Multithreading Application

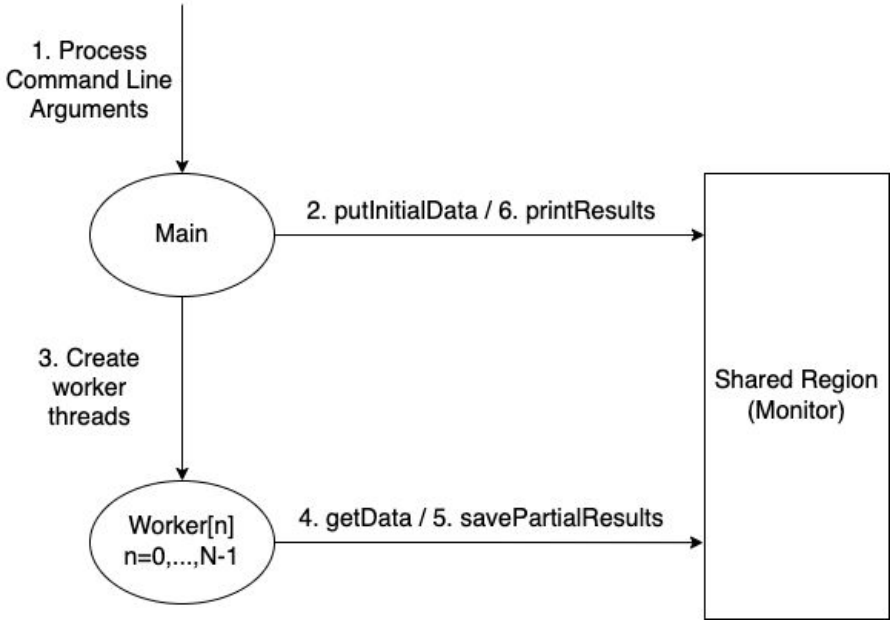
CLE - Computação em Larga Escala
Assignment 1
April 2022

Mário Silva - 93430
Pedro Marques - 92926

Text Processing - Multithreading Implementation

- **Main Objective:** Count the number of words, words starting with vowels and words ending in consonants, efficiently, by splitting processing load for worker threads.

1. Main thread processes the command line arguments.
2. Main thread initializes the shared region array of structures with the given file names.
3. Main thread creates the worker threads.
4. Workers fetch one chunk at a time from a file to process in the shared region.
5. Workers then save the results of the processing of the chunk.
6. Finally, the main thread prints the final results.



Text Processing - Results

Timing results for processing **all file texts**, with **4096 bytes maximum** size **per processing chunk**.

Basic Command:

./prog1 -f texts/text0.txt -f texts/text1.txt -f texts/text2.txt -f texts/text3.txt -f texts/text4.txt -m 4096

For **1** worker (*-n 1*):

Elapsed Time = 0.001575s

For **2** workers (*-n 2*):

Elapsed Time = 0.000976s

For **4** workers (*-n 4*):

Elapsed Time = 0.000749s

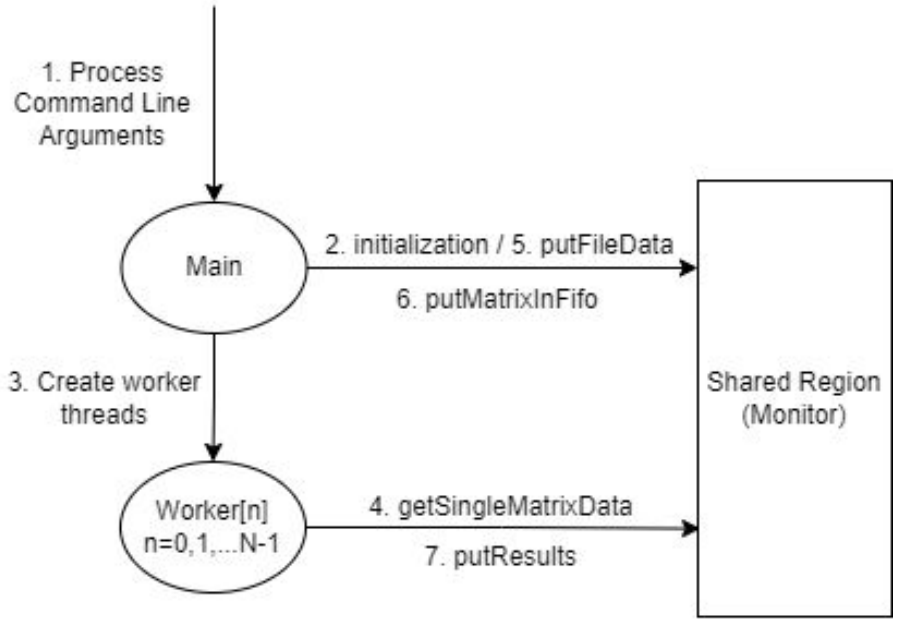
For **8** workers (*-n 8*):

Elapsed Time = 0.000742 s

Matrix Determinant - Multithreading Implementation

- **Main Objective:** Given a file with matrices, calculate the determinant of each one, efficiently, by splitting processing load for worker threads.

1. Main thread processes the command line arguments.
2. Main thread creates the worker threads.
3. Main thread reads each matrix of each file.
4. Each matrix is inserted in a *Shared Memory in a FIFO*.
5. Workers retrieve and process the matrix, calculating the **determinant**.
6. Workers insert results in the *Shared Memory*.
7. When all files have been processed, the main thread retrieves and presents the results.



Matrix Determination - Results

Timing results for processing **mat512_128.bin** and **mat512_256.bin** files, with a **FIFO queue of size 8**.

Basic Command:

./prog2 -f longMatrix/mat512_128.bin -f longMatrix/mat512_256.bin -k 8

For **1** worker (*-n 1*):

Elapsed Time = 1.24s

For **2** workers (*-n 2*):

Elapsed Time = 0.63s

For **4** workers (*-n 4*):

Elapsed Time = 0.33s

For **8** workers (*-n 8*):

Elapsed Time = 0.20 s

Conclusion

After reviewing the achieved results, we came to the following conclusions:

- With the increased number of worker threads, a clear **decrease in execution times** was registered.
- **For the first program, the difference in execution times is not as noticeably** due to the disk overhead, since the problem's nature is **not as CPU and memory intensive** as the several disk accesses to read the files.
- On the other hand, the **second program requires more CPU power**, so splitting workloads between threads significantly **improves the performance of the program**.
- In essence, we can conclude that these **multithreaded programs offer a boost in performance** when compared to their single threaded counterparts.