**CCT College Dublin**

**Assessment Cover Page**

|  |  |
| --- | --- |
| **Module Title:** | Concurrent Systems |
| **Assessment Title:** | CA1: Application of Concurrency to Common Tasks |
| **Lecturer Name:** | Sam Weiss |
| **Student Full Name:** | Amanda de Toledo Ferraz |
| **Student Number:** | 2021256 |
| **Assessment Due Date:** | 24/03/2024 |
| **Date of Submission:** | 24/03/2024 |

**Declaration**

|  |
| --- |
| By submitting this assessment, I confirm that I have read the CCT policy on Academic Misconduct and understand the implications of submitting work that is not my own or does not appropriately reference material taken from a third party or other source. I declare it to be my own work and that all material from third parties has been appropriately referenced. I further confirm that this work has not previously been submitted for assessment by myself or someone else in CCT College Dublin or any other higher education institution. |

**APPLICATION OF CONCURRENCY TO COMMON TASKS**

Concurrency vs. Single-Threaded Approach: Tradeoff Analysis

**GitHub repository link:** <https://github.com/CCT-Dublin/ca1-AmandaTFerraz.git>

**Concurrency vs. Single-Threaded Approach: Tradeoff Analysis**

**Introduction**

In today’s computing landscape, the demand for efficient programming has led us to the use of concurrent programming. Concurrency enables programs to perform multiple tasks simultaneously, through multi-threading, improving performance and responsiveness. However, the adoption of concurrent approaches may bring tradeoffs that must be carefully considered.

**Task 1: Find standard deviation**

**Advantages:** By distributing the workload across multiple threads, it can significantly accelerate computation, having faster processing times, particularly for large datasets. I can also potentially improve overall system performance with the use of modern processors with multiple cores.

**Disadvantages:** The use of multi-threaded computation brings complexity, requiring careful management of synchronization and thread interactions. It can make de code harder to develop, debug, and maintain. It can also overbalance the performance gains, especially for smaller datasets or simpler calculations, where the overhead of thread creation and management can be high compared to computational workload.

**Task 2: Multiply matrices**

**Advantages:** It enables parallelism, allowing different parts of the program to execute simultaneously. This can bring significant speed improvements, especially for large matrices where the workload can be divided among threads. Besides that, it facilitates efficient resource utilization by harnessing multiple threads to fully utilize available CPU cores.

**Disadvantages:** Synchronization overhead happens when managing access to shared data. This extra work to coordinate them, like when they need to update the result matrix, can slow down the process and cancel out the benefits of running tasks in parallel. Also, when we have too many threads, there's a risk of reaching a limit where they start to compete for shared resources or face problems with the amount of data they can handle.

**Task 3: Merge sort implementation for sorting data in descending order**

**Advantages:** It enables faster sorting by leveraging to process different parts of the array simultaneously, reducing sorting times, especially for large datasets. Concurrency also enhances system responsiveness by allowing concurrent execution of other tasks while sorting.

**Disadvantages:** Implementing merge sort with concurrency can be complex, requiring careful management of recursive splitting and merging operations, which may introduce additional complexity and overhead to the codebase. It also, there is a risk of potential load imbalance among threads, particularly if the workload distribution or partitioning strategies are not optimized, reducing sorting efficiency.

**Tradeoffs of using concurrency instead of a single-threaded program for reading data from the CSV file**

**Advantages:** Concurrency enables faster data processing by allowing multiple threads to read data simultaneously, particularly beneficial for large datasets. It also optimizes resource utilization by leveraging CPU cores efficiently and enhances system responsiveness by enabling concurrent execution of other tasks alongside file reading.

**Disadvantages:** Implementing concurrency adds complexity to the codebase, necessitating careful management of threads and synchronization. Resource contention among threads and potential overhead from managing multiple threads can also pose challenges, potentially offsetting the performance gains. Ultimately, the decision to use concurrency depends on factors such as the application requirements, dataset characteristics, and the balance between performance and complexity considerations.

**Conclusion**

While concurrency offers substantial performance benefits through parallelism and efficient resource utilization, it also introduces complexities and overhead. The decision to adopt a concurrent approach should be carefully considered based on task requirements, data characteristics, and available hardware resources.