CLE Assignment 1

Vowel Count and Bitonic Sorting

Afonso Campos - 100055
Simão Arrais - 85132

27/03/2023

Vowel Count - Data Transfer Policy

Shared Memory:

- (char **) fileNames[fileNum]
- (uns. int *) wordCount[fileNum]
- (uns. int *) vowelCounts[fileNum][vowelNum]
- (uns. int *) fileBuffer[fileNum] // number of bytes processed per file
- (bool *) fileOver[fileNum]
- (int *) currFileWorker[workerNum] // stores the index of the file last processed/currently being processed by a worker
- (int) currFile // incates the next file to be processed
- (bool) workFinished

Main Lifecycle:

- process user input;
- storeFileNames(char ** names);
- create worker threads, await their termination;
- printResults().

Worker Lifecycle:

- readFromFile(uint id, uchar * chunk, int * chunkSize);
- if the above function returns true, quit.
- else:
  - process text chunk;
  - updateCounts(uint id, uint wordCount, uint * voweCount);
  - restart lifecycle.

Vowel Count - Implementation Details

Text Chunk Integrity:

- retrieve text chunk from file with pre-determined byte size;
- if the EOF was reached, nothing was cut.
- else, we look at the next unread byte, if it's the start of a separator, nothing was cut.
- else, something was cut!
- to solve this, we iterate over the captured text chunk in inverted order, until we find a separator character, once we do, that becomes our cutting point for the text chunk.

File Processing Order:

- processing starts in file with index 0;
- if during the read operation a file reached EOF, it's marked as finished (fileFinished);
  - if all files are marked as finished, workFinished is set to true;
  - else, currFile is increased by one.
- at the start of readFromFile, workers check if workFinished is set to true, if so, the worker quits.

Vowel Count - Results

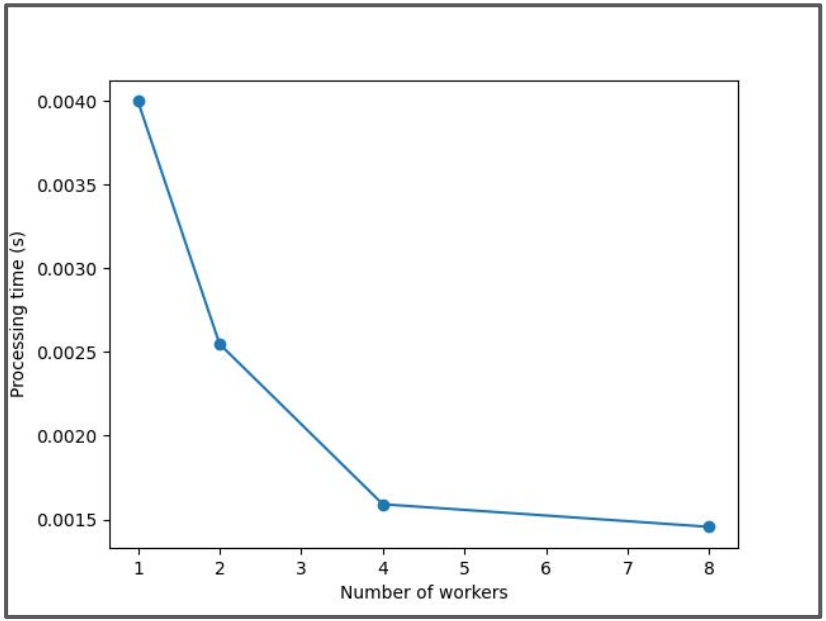
Benchmarks for the processing of the 5 text files provided in class:

1 worker	2 workers	4 workers	8 workers
0.003997s	0.002548s	0.001591s	0.001456s

As seen, constantly doubling the number of workers has diminishing returns;

Eventually, adding more workers would produce the reverse effect: processing time would increase;

This is due to syncronization and mutual exclusion, syncronizing the workers takes time as well.



Bitonic Sorting - Data Transfer Policy

Shared Memory:

- (char *) fileName
- (int) sequenceSize
- (int *) sequence[sequenceSize]
- (uint **) workerRange[workerNum][2]
- (int *) workerCommand[workerNum]
- (int) waitingWorkers
- (int) finishedWorkers

Main Lifecycle:

- process user input;
- storeFileName(char * name);
- create worker and distributor threads, await their termination;
- validateSequence().

Distributor Lifecycle:

- readFromFileAndStore();
- while there is one, or more, active workers distributeRanges();

Worker Lifecycle:

- fetchSubSequence(uint id, int * command, int * chunkSize, int ** chunk);
- if the above function returns true, quit.
- else:
  - sort sequence chunk according to command;
  - signalFinished(uint id);
  - restart lifecycle.

Bitonic Sorting - Implementation Details

Synchronization (pthread conditions):

- allWorkersWaiting: distributor waits until all active workers are available/waiting for work before distributing worker commands and ranges;
- allWorkersFinished: after distributing ranges, the distributor waits for all workers to finish sorting their respective sub-sequences and then, reduces the number of active workers in half with the DIE command.
- waitForWork: workers wait for the distributor to signal the worker ranges and commands are updated for the run.

Worker Commands:

- -2: order non bitonic sequence (decreasing);
- -1: order bitonic sequence (decreasing);
- 0: worker waiting;
- 1: order bitonic sequence (increasing);
- 2: order non bitonic sequence (increasing);
- 3: die/quit.

At the start of fetchSubSequence, the workers check if their command is set to 3 (DIE) and successfully quit if so.

The distributor gives priority to the lower id workers to remain alive. It also handles all the logic in creating bitonic sequences after each run (workers with an odd id sort incr., even id decr.

Bitonic Sorting - Results

SeqSize\ThreadNum	1	2	4	8
32	0.001295	0.000740	0.000805	0.000948
256K	0.080858	0.045834	0.036262	0.028565
1M	0.291327	0.165398	0.134443	0.120831
16M	5.391008	3.104499	1.982710	2.097874