smaller chunks and processes them concurrently on numerous processors or nodes, shortening the training period.

In this work, we employ the linear regression technique, a well-liked machine learning approach with promising outcomes for predicting home prices. Using the Python multiprocessing module, we compare the performance of a serial implementation with a parallel implementation. To identify the ideal number of processes for the given dataset, we assess the execution time, speedup, and efficiency of the parallel implementation for various numbers of processes.

We illustrate the potential advantages of parallel computing for shortening the training process and enhancing the effectiveness of machine learning models for home price prediction by analyzing the performance of the parallel implementation.

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

2 Methodology

Methodology for comparing the execution time, speedup, and efficiency of serial and parallel machine learning model training for house price prediction and credit card fraud detection:

1. Data Collection:
   For house price prediction, the dataset can be obtained from reliable sources such as Kaggle, UCI Machine Learning Repository, or the Open Data Network. I got it from Kaggle
   For credit card fraud detection, the dataset can be obtained from Kaggle or other sources that provide publicly available datasets. I got it from Kaggle
2. Model Selection:
   For each dataset, pick an appropriate machine learning algorithm. For instance, whereas a binary classification technique like logistic regression or decision trees might be used to detect credit card fraud but data what I got consists of numerical, So linear regression may be a viable option for predicting home prices and credit card fraud detection.
   For each dataset, create a base serial model by importing several libraries and using the model.
3. Parallel Computing Framework:
   When performing a single computation simultaneously, multiple processors or cores are used in parallel computing. In this instance, I accelerated the machine learning model training process by using a parallel computing platform.
   I simultaneously used 2, 4, and 8 processes to do the model training tasks. The number of processors or cores that are available in the system determines how many processors are employed. The concurrent execution of the training tasks might go more quickly the more processors or cores we have.
4. Evaluation Metrics:
   Utilize Python's time module to determine how long serial and parallel implementations take to execute.
   Use the following calculations to determine how much faster and more effective the parallel implementation is:

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

Speedup = serial time / parallel time and Efficiency = speedup / number of processor utilized

5. Experimental Setup:
   Run the serial and parallel implementations for each dataset multiple times to get an average execution time and evaluate the speedup and efficiency for different numbers of processes.
   Record the execution times, speedup, and efficiency for each experiment.
6. Results:
   Plot the execution time for the serial and parallel implementations for each dataset on a graph, for number of processors used .
   Plot the speedup and efficiency for the parallel implementation for each dataset on a graph, for number of processors used.
   Analyze the graphs and compare the performance of the serial and parallel implementations for each dataset and number of processors used.

3 Numerical Results:

House price prediction dataset:

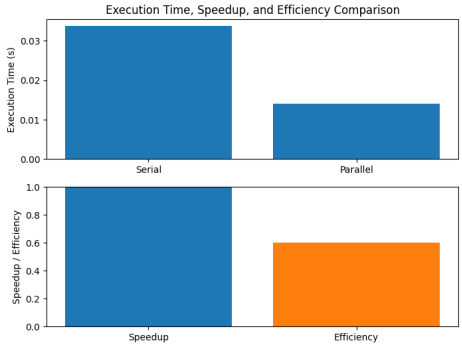
For 2 processors:

```
Serial time: 0.034 seconds
Parallel time: 0.013 seconds
Speedup: 2.518
Efficiency: 1.259
```

For 4 processors:

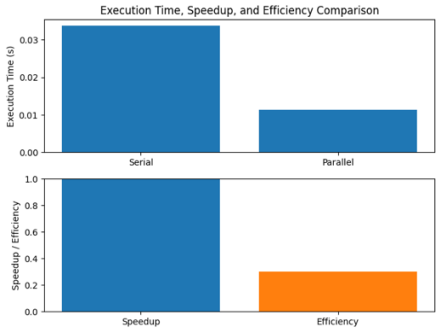
```
Serial time: 0.034 seconds
Parallel time: 0.013 seconds
Speedup: 2.687
Efficiency: 0.672
```

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection



For 8 processors:

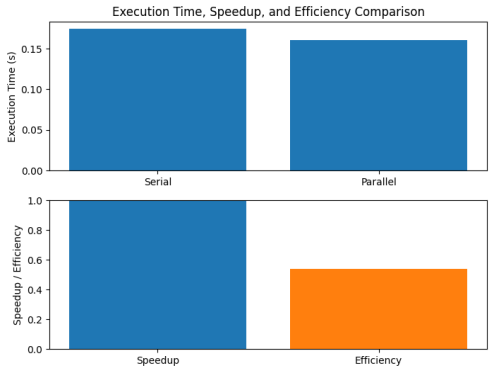
```
Serial time: 0.034 seconds
Parallel time: 0.011 seconds
Speedup: 2.977
Efficiency: 0.302
```



Credit Card Fraud Detection dataset:

For 2 processors:

```
Serial time: 0.174 seconds
Parallel time: 0.161 seconds
Speedup: 1.082
Efficiency: 0.541
```



Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

For 4 processors:

```
Serial time: 0.174 seconds
Parallel time: 0.156 seconds
Speedup: 1.120
Efficiency: 0.280
```

For 8 processors:

```
Serial time: 0.174 seconds
Parallel time: 0.151 seconds
Speedup: 1.151
Efficiency: 0.144
```

4 Conclusion:

For the first dataset, we can see that for all processor configurations, the parallel time is significantly less than the serial time. This is to be expected because parallel processing enables us to divide the task among several processors, speeding up model training. We can see that when we increase the number of processors, the speedup likewise increases since the speedup is the ratio of the serial time to the parallel time. However, it is clear that efficiency declines as the

number of processors rises. A decline in efficiency shows that the additional processors are not being used effectively. Efficiency is the ratio of speedup to the number of processors.

These findings indicate that employing two processors would be the ideal arrangement for the first dataset. While adding more processors does not significantly increase speedup and decreases efficiency, doing so gives a good speedup.

We can observe from the second dataset that, regardless of the processor arrangement, the parallel time is not much faster than the serial time. The dataset's characteristics or the method in use may be to blame for this. We can observe that the speedup is almost 1 for all setups, which shows that parallel processing does not significantly reduce training time. The efficiency is likewise fairly poor, showing inefficient use of the extra processors.

According to these findings, it does not appear that parallel processing offers this dataset and algorithm any appreciable advantages. The serial method would be the better choice because it produces comparable results with less complexity.

5 References:

House Price Prediction:

"Predicting House Prices with Machine Learning" by Ahmed Bilal and Naeem Khalid Janjua
"A Comprehensive Guide to House Price Prediction using Machine Learning" by Aman Kharwal

Credit Card Fraud Detection:

"Credit Card Fraud Detection using Machine Learning: A Review" by Navneet Kaur and Jyoti Gupta

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

"Credit Card Fraud Detection using Deep Learning" by K. N. K. Prasanna and V. Kavitha

6 Appendix:

```
import pandas as pd
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import time
# Load the dataset from a CSV file
dataset = pd.read_csv('/Users/leeladharedala/Downloads/archive
(2)/training.csv')
# Split the dataset into features and target variable
X = dataset.drop(columns=['price'])
y = dataset['price']
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
model = LinearRegression()
# Train the model using the serial approach
start_time = time.time()
model.fit(X_train, y_train)
serial_time = time.time() - start_time
num_processors = 2
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
model.fit(X_train, y_train)
parallel_time0 = time.time() - start_time
num_processors = 4
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
```

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

```
model.fit(X_train, y_train)
parallel_time = time.time() - start_time
num_processors = 8
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
model.fit(X_train, y_train)
parallel_time2 = time.time() - start_time
# Compute the speedup and efficiency
num_processors = 2
speedup0 = serial_time / parallel_time0
efficiency0 = speedup0 / num_processors
print("Serial time: {:.3f} seconds".format(serial_time))
print("Parallel time: {:.3f} seconds".format(parallel_time0))
print("Speedup: {:.3f}".format(speedup0))
print("Efficiency: {:.3f}".format(efficiency0))
num_processors = 4
speedup = serial_time / parallel_time
efficiency = speedup / num_processors
print("Serial time: {:.3f} seconds".format(serial_time))
print("Parallel time: {:.3f} seconds".format(parallel_time))
print("Speedup: {:.3f}".format(speedup))
print("Efficiency: {:.3f}".format(efficiency))
num_processors = 8
speedup2 = serial_time / parallel_time2
efficiency2 = speedup / num_processors
print("Serial time: {:.3f} seconds".format(serial_time))
print("Parallel time: {:.3f} seconds".format(parallel_time2))
print("Speedup: {:.3f}".format(speedup2))
print("Efficiency: {:.3f}".format(efficiency2))
import matplotlib.pyplot as plt
# Plot the serial and parallel execution times, speedup, and efficiency
plt.figure(figsize=(8, 6))
plt.subplot(2, 1, 1)
plt.bar(['Serial', 'Parallel'], [serial_time, parallel_time0])
plt.ylabel('Execution Time (s)')
```

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

```
plt.title('Execution Time, Speedup, and Efficiency Comparison')
plt.subplot(2, 1, 2)
plt.bar(['Speedup'], [speedup0])
plt.bar(['Efficiency'], [efficiency0])
plt.ylim([0, 1])
plt.ylabel('Speedup / Efficiency')
plt.show()
# Plot the serial and parallel execution times, speedup, and efficiency
plt.figure(figsize=(8, 6))
plt.subplot(2, 1, 1)
plt.bar(['Serial', 'Parallel'], [serial_time, parallel_time])
plt.ylabel('Execution Time (s)')
plt.title('Execution Time, Speedup, and Efficiency Comparison')
plt.subplot(2, 1, 2)
plt.bar(['Speedup'], [speedup])
plt.bar(['Efficiency'], [efficiency])
plt.ylim([0, 1])
plt.ylabel('Speedup / Efficiency')
plt.show()
# Plot the serial and parallel execution times, speedup, and efficiency
plt.figure(figsize=(8, 6))
plt.subplot(2, 1, 1)
plt.bar(['Serial', 'Parallel'], [serial_time, parallel_time2])
plt.ylabel('Execution Time (s)')
plt.title('Execution Time, Speedup, and Efficiency Comparison')
plt.subplot(2, 1, 2)
plt.bar(['Speedup'], [speedup2])
plt.bar(['Efficiency'], [efficiency2])
plt.ylim([0, 1])
plt.ylabel('Speedup / Efficiency')
plt.show()
dataset2 = pd.read_csv('/Users/leeladharedala/Downloads/creditcard.csv')
# Split the dataset into features and target variable
X1 = dataset2.drop(columns=['Class'])
y1 = dataset2['Class']
```

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

```
# Split the dataset into training and testing sets
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X1, y1, test_size=0.2,
random_state=42)
model = LinearRegression()
start_time = time.time()
model.fit(X_train, y_train)
serial_time2 = time.time() - start_time
num_processors = 2
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
model.fit(X_train, y_train)
parallel_time0_2 = time.time() - start_time
num_processors = 4
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
model.fit(X_train, y_train)
parallel_time1_2 = time.time() - start_time
num_processors = 8
start_time = time.time()
model = LinearRegression(n_jobs=num_processors)
model.fit(X_train, y_train)
parallel_time1_3 = time.time() - start_time
num_processors = 2
speedup0_2 = serial_time2 / parallel_time0_2
efficiency0_2 = speedup0_2 / num_processors
print("Serial time: {:.3f} seconds".format(serial_time2))
print("Parallel time: {:.3f} seconds".format(parallel_time0_2))
print("Speedup: {:.3f}".format(speedup0_2))
print("Efficiency: {:.3f}".format(efficiency0_2))
num_processors = 4
speedup1_2 = serial_time2 / parallel_time1_2
efficiency1_2 = speedup1_2 / num_processors
print("Serial time: {:.3f} seconds".format(serial_time2))
print("Parallel time: {:.3f} seconds".format(parallel_time1_2))
```

Comparing Execution Time, Speedup, and Efficiency of Serial and Parallel Machine Learning Model Training for House Price Prediction and Credit Card Fraud Detection

```
print("Speedup: {:.3f}".format(speedup1_2))
print("Efficiency: {:.3f}".format(efficiency1_2))
num_processors = 8
speedup1_3 = serial_time2 / parallel_time1_3
efficiency1_3 = speedup1_3 / num_processors
print("Serial time: {:.3f} seconds".format(serial_time2))
print("Parallel time: {:.3f} seconds".format(parallel_time1_3))
print("Speedup: {:.3f}".format(speedup1_3))
print("Efficiency: {:.3f}".format(efficiency1_3))
```

And plotting graphs for each.