UFO - Assignment 3 - Optimization

Mathias, Magnus, Rasmus

April 2021

1 Program introduktion

Denne opgave går ud på at vi skal finde et tidligere program, som vi kan optimere. Vi har valgt at bruge det udleverede letterFrequency projekt. Derudover har vi brugt 5. udgave af Peter Sestofts benchmarking metode, modificeret så den passer til vores program. Alt koden er samlet i Main klassen. Den originale `tallyChars()` metode bliver målt hvis man bruger `mark5original()` metoden, og vores optimerede udgave bliver målt i `mark5fast()`. For at kontrollere at begge metoder resulterer i den samme optælling, findes versioner af de 2 metoder med printfunktionen tilknyttet. Vi var dog ikke interesseret i at målingen skulle influeres af spam i konsollen, da der ville blive printet resultater flere millioner gange.

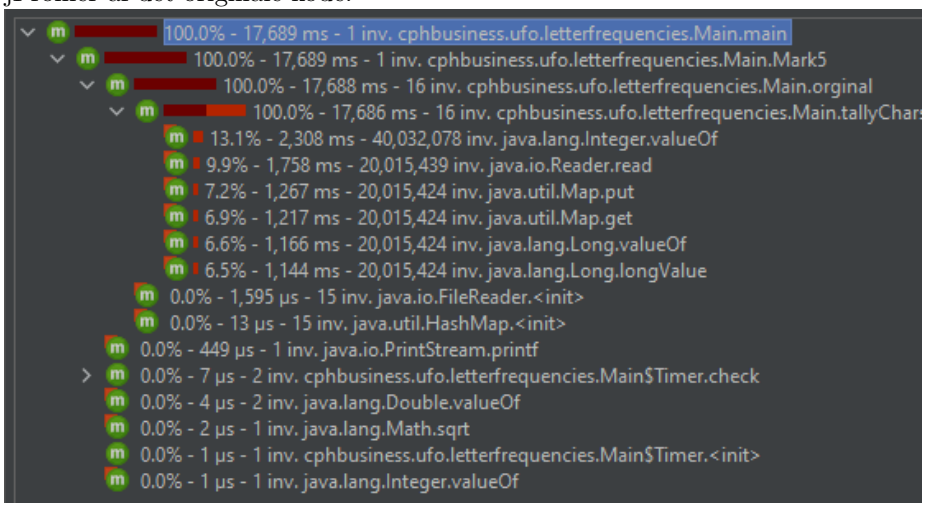
2 Original program

2.1 Koden

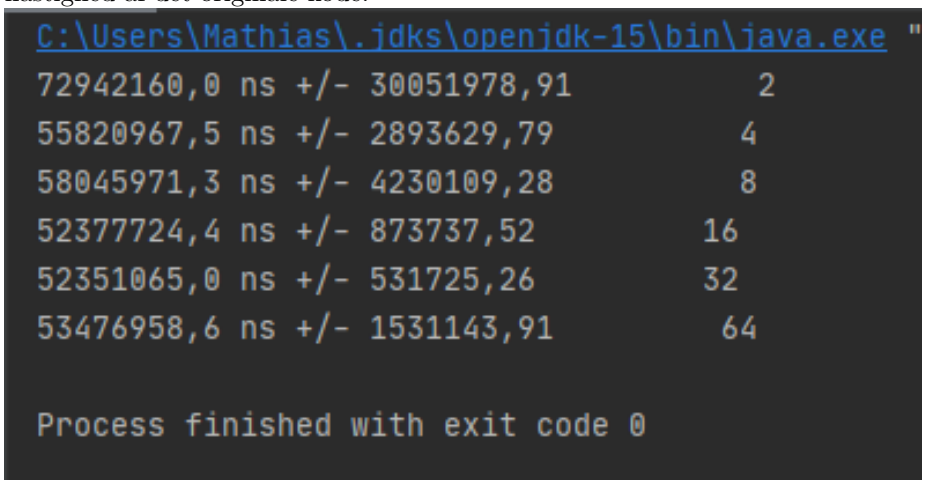
```
private static void orginal() throws IOException {
       String fileName = "filepath";
       Reader reader = new FileReader(fileName);
       HashMap<Integer, Long> freq = new HashMap<>();
       tallyChars(reader, freq);
   }
private static void tallyChars(Reader reader, Map<Integer, Long> freq)
    throws IOException {
   int b;
   while ((b = reader.read()) != -1) {
       try {
           freq.put(b, freq.get(b) + 1);
       } catch (NullPointerException np) {
           freq.put(b, 1L);
       }
   }
}
```

2.2 Dokumentation af performance

jProfiler af det originale kode.



hastighed af det originale kode.



2.3 Hypotese over hvor problemet opstår

Efter første gennemgang fandt vil følgende potentielle bottlenecks og muligheder for forbedringer.

1. Filereader læser en char af gangen i while-loopet.

   Vi regner med at dette kan gøres hurtigere. Enten ved at hente en linje ud af gangen, eller hele filen, hvis der er tale om en mindre fil.

2. get/put metoderne i freq map.

get og put metoderne til HashMappet freq er dyre. Vi regner med at dette kan gøres bedre.

3. Unødvendig try/catch af nullpointer.

   Vi så det unødvendigt at bruge en try/catch, da metoden getOrDefault() kan bruges i stedet for get(). Denne ændring vil dog ikke gøre lige så stor forskel som resten.

3 Optimeret løsning

3.1 Performance

Iteration af hver enkel char, er blevet byttet ud med en streamoperation på `Files.lines()`. Her bliver stream-operationen `flatMapToInt(String::chars)` anvendt til at konvertere vores liste af enkle linjer til en stream af ints (int-value af chars) som vi kan arbejde med. Konverting af char til ASCII var en stor byrde på tidsforbruget i den orginale kode, men med den nye stream som indeholder vores allerede konverterede ints, har vi muligheden for at samle den til vores map. For at få vores samlede resultate ud, bruger vi boxed og collect. Her samler vi vores ASCII numre samt antalet af instanser af dem ved hjælp af:

```
        .collect(Collectors.groupingBy(Function.identity(),
            Collectors.counting()))
```

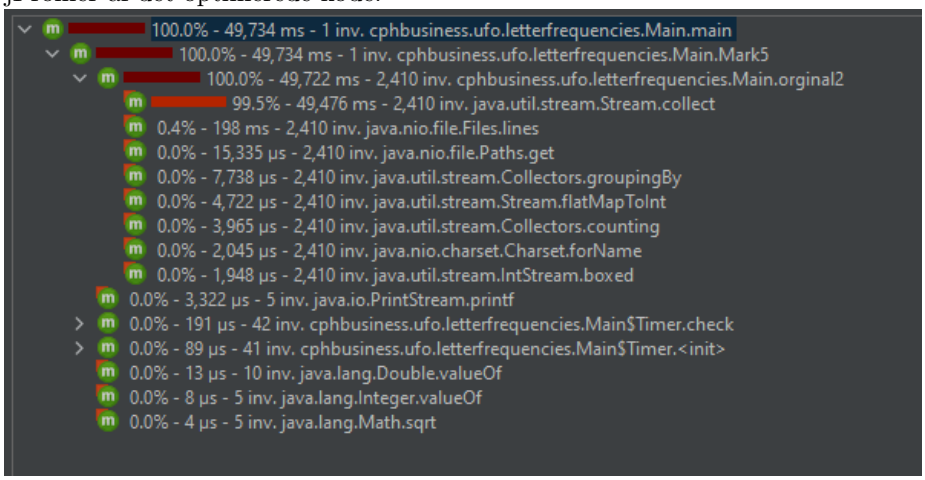
For at optimere yderligere, har vi valgt at køre vores streams parallelt.

3.2 Kode

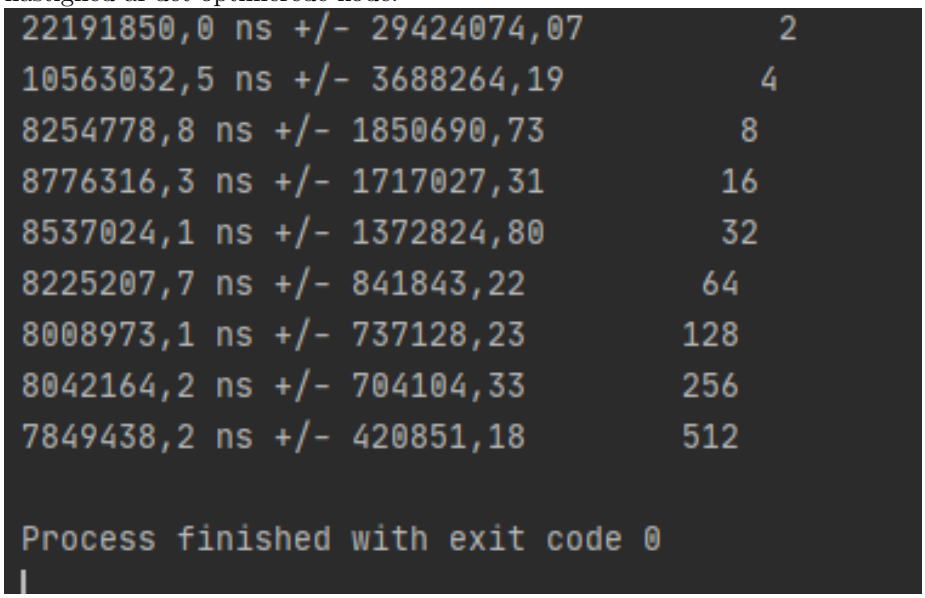
```
private static void orginal2() throws IOException {
       String fileName = "Path/FoundationSeries.txt";

       Path filepath = Paths.get(fileName);

       Map<Integer, Long> freq =
              Files.lines(filepath, Charset.forName("UTF-8")).parallel()
                     .flatMapToInt(String::chars)
                     .boxed()
                     .collect(Collectors.groupingBy(Function.identity(),
                         Collectors.counting()));


   }
```

3.3 Dokumentation af performance

jProfiler af det optimerede kode.



hastighed af det optimerede kode.



4 Resultater

Efter optimering af koden, er metoden nu 7.27 gange hurtigere end den orginale, svarende til en reduktion på 86% Resultatet vil formentlig variere meget baseret på hardware, vi har kun kørt tests på 1 enkel maskine. Nedenfor ses et boxplot af vores resultater.

Vi har ikke umiddelbart kunne optimere algoritmen yderligere. Et forslag til at finde den hurtigste måde er at bruge Cunninghams Law: "the best way to get the right answer on the internet is not to ask a question; it's to post the wrong answer." Altså kunne man lave et post på Stackoverflow og sige man har lavet den hurtigst mulige algoritme, så skal der nok være nogen der poster en mere effektiv version hvis det findes :-) :-) :-) :-)