Documentation on Parallel Processing Optimization for Chess Analysis

# Objective

The goal of this project was to optimize the analysis of chess game data by leveraging parallel processing techniques. The primary focus was to reduce the time taken to process large datasets of chess games using the Stockfish engine for game analysis. We aimed to identify and mitigate bottlenecks in the processing pipeline and evaluate the efficiency gains from using multiple CPU cores.

# Initial Setup and Bottleneck Identification

## 1. Initial Performance Measurement:

- The first step involved measuring the performance of the existing single-threaded application. We used Python's `time` module to capture the duration of processing a set number of games. This measurement served as our baseline for future comparisons.

## 2. Bottleneck Detection:

- Using profiling tools and manual inspection, we identified that the Stockfish chess engine integration was a significant bottleneck. Each game analysis required initializing and querying the Stockfish engine, which was computationally expensive and time-consuming.

- Additionally, we identified that data serialization and deserialization, particularly with the JSON format for storing pre-calculated evaluations, could potentially hinder performance when scaled across multiple processes.

# Parallel Processing Implementation

## 1. Parallel Strategy Development:

- We decided to implement parallel processing using Python’s `concurrent.futures` module, specifically the `ProcessPoolExecutor`, which facilitates easy distribution of tasks across multiple processors.

- We divided the dataset into chunks, with each chunk being processed by a separate processor. This approach aimed to utilize CPU resources better and decrease overall processing time.

## 2. Implementation Steps:

- **Stockfish Instance Management:** Ensured each parallel process had its independent Stockfish instance to avoid contention.

- **Data Chunking:** Implemented a mechanism to split the game data evenly among the available processors. We used slicing based on the total number of games and the number of processors.

- **Caching corrected:** All the analyzer’s were useing the same cache, which was not a problem by itself, but more problematic was the aspect, that were adding new indexes to the cached json. Which resulted in the other processes waiting on the change, slowing the system dramatically down.

- Dummy Analysis Function: To isolate the performance improvements due to parallel processing from the overhead introduced by Stockfish, we created a simplified version of the analysis function that simulated data processing without actual game analysis. This function generated random evaluation values as placeholders for actual Stockfish output.

## 3. Testing Parallel Efficiency:

- We conducted controlled tests to measure the performance improvements with varying numbers of processors. Tests were carried out with 1, 2, 3, and 4 cores to observe the scaling behavior.

- Each test involved processing a fixed number of games, and the time taken for each configuration was recorded and compared against the baseline.

# Results and Observations

- The results indicated a non-linear speedup. While using more cores reduced the processing time, the rate of decrease in time was not proportional to the number of cores added. This outcome highlighted the diminishing returns of adding more processing power beyond a certain point, likely due to polution of the hsot system, more than 4 cores were not benefitting the run time. But this most likely will not happen on a clean system, like an capsuled system running a server with only this application on.

Therefore for further testing we will perceed running on 4 cores and when deploying the application on the server we will increase to 8.

**Times of test\_file:**

Time taken with 1 cores: 51.95 seconds

Time taken with 2 cores: 38.27 seconds

Time taken with 3 cores: 28.66 seconds

Time taken with 4 cores: 31.93 seconds

**stockfish\_analyzer.py:**

75'000 positions:

Processors: 1 847.73 seconds

Processors: 4 424.79 seconds

Processors: 8 379.44 seconds

## Conclusions and Future Work

- The parallel processing implementation successfully reduced the total processing time compared to the single-threaded approach. However, the efficiency gain was less than linear, suggesting room for further optimization.

- Future efforts could focus on optimizing the data handling and management of Stockfish instances to reduce overhead. Additionally, exploring other models of concurrency, like asynchronous programming or threading (for I/O-bound tasks), might yield better performance for certain parts of the workflow.

- Continued profiling and bottleneck analysis will be crucial as we scale the application to handle larger datasets or integrate more complex analysis features.

This documentation provides a structured overview of our approach to implementing and evaluating parallel processing in the chess game analysis project. It reflects our commitment to continuous improvement and efficiency in handling computationally intensive tasks.