1. introduction (what is distributed computing and its types)

2. difference between shared address and distributed address space determine the type of your problem

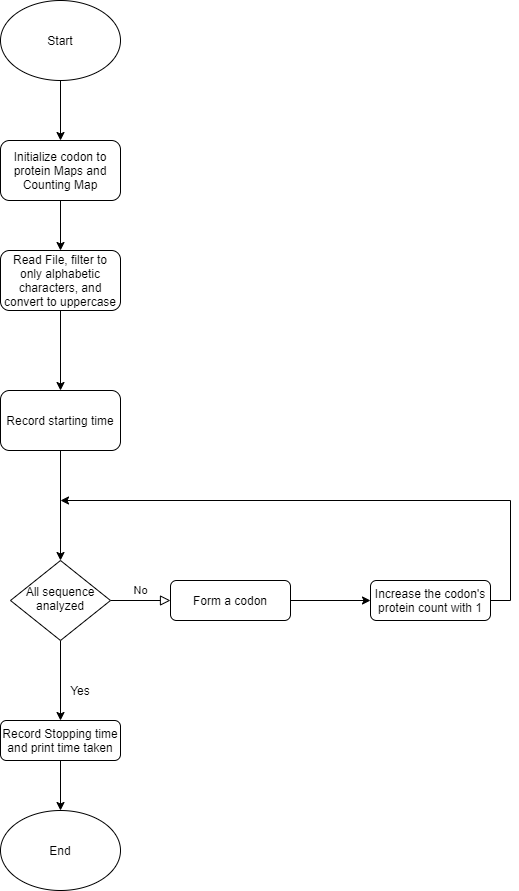
3. different applications in distributed computing

4. specific application in the applied topic

5. literature review of the selected topic

6. proposed sequential algorithm (Pseudo Code, Flow chart) and its complexity

Flow Chart



Pseudocode

sequential(string inStr)

    // Record starting time

    start <- getTime()

    for each 3 characters

        temp <- form codon from 3 characters

        // Increment protein count by 1

        countMapSeq[codonMap[temp]] += 1;

    // Record starting time

    end <- getTime()

    // Print consumed time

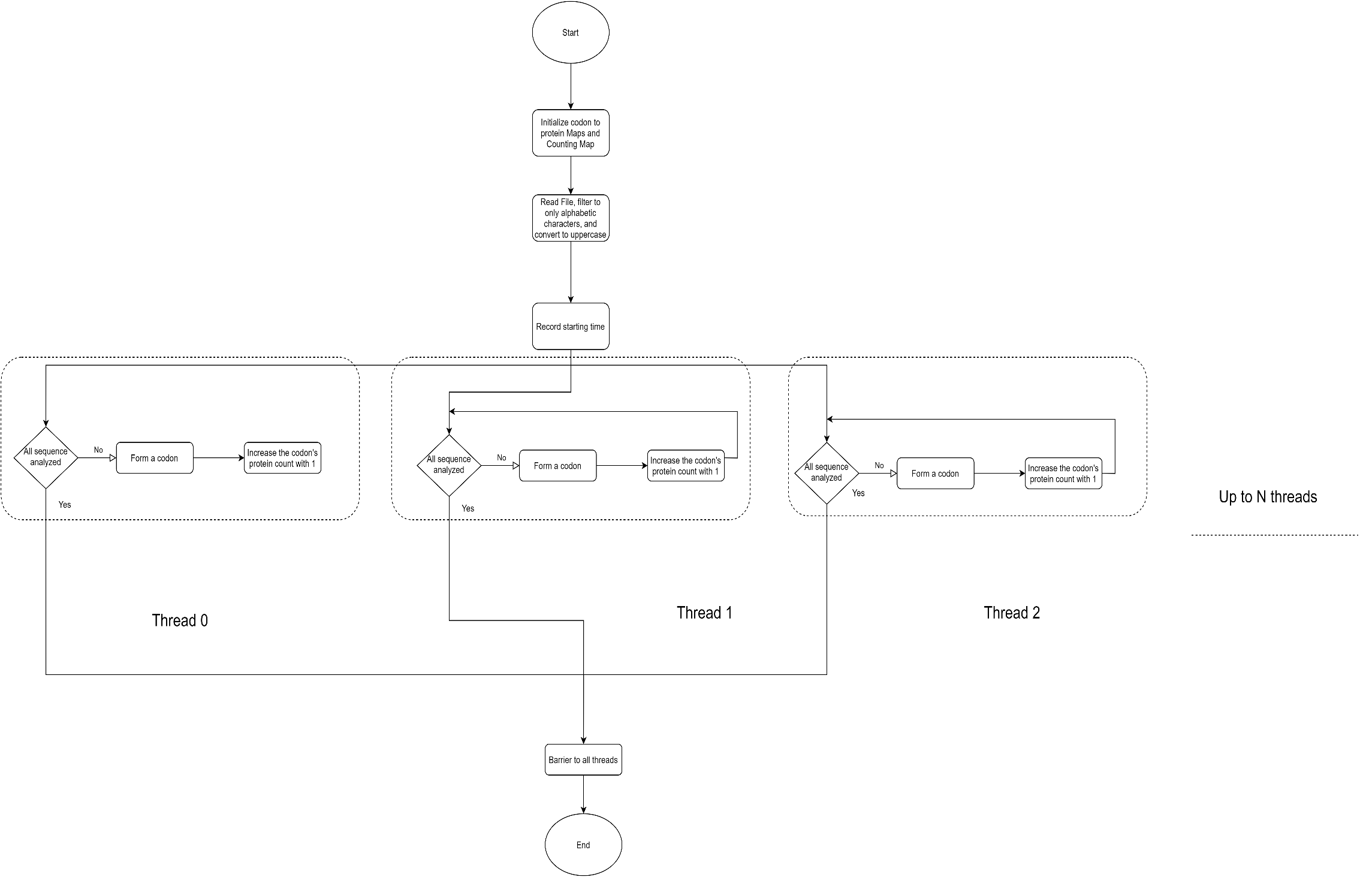
    print(end-start)

complexity

O(N)

7. proposed parallel algorithms (Pseudo Code, Flow chart) and their complexities

Loop-level parallelism Flow chart



Loop-level parallelism Pseudo Code

parallel(string inStr)

    // Record starting time

    start <- getTime()

    // Parallelize loop iterations

    for each 3 characters

        temp <- form codon from 3 characters

        // Increment protein count by 1 atomically

        countMapSeq[codonMap[temp]] += 1;

    // Record starting time

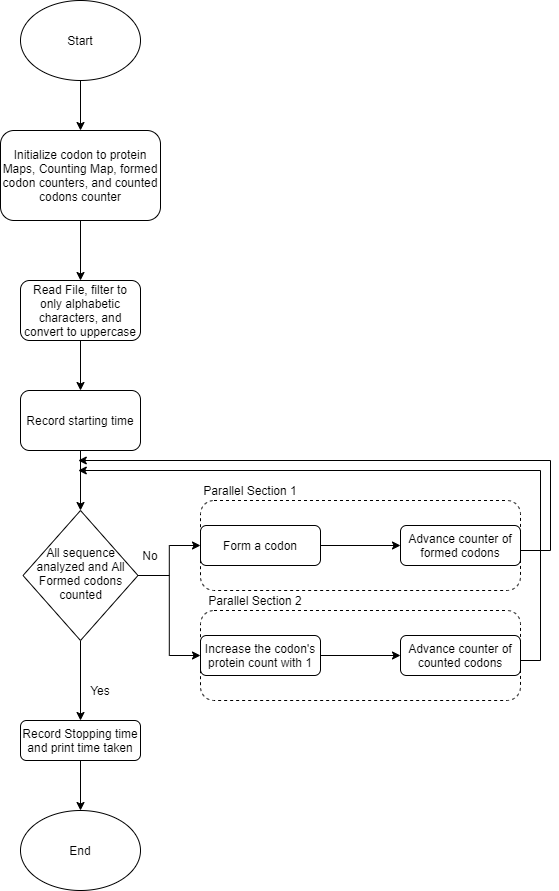
    end <- getTime()

    // Print consumed time

    print(end-start)

Loop-level parallelism complexity

O(N)

Functional decomposition parallelism Flow Chart.

Functional decomposition parallelism Pseudo code.

parallelSections(string inStr)

    // Codons Arr of size (sequence length / 3)

    codonsArr

    // Counter of formed codons

    formedCodons <- 0

    // Counter of counted codons

    countedCodons <- 0

    // Record starting time

    start <- getTime()

    // Parallel section 1, forms codons

    {

        for each 3 characters

            codonsArr[formedCodons] <- codon from 3 characters

            formedCodons <- formedCodons + 1

    }

    // Parallel section 2, counts proteins of formed codons

    {

        while( (not all codons formed) or (not all formed codons counted) )

            while(not all formed codons counted)

                countMapSeq[codonMap[codonsArr[countedCodons]]] += 1;

                countedCodons <- countedCodons+1

    }

    // Record starting time

    end <- getTime()

    // Print consumed time

    print(end-start)

complexity

O(N)

8. performance evaluation (speedup, efficiency, iso-efficiency)

**Done in excel.**

9. Used H/W Specs, results, and discussions (table with time and number of threads)

Processor Intel(R) Core (TM) i7-8550U CPU @ 1.80GHz, 4 Core(s), 8 Logical Processor(s)

Installed Physical Memory (RAM) 16.0 GB

Storage Disk Model ST1000LM035-1RK172

**Done in excel.**

Discussions:

The effect of increased number of threads on the speedup of both parallel implementations was different. The reason being, when threads were increased in the case of parallel sections, more threads made no difference as there was only 2 sections, so only the overhead of handling more threads harmed the speedup while nothing helped the speed up. On the other hand, the loop-level parallelization speedup was not monotonically decreasing/increasing when the number of threads was increased.

10. graph (time vs number of threads) for each parallel solution

**Done in excel.**

11. conclusion and comment on the results

12. appendix (code)