CAB401   
  
PARALLELIZATION PROJECT REPORT

Dac Duy Anh Nguyen n10603280

Promoter

Table of Contents

[I. Original sequential application 3](#_Toc85369899)

[1) What it does 3](#_Toc85369900)

[2) How it works 3](#_Toc85369901)

[II. Potential parallelism analysis 3](#_Toc85369902)

[III. Tools and techniques utilized 4](#_Toc85369903)

[IV. Outcomes 4](#_Toc85369904)

[V. Difficulties 4](#_Toc85369905)

[VI. Reflection 4](#_Toc85369906)

[VII. References 5](#_Toc85369907)

[VIII. Appendixes 5](#_Toc85369908)

# I. Original sequential application

## 1) What it does

The application is utilized to identify different promoters in reference list from Ecoli bacteria

## 2) How it works

Through a brief analysis in run() function, the program first loads Ecoli DNA and reference genes using ParseReferenceGenes function.

After that, in the **first** **For** **loop** (number 1), the program iterates through each Ecoli DNA file to retrieve its gene record using Parse function.

Another **inner For loop** (number 2) iterates through the reference genes to compare it with each Ecoli gene in its **inner For loop** (number 3).

The comparation process is handled by Homologous function to determine if 2 genes serve the same purpose. In this case, they are an Ecoli gene from Ecoli records (from For loop 3) and another gene from reference genes (from For loop 2).

If the Homologous function return true, the program proceeds to extract upstream region of the Ecoli gene by GetUpStreamRegion function and use it to predict if that Ecoli gene is a promoter. If true, the results are stored in ‘consensus’ variable by adding 1 to the counter of that promoter type and the total matches.

\*\*\*Watch intro video again

# II. Potential parallelism analysis

|  |
| --- |
| Your analysis of potential parallelism within the application. This might include  identification of existing loops or control flow constructure where parallelism might be found. Explanation of the data and control dependences that you analyzed to determine which sections of code were safe to parallelize. Which of these is likely to be of sufficient granularity to be worth exploiting? Is it scalable parallelism? A discussion of changes required to expose parallelism, such as replacing algorithms or code restructuring transformations.  c. map computation to processor  It uses 3-nested for loop which runs and analyzes large text files (DNA) and is being executed in a sequential way. As a consequence, the computing time takes a substantial amount of time and could be reduced if applying parallelization. |

From my initial analysis of the given sequential program, Homologous function dominantly consumes 99.7% of the CPU computation. More specifically, it’s the align function in SmithWatermanGotoh class that get called by the Homologous function.

Text

Description automatically generated with medium confidence

Graphical user interface, application

Description automatically generated

Since SmithWatermanGotoh function is too complicated to rewrite for better performance and required a deep understanding of problem, the work of this report focuses only to parallelize the app and increase performance at other parts where possible

--I’m here—

Transforming the For loops structure: move For loop 3 to inner For loop 1 and execute For loop 3 before For loop 2. However, extraction and prediction parts will remain in For loop 2. This makes the memory and processes relating to Ecoli genes data are close to each other, locality?

After the transformation above, the program can be paralleled at each of the For loops.

Identified dependencies:

* upStreamRegion
* prediction
* consensus

# III. Tools and techniques utilized

Hardware specs:

Software:

Techniques:

Parallel stream

ThreadPool

ExecutorService

Locality

# IV. Outcomes

d. timing and profiling results, speedup graphs

e. same results (number of promoters identified)?

# V. Difficulties

Identify data dependencies and make some variables ThreadLocal, avoid data race

Speedup after parallelizing 3 for loops is still not good. Identify causes…

# VI. Reflection

# VII. References

<https://www.baeldung.com/java-executor-service-tutorial>

<https://www.baeldung.com/java-when-to-use-parallel-stream>

# VIII. Appendixes

parallelStream 111 50s – 76s

A picture containing chart

Description automatically generated

Diagram, timeline

Description automatically generated

101 57s 78s 79s

Diagram

Description automatically generated

Chart

Description automatically generated with medium confidence

011 72 less overhead?

Diagram

Description automatically generated

A screenshot of a computer

Description automatically generated with medium confidence

001 77

Diagram

Description automatically generated

Chart

Description automatically generated

110 83

Thread waiting for other thread to finish

Timeline

Description automatically generated

A picture containing chart

Description automatically generated

100 108

Too many thread waiting

Timeline

Description automatically generated with medium confidence

Timeline

Description automatically generated

010 101s

Only create 5 threads

Timeline

Description automatically generated A picture containing timeline

Description automatically generated

Sheep1 62 77 78 85 50 60

Sheep2 52

Sheep2 PS 12 threads average 63s

Me 8 threads average 65s

Me 12 threads average 70s

Me 8 threads without specify average 84s