Tomer Zwi
uni tzz27
Professor Edwards
December 18th 2019

Auto Suggest

Introduction:

In this project, I created a simple auto suggest using bigrams, the probability of two words occurring together.
The user provides a word and the program returns the top five word suggestions based on the yelp reviews data set.
The auto suggest task is used every day by millions of users. I'm trying to show that running auto suggest in parallel shortens the time it takes to complete the task.

Code:

Things that I tried previously but does not work:

Sqlite: originally, I planned to use Sqlite in Haskell, however, implanting it can take a lot time so I decided to change and use a map. I am storing the data as map (key and value) where the key is the word (first word) and the value is (second word in bigram and Int which represent the number of words it

By reading the results of threadscope I discovered that reading line by line is slower than reading the whole text.
I discovered that for my particular hard drive, I could not read files in parallel.
(I build the list of bigram asynchronously (in parallel)
https://www.quora.com/Can-an-SSD-solid-state-drive-perform-read-and-write-operations-simultaneously

Code I used:

I have two programs - one is for running in parallel and one is for running not in parallel (synchronously). Both problems are similar but different in the main function. I separated the code to 3 parts:

Pure functions:

non_digit_in_word:

```
-- take a string and return a string in a lower case after removing any punctuation.

'''
non_digit_in_word :: String -> String
non_digit_in_word [] = []
non_digit_in_word (x:xs)
        | isAlpha x = toLower x : non_digit_in_word xs
        | otherwise = non_digit_in_word xs

''''
clean_words:
-- get a list of strings and return a list of strings without punctuation/digits
'''
clean_words:: [String] -> [String]
clean_words xs =  fmap non_digit_in_word xs
'''
```

makeBigrams:

```
{-
It takes a list of words that returns a list of bigrams which is a tuple of two words .
 for example for ["hello","world","chocolate"] it will return
 [("hello", "wordl"),("world","chocolate")])

-}


'''

makeBigrams :: [String] -> [(String,String)]
makeBigrams [] = []
makeBigrams xs =
  snd (Prelude.foldr (\x (a,b) ->
            case a of
              Nothing -> (Just x,b)
              Just y  -> (Just x,(x,y):b)
          ) (Nothing,[]) xs)

'''
```

mkDatabase2:

```
{-
take a list of tuples of two variable and store them as key value where the key is a word
(that the user will input later) and the value is a word suggestion and an int representing the
number
of times the bigram exists in the corpus.

-}


'''
```

```
mkDatabase2 :: [(String,String)] -> Map String [(String, Int)]
mkDatabase2 m =
  let p = fromListWith (\x y ->(x++y)) $ fmap (\(a,b) -> (a,[b])) m
  in fmap (\m-> toList $ fromListWith (+) $ fmap (\x -> (x,1)) m ) p

‘’’
getTopFiveResults:

{-
Get a Maybe list of words and return the top 5 words that have the most occurrences
(if there aree less than 5 words it will return “” on the list in ascending order).
-}

‘’’


getTopFiveResults :: Maybe [(String, Int)] -> [String]
getTopFiveResults Nothing = []
getTopFiveResults (Just lis) = fmap (fst) $ (Prelude.take 5 $ Data.List.sortBy (\(_,x)
(_,y)-> compare x y) lis )

‘’’
```

IO function:

```
getUserWord
--getting I0 string from the user via the command line and return it in a lower case


‘’’
getUserWord :: IO String
getUserWord = do
  putStrLn "Please enter an input"
  line <- getLine
  return $ fmap toLower line
```

‘’’

Main:

```
main :: IO ()
main = do
```

{-
First it creates a list of all the files I want to use.
Then it reads them synchronously using mapM readFile.
-}

```
  z <- mapM readFile $ fmap (\x-> "revi" ++  show x ) [0..99]
```

{-
z is a list of strings which are the contents of each file.
split the string into lines in parallel,
then split each line into words and run clean words on each word,
then concatenate them all together.
-}

```
let x = concat $ parMap rpar (parMap rpar (clean_words . words) . lines) z
```

-- It takes list of lists of strings and turns it into a list to bigrams in parallel.

```
  let y = concat $ parMap rpar makeBigrams x
```

In synchronously I used fmap instead parMap and rpar as below:

```
  let x = concat $ fmap  ( fmap  (clean_words . words) . lines) z
  let y = concat $ fmap makeBigrams x
```

{-
Below I make the database from the bigrams and then get a word from the user

and return the top 5 suggested words.
In case the user input is empty return an error message and in case the user input contains more than one word take the first one.
-}

```
  let database = mkDatabase2 y
  getUserWordWhileNotQuit database
  where
    getUserWordWhileNotQuit database  = do
      word <- getUserWord
      let z = words word
      case z of
        [] -> putStrLn "Erorr - empty input"
        _ -> do
          putStrLn "Here are the suggestion words (if found)"
          putStrLn $ show $ getTopFiveResults $ Data.Map.lookup (head z) database
‘’’
```

Data I used for the tests:

Each file contains 1000 Yelp reviews and weighs 1.2 MB on average. Obtaining from (https://www.yelp.com/dataset)

Instruction to Run:

Before compile:
Should have two programs (one parallel and one for non parallel), threadscope, and 100 files. To choose the amount of files you want to run please change the line below:

```
  z <- mapM readFile $ fmap (\x-> "revi" ++  show x ) [0..99]
```

first compile:
stack ghc No_Parallel -- -O2 -rtsopts -eventlog -threaded
stack ghc Project_compelte -- -O2 -rtsopts -eventlog -threaded

Then for running:
./code +RTS (-NX) - l -s

-NX if you want to run on parallel.

to see the eventlog - ./threadscope code.eventlog

Tests

For all the tests I use the same input: chocolate
I used the following tests:
2 files
10 files
20 files
50 files
100 files

Two Files test

Non parallel

Return: ["","addiction","all","banana","bar"]

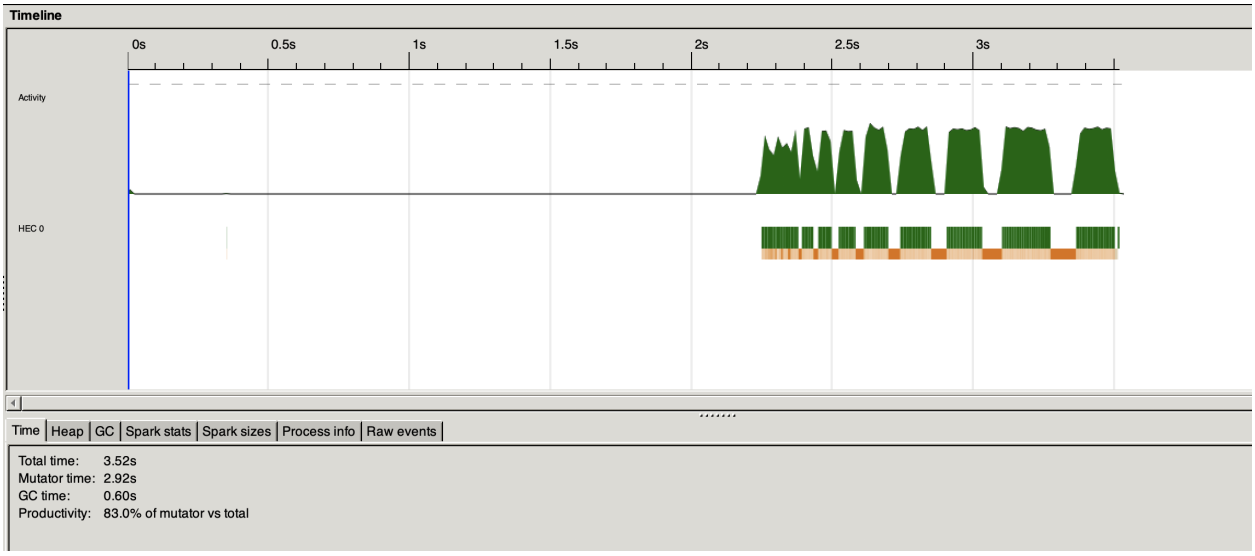
55 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 4 (1 bound, 3 peak workers (3 total), using -N1)

SPARKS: 0(0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

Total time 1.207s (3.518s elapsed)

eventlog:



Parallel 2 cores:

["","addiction","all","banana","bar"]

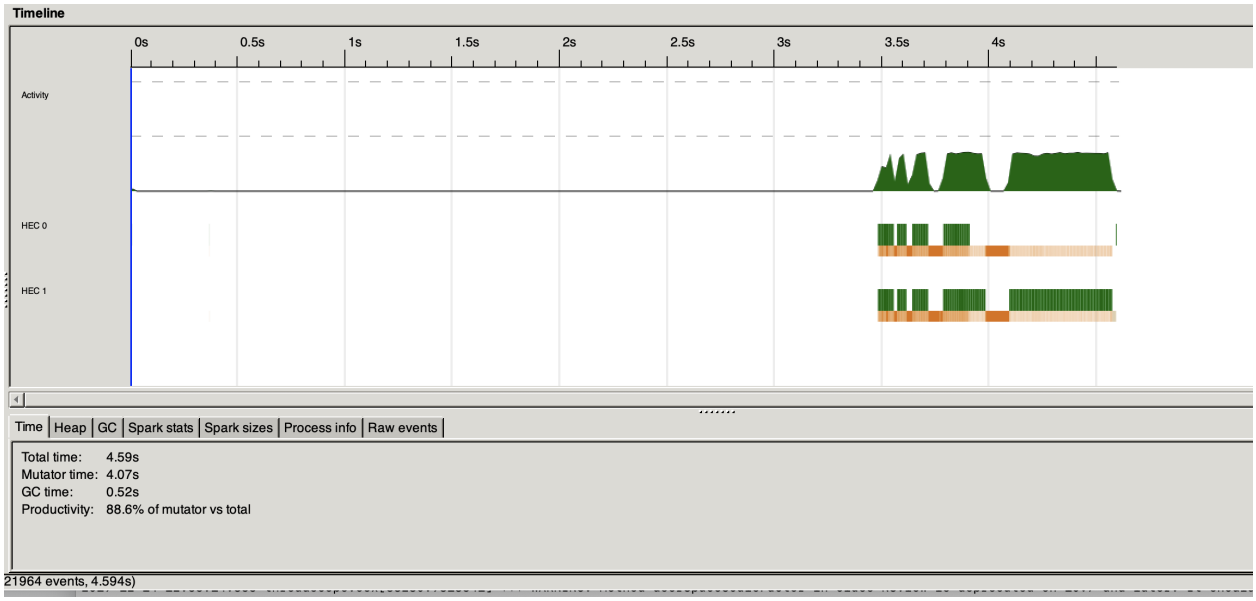
111 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 6 (1 bound, 5 peak workers (5 total), using -N2)

SPARKS: 34532(33376 converted, 577 overflowed, 0 dud, 560 GC'd, 19 fizzled)

Total time 1.639s (4.594s elapsed)

EventLog:



Parallel 4 cores:

["","addiction","all","banana","bar"]

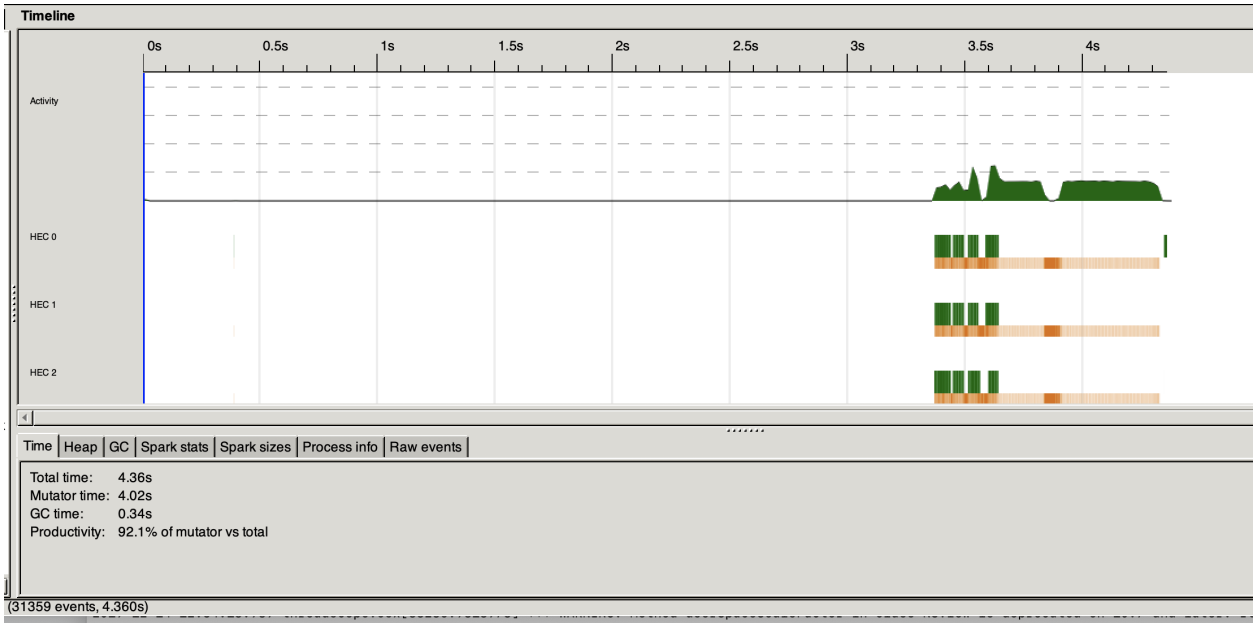
105 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 10 (1 bound, 9 peak workers (9 total), using -N4)

SPARKS: 34532(33924 converted, 303 overflowed, 0 dud, 297 GC'd, 8 fizzled)

Total time 2.146s (4.360s elapsed)

Eventlog:



Parallel 6 cores:

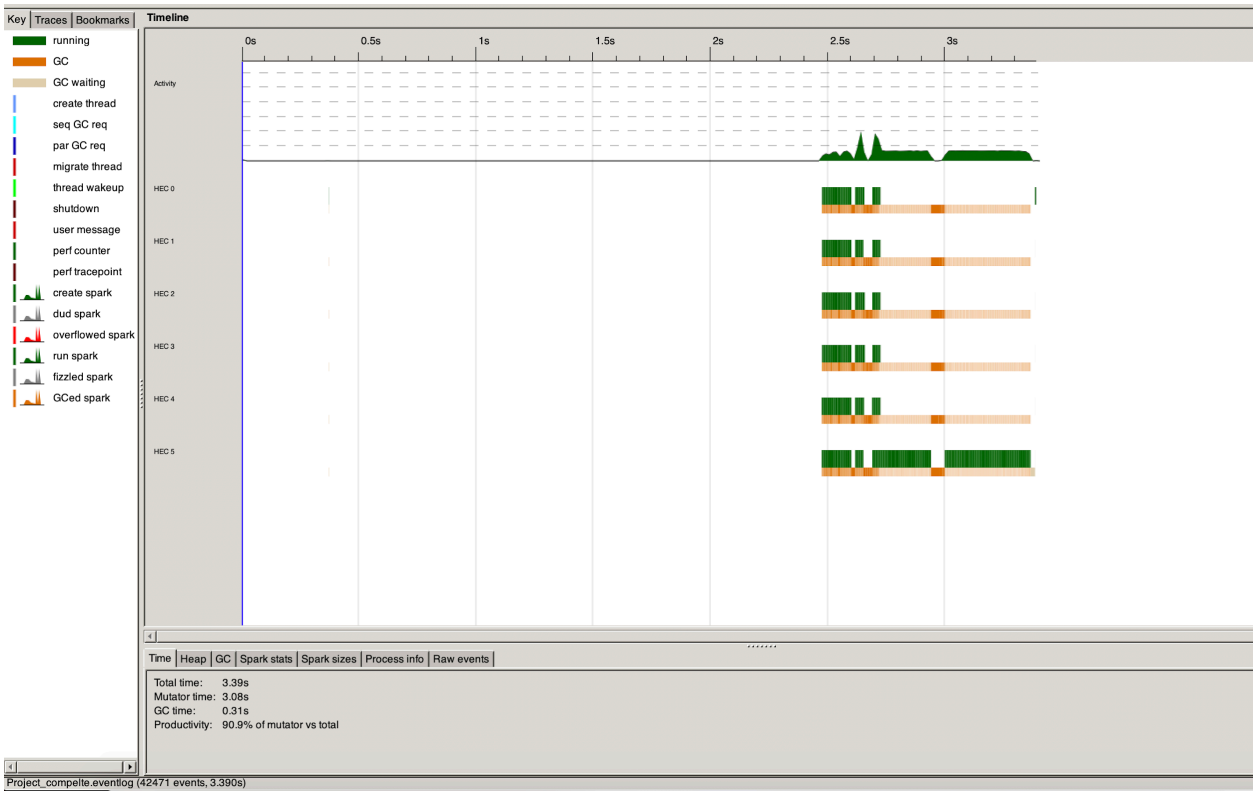
["","addiction","all","banana","bar"]
02 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 14 (1 bound, 13 peak workers (13 total), using -N6)

SPARKS: 34532(33690 converted, 420 overflowed, 0 dud, 401 GC'd, 21 fizzled)

Total time 2.634s (3.390s elapsed)

Eventlog



Parallel 8 cores:

["","addiction","all","banana","bar"]

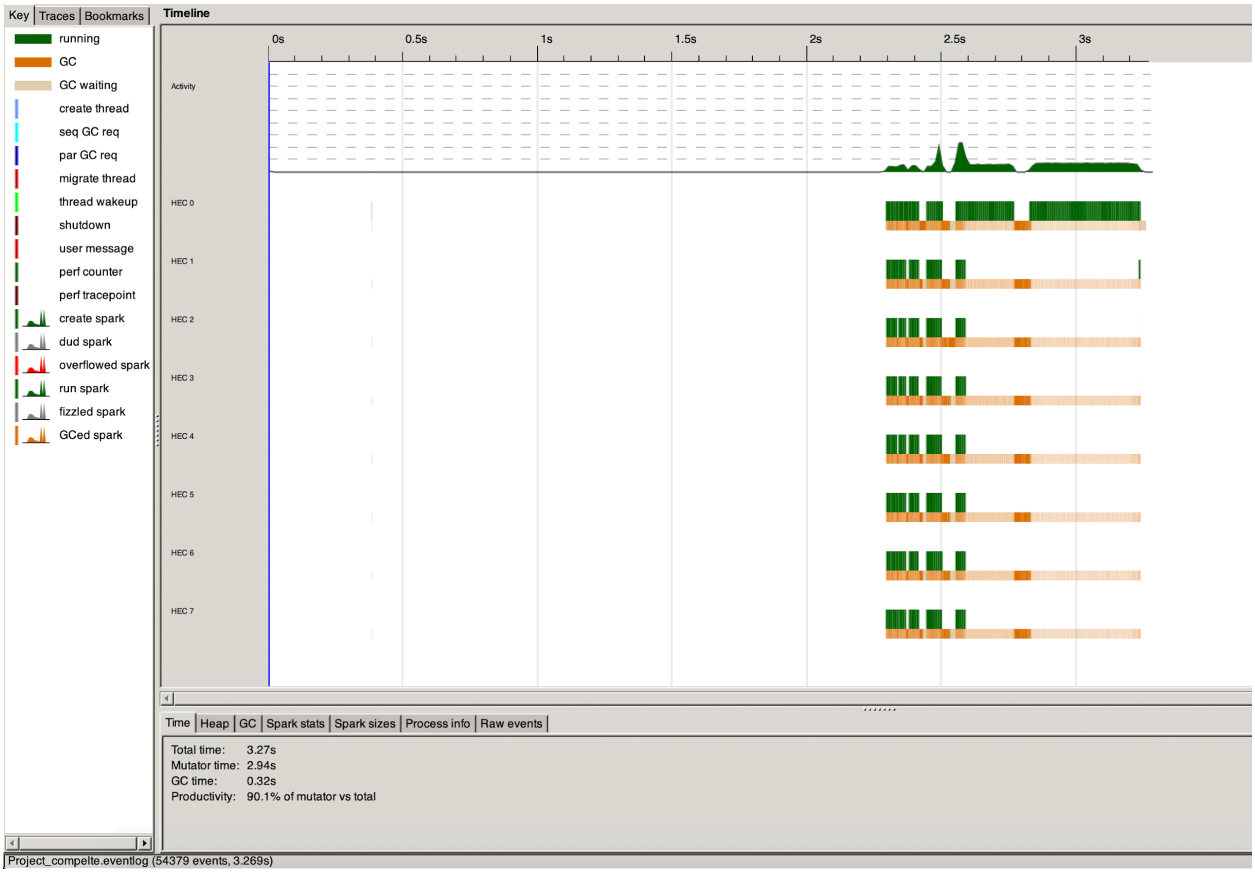
109 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 18 (1 bound, 17 peak workers (17 total), using -N8)

SPARKS: 34532(33852 converted, 339 overflowed, 0 dud, 331 GC'd, 10 fizzled)

Total time 3.513s (3.269s elapsed)

Eventlog



Observation for 2 files:

The best result is the 8 cores with 3.29 seconds, but it is the only tasks that non parallel run faster than parallel with 2 cores and 4 cores.
Also the improvement is very small (maybe it is because the task is very simple and small)

10 Files Test

Non parallel:

["affaire","after","ale","all","amaretto"]

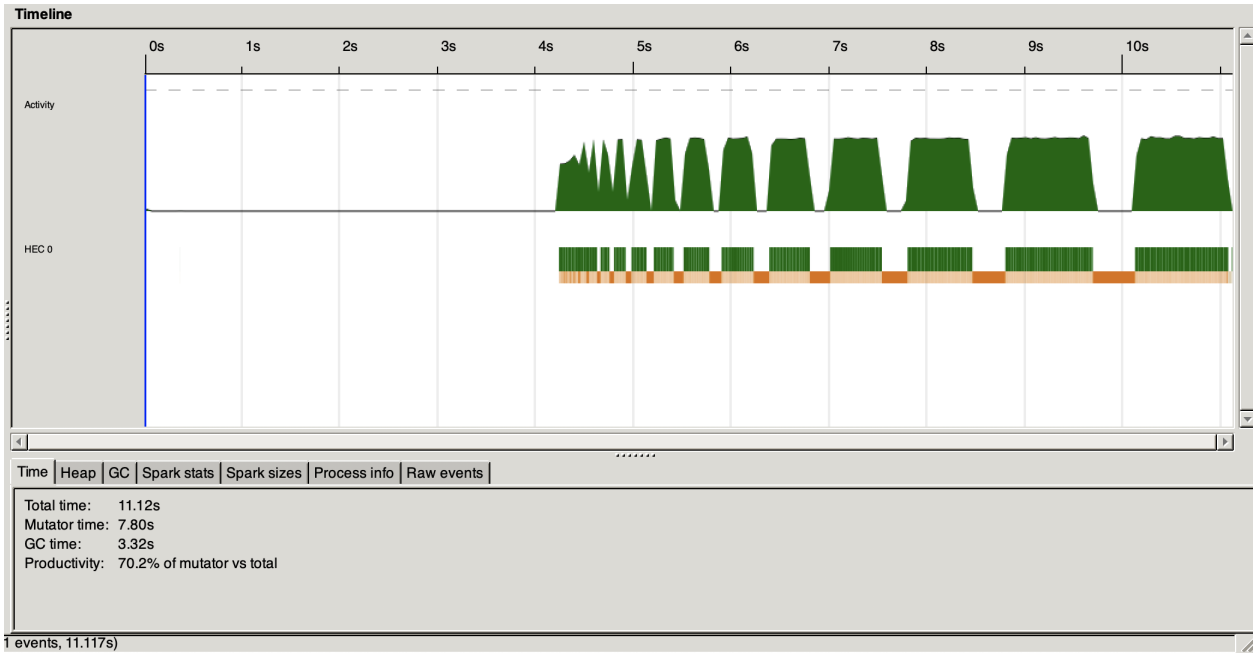
260 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 4 (1 bound, 3 peak workers (3 total), using -N1)

SPARKS: 0(0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

Total time 6.607s (11.117s elapsed)

Eventlog:



Parallel 2 cores:

["affaire","after","ale","all","amaretto"]

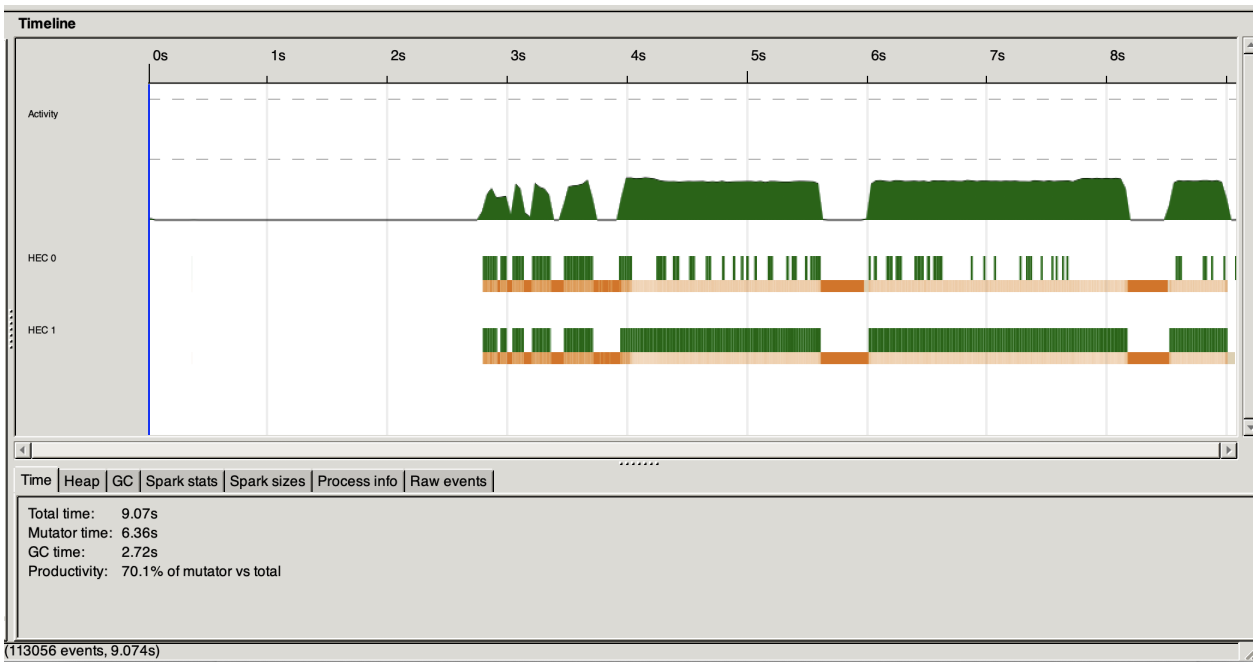
328 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 6 (1 bound, 5 peak workers (5 total), using -N2)

SPARKS: 169544(67565 converted, 50991 overflowed, 0 dud, 49190 GC'd, 1798 fizzled)

Total time 8.961s (9.074s elapsed)

Eventlog:



Parallel 4 cores:

["affaire","after","ale","all","amaretto"]

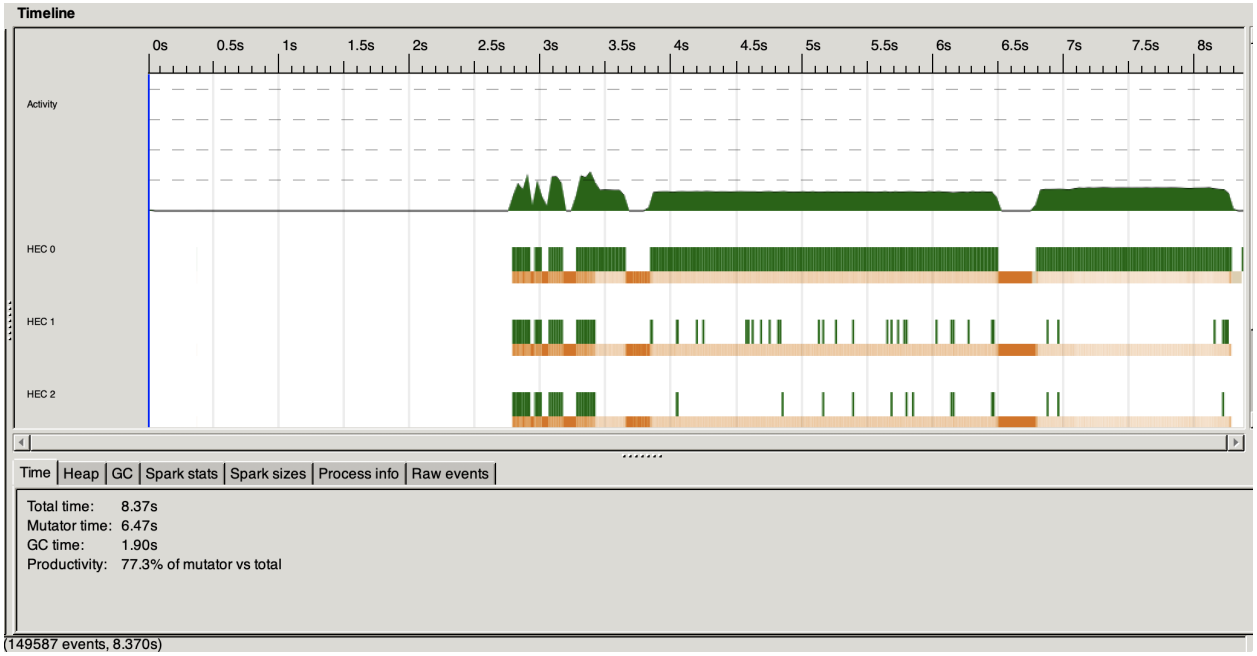
410 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 10 (1 bound, 9 peak workers (9 total), using -N4)

SPARKS: 169544(108192 converted, 30677 overflowed, 0 dud, 29447 GC'd, 1228 fizzled)

Total time 11.501s (8.369s elapsed)

Eventlog:



Parallel 6 cores:

["affaire","after","ale","all","amaretto"]

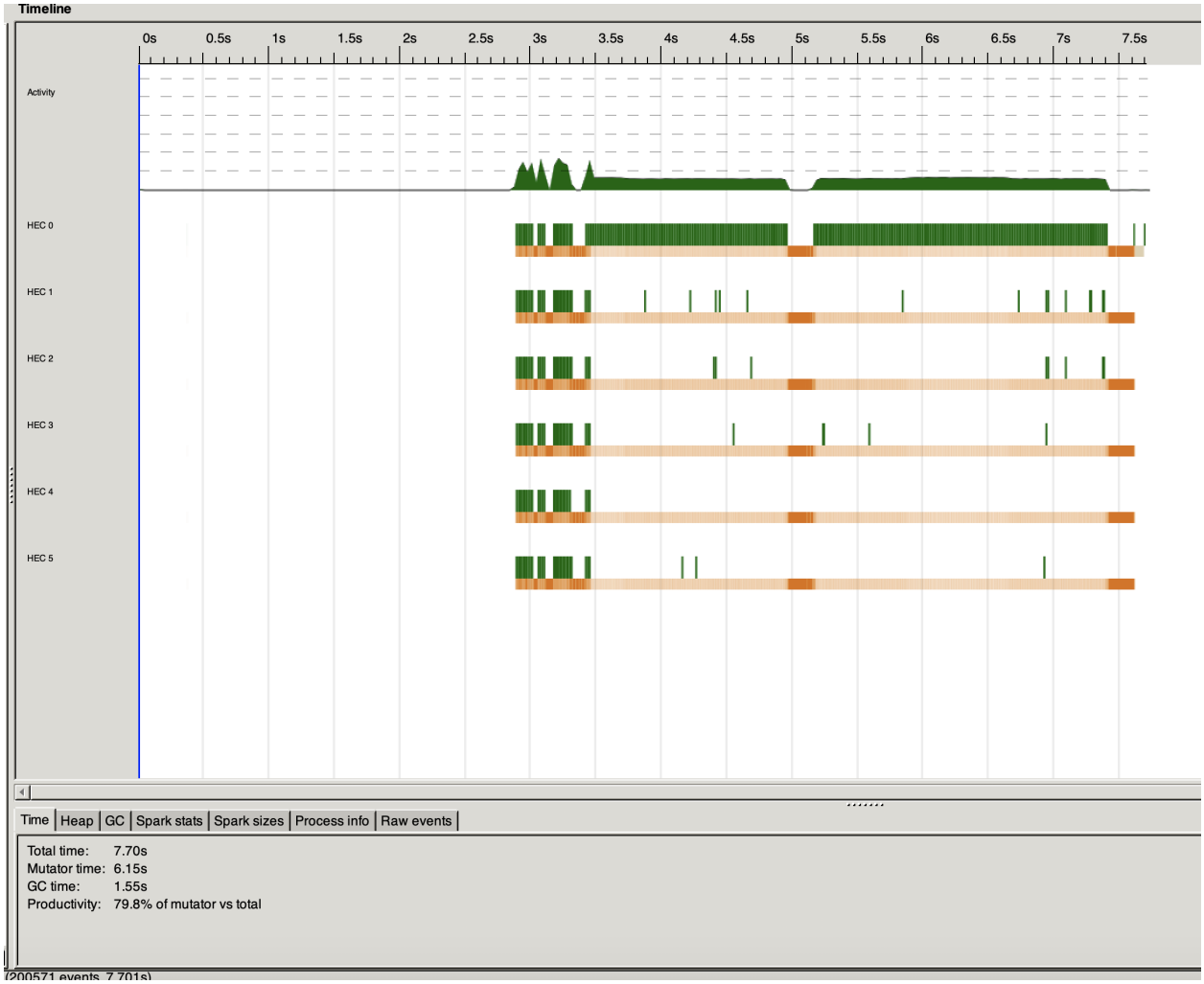
377 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 14 (1 bound, 13 peak workers (13 total), using -N6)

SPARKS: 169544(110485 converted, 29531 overflowed, 0 dud, 28370 GC'd, 1158 fizzled)

Total time 14.004s (7.701s elapsed)

Eventlog:



Parallel 8 cores:

["affaire","after","ale","all","amaretto"]

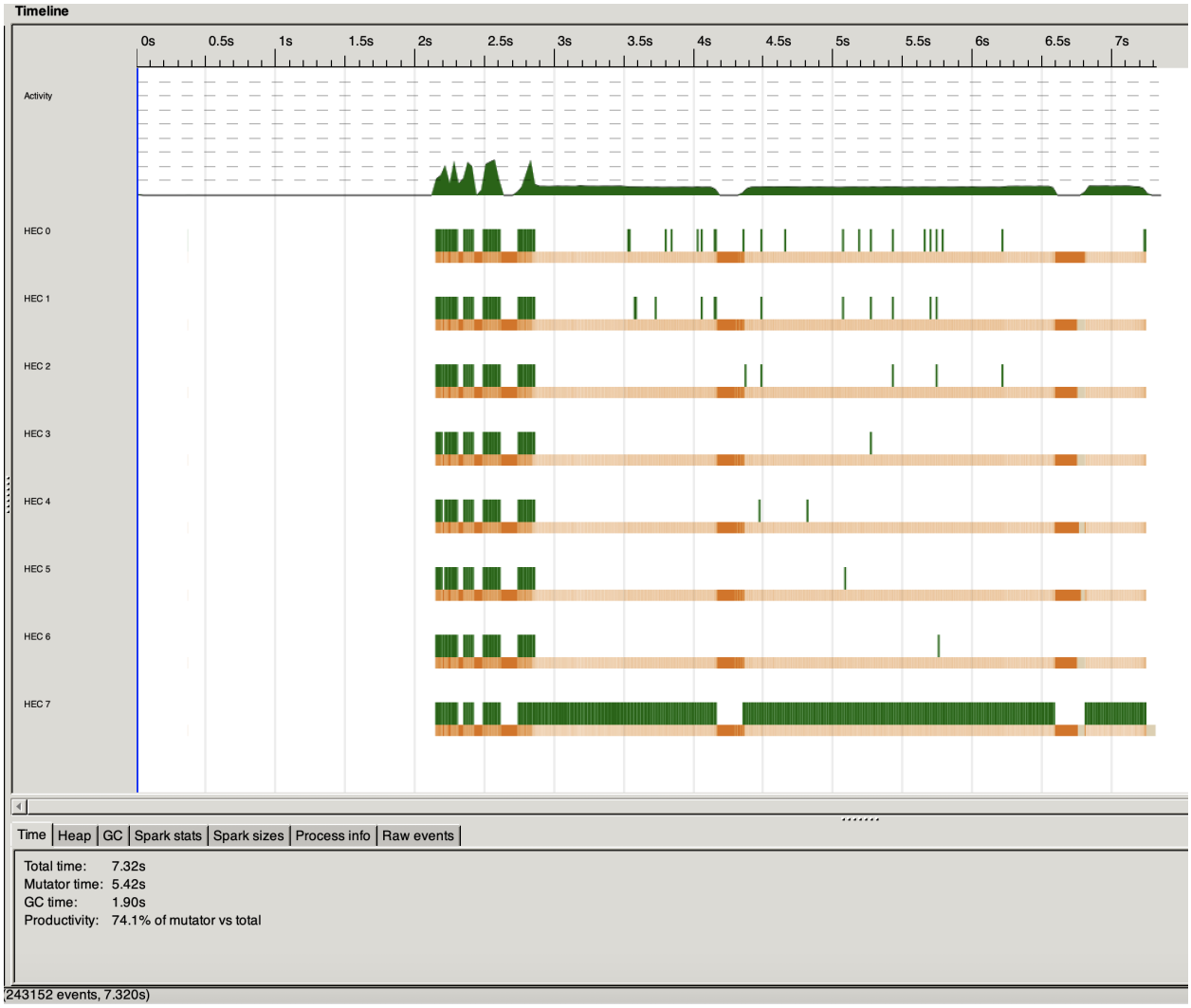
364 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 18 (1 bound, 17 peak workers (17 total), using -N8)

SPARKS: 169544(118807 converted, 25371 overflowed, 0 dud, 24246 GC'd, 1120 fizzled)

Total time 19.476s (7.320s elapsed)

Eventlog:



Observation for 10 files test:

The best test is 8 cores with 7.3 seconds as I increase the cores the time it takes to complete the task is shorter and the number of succesfully converted sparks is higher. All the results are better compared to non-parallel.

20 Files Test

Non parallel:

["affaire","after","ale","amaretto","andor"]

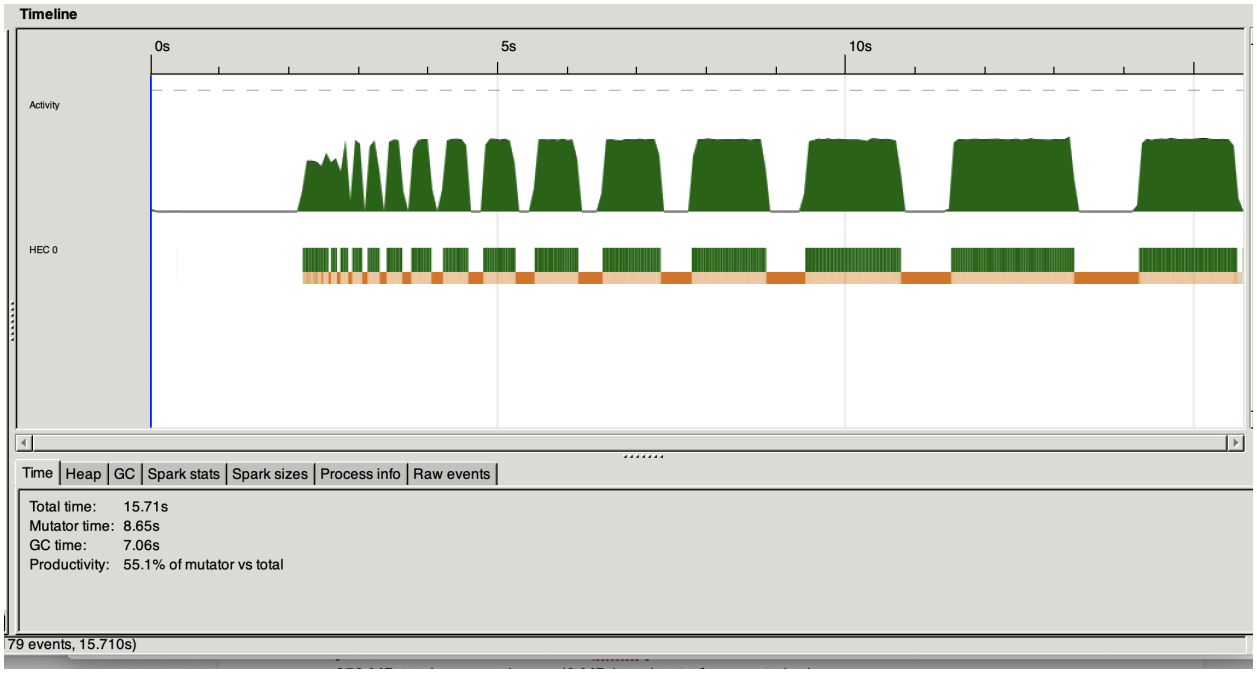
553 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 4 (1 bound, 3 peak workers (3 total), using -N1)

SPARKS: 0(0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

Total time 13.030s (15.710s elapsed)

Eventlog:



Parallel 2 cores:

["affaire","after","ale","amaretto","andor"]

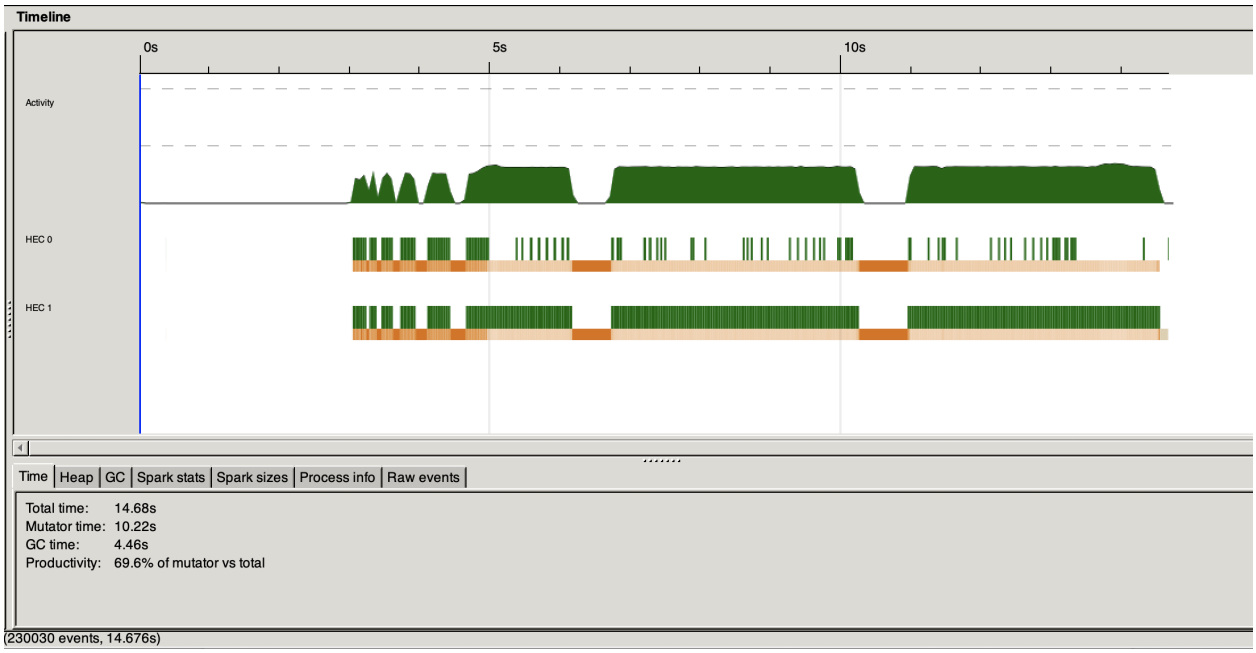
646 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 6 (1 bound, 5 peak workers (5 total), using -N2)

SPARKS: 341804(112132 converted, 114840 overflowed, 0 dud, 111436 GC'd, 3396 fizzled)

Total time 16.456s (14.676s elapsed)

Evenlog:

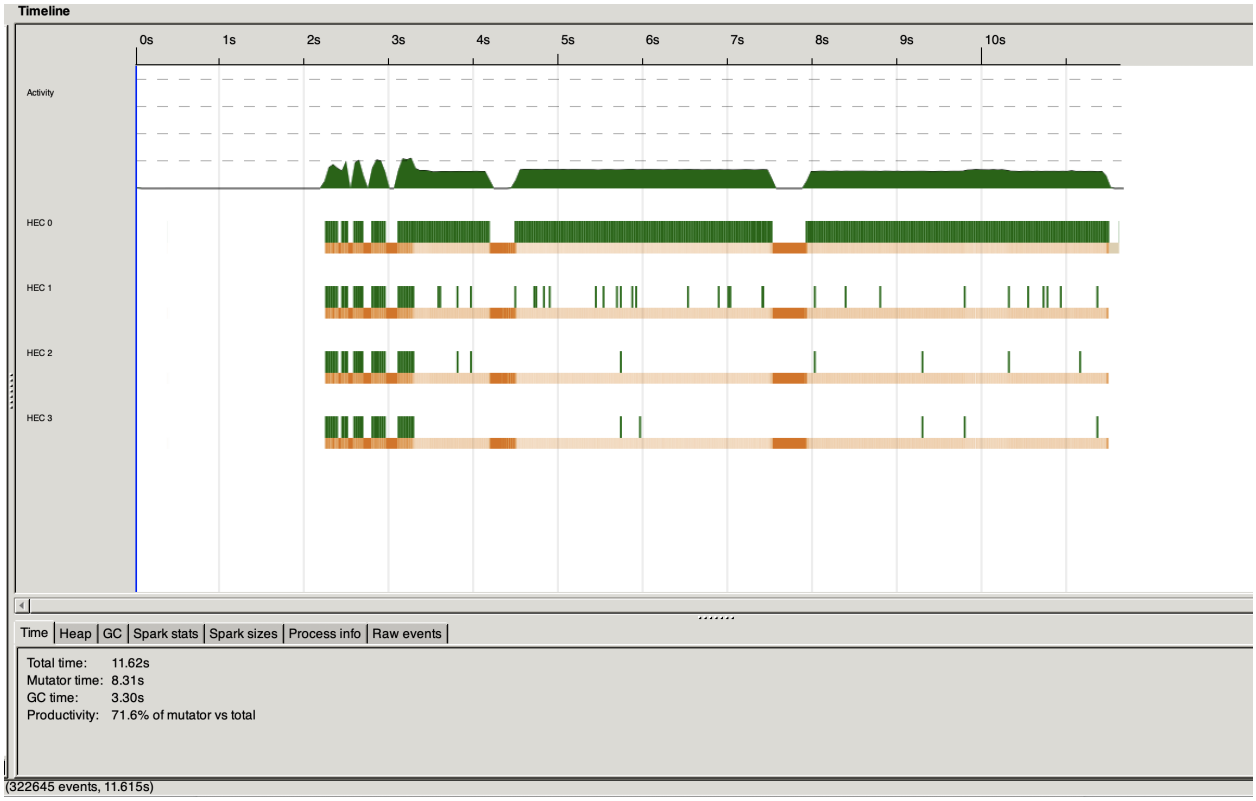


Parallel 4 cores::

["affaire","after","ale","amaretto","andor"]
658 MB total memory in use (0 MB lost due to fragmentation)
TASKS: 10 (1 bound, 9 peak workers (9 total), using -N4)
SPARKS: 341804(157822 converted, 91994 overflowed, 0 dud, 88754 GC'd, 3234 fizzled)

Total time 19.615s (11.615s elapsed)

Eventlog:



Parallel 6 cores:

["affaire","after","ale","amaretto","andor"]

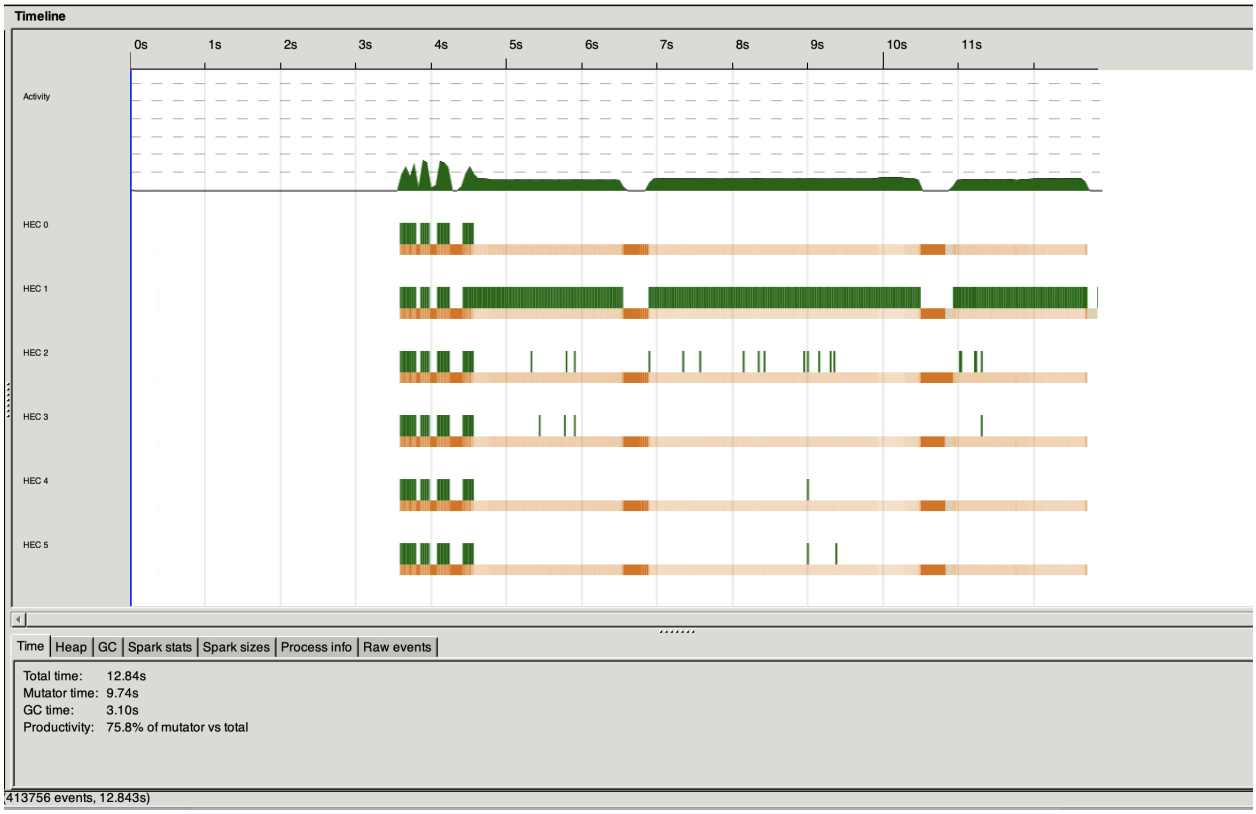
705 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 14 (1 bound, 13 peak workers (13 total), using -N6)

SPARKS: 341804(188792 converted, 76509 overflowed, 0 dud, 73883 GC'd, 2620 fizzled)

Total time 25.851s (12.843s elapsed)

Eventlog:



Parallel 8 cores:

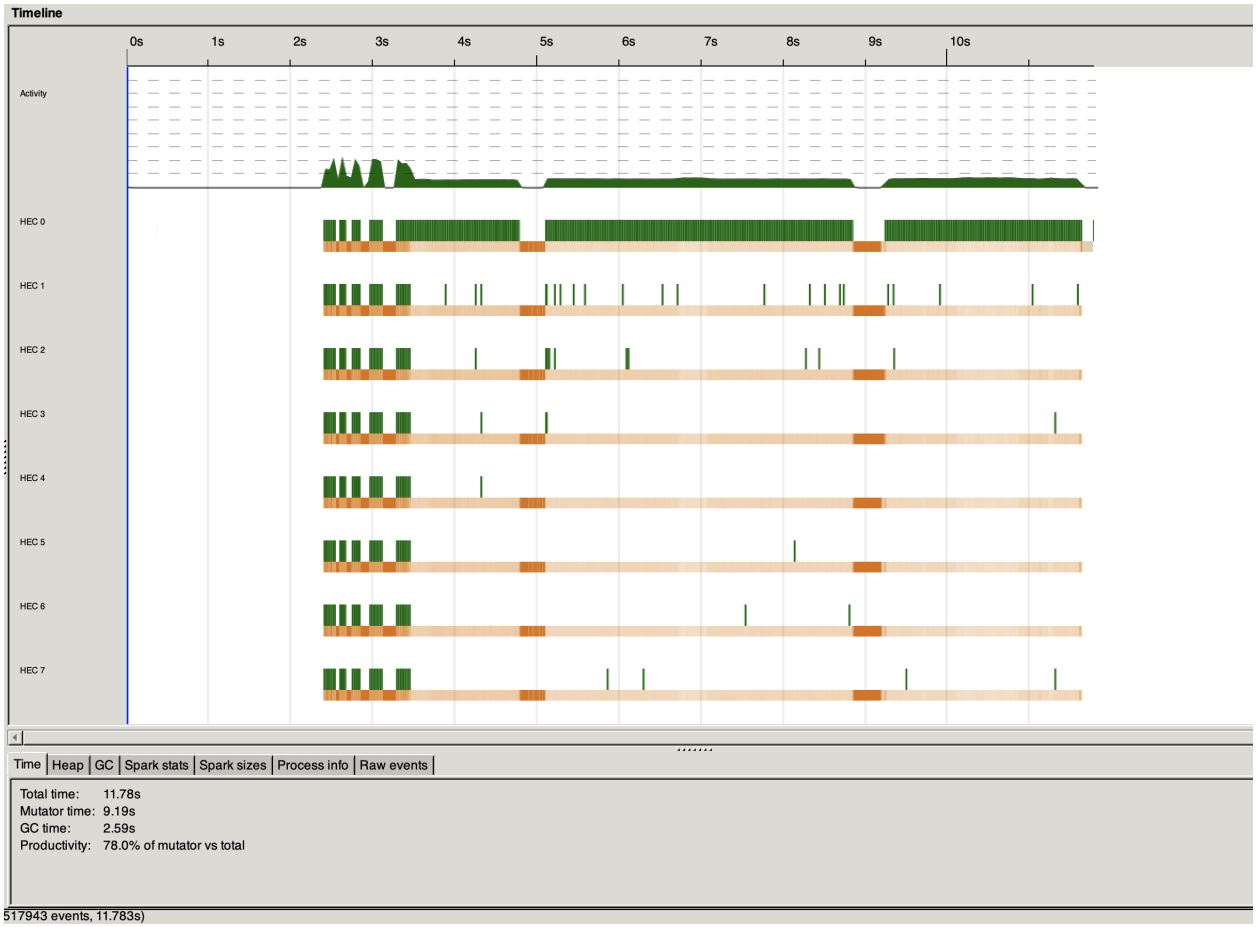
["affaire","after","ale","amaretto","andor"]
709 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 18 (1 bound, 17 peak workers (17 total), using -N8)

SPARKS: 341804(199831 converted, 70990 overflowed, 0 dud, 68063 GC'd, 2920 fizzled)

Total time 32.669s (11.783s elapsed)

Eventlog:



Observation for 20 files test:

All the parallel results were better than non-parallel.
The best result in terms of speed is 4 cores with 11.6 second. When running with 6 cores I got a slower time result so I thought that 8 cores would be even slower but it was not the case. 8 cores runs faster than 6 cores.

50 Files Test

No parallel:

["abuelita","accesses","added","addict","affaire"]

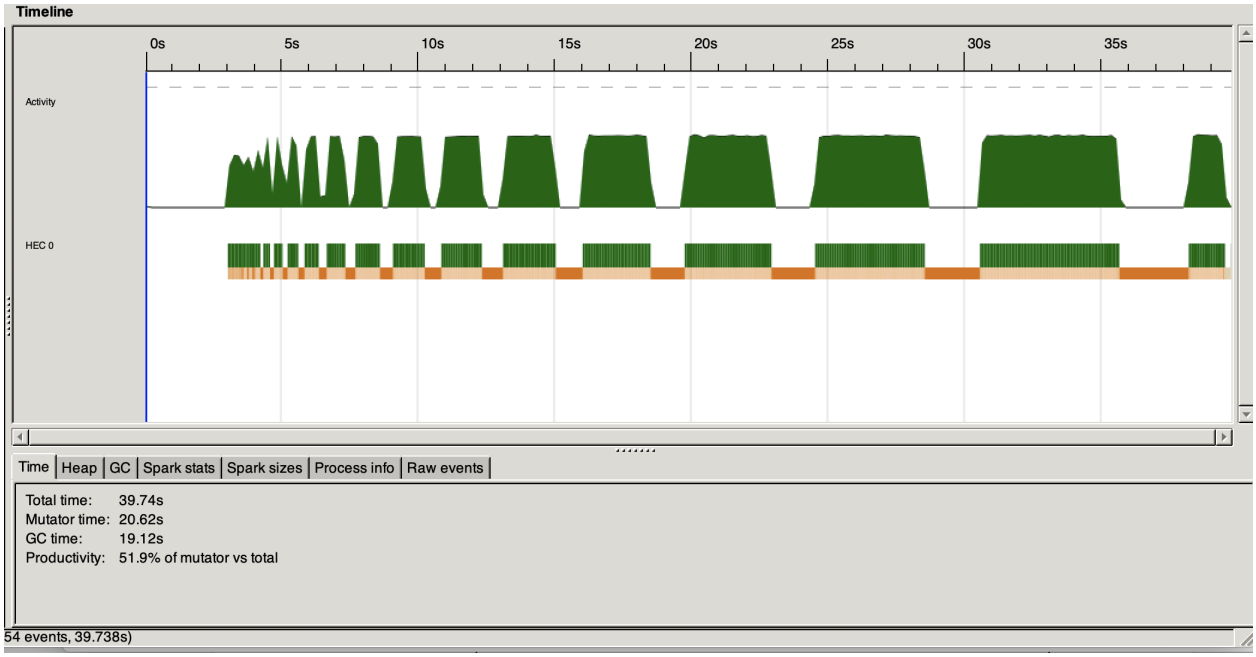
1537 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 4 (1 bound, 3 peak workers (3 total), using -N1)

SPARKS: 0(0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

Total time 35.314s (39.738s elapsed)

image:



Parallel 2 cores:

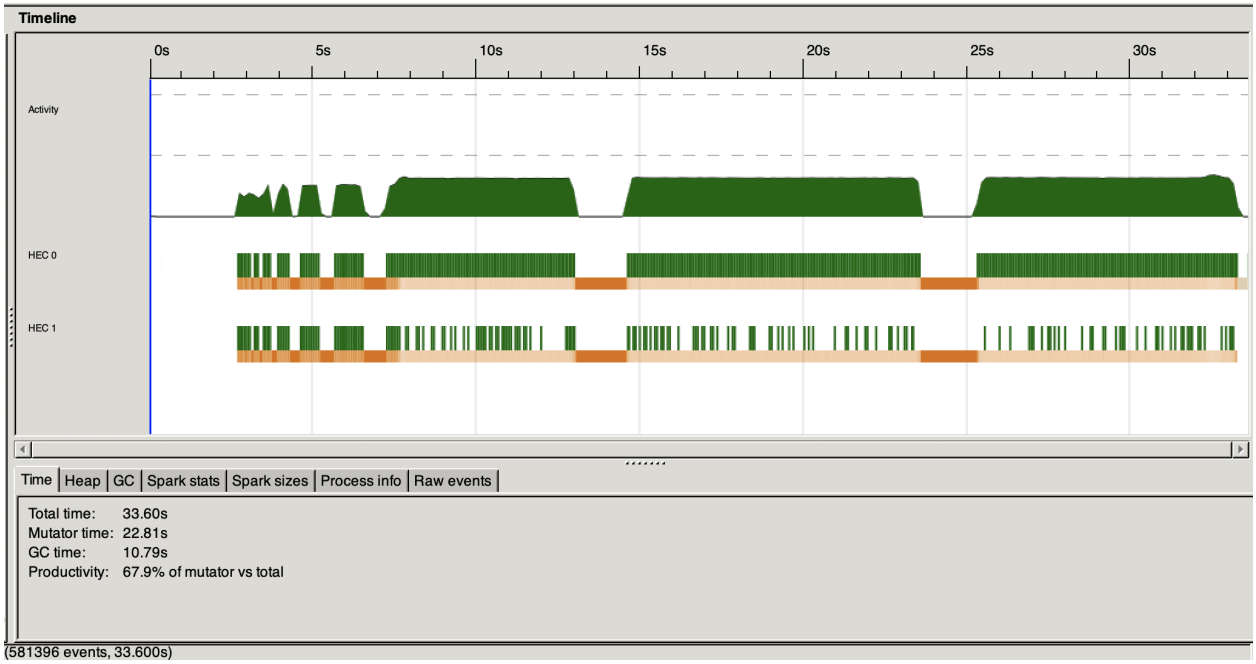
["abuelita","accesses","added","addict","affaire"]

1573 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 6 (1 bound, 5 peak workers (5 total), using -N2)

SPARKS: 861552(241862 converted, 309857 overflowed, 0 dud, 300416 GC'd, 9417 fizzled)
Total time 43.658s (33.599s elapsed)

Eventlog:



Parallel 4 cores:

["abuelita","accesses","added","addict","affaire"]

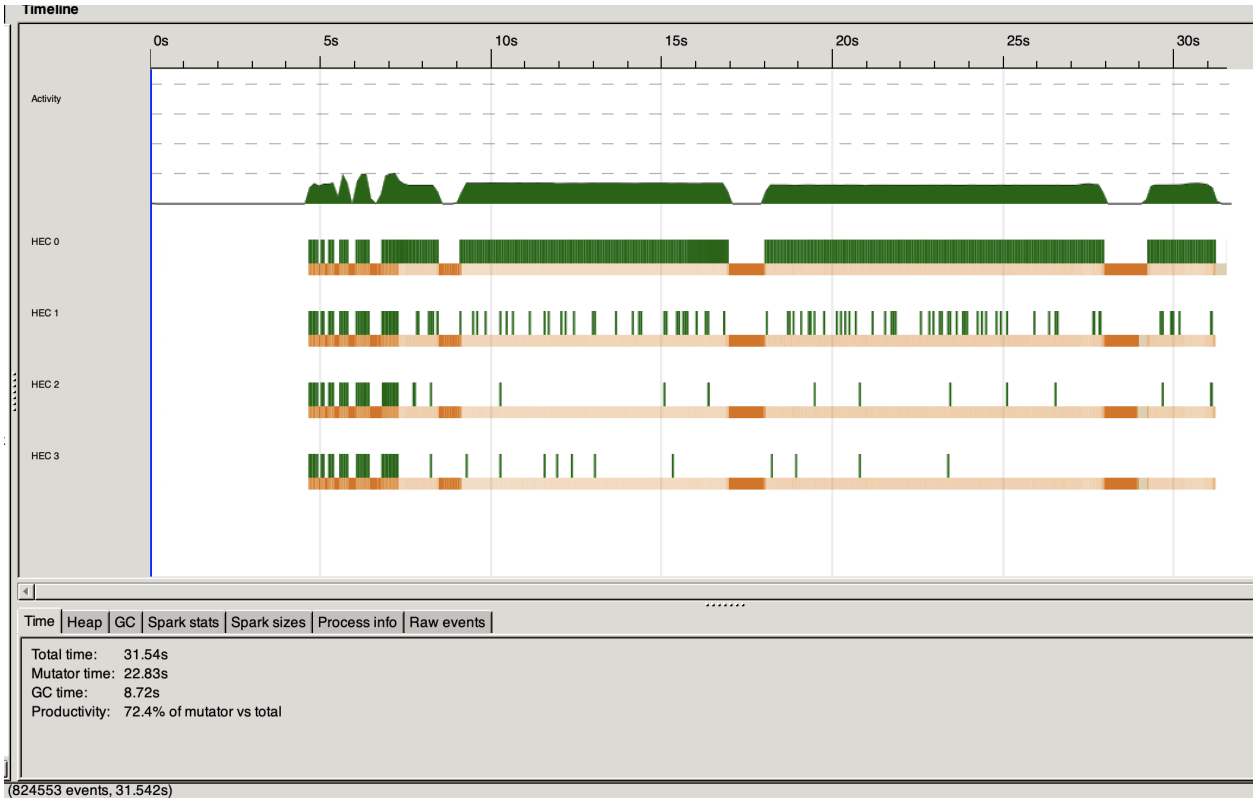
1654 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 10 (1 bound, 9 peak workers (9 total), using -N4)

SPARKS: 861552(365731 converted, 247918 overflowed, 0 dud, 240087 GC'd, 7816 fizzled)

Total time 58.005s (31.542s elapsed)

Eventlog:



Parallel 6 cores:

["abuelita","accesses","added","addict","affaire"]

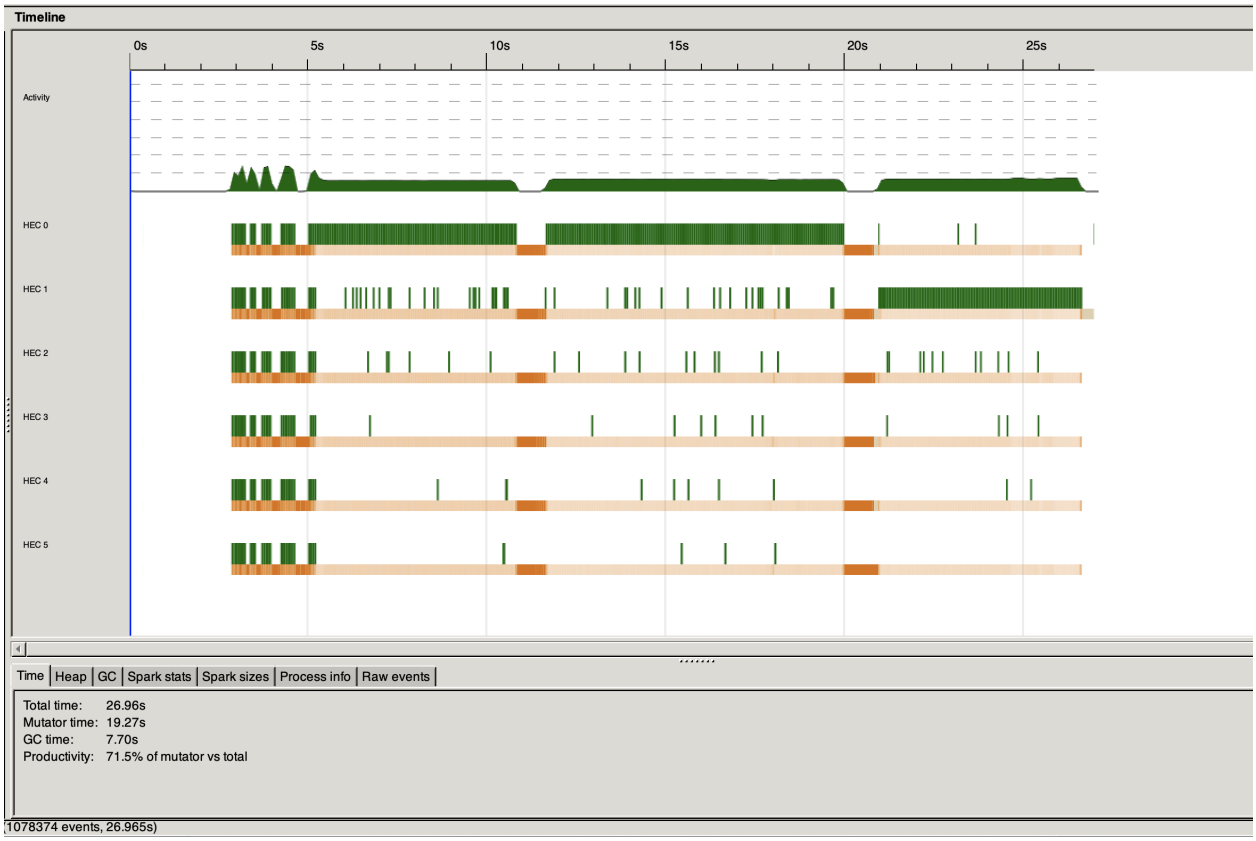
1687 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 14 (1 bound, 13 peak workers (13 total), using -N6)

SPARKS: 861552(409063 converted, 226256 overflowed, 0 dud, 217810 GC'd, 8423 fizzled)

Total time 65.916s (26.964s elapsed)

Eventlog:



Parallel 8 cores:

["abuelita","accesses","added","addict","affaire"]

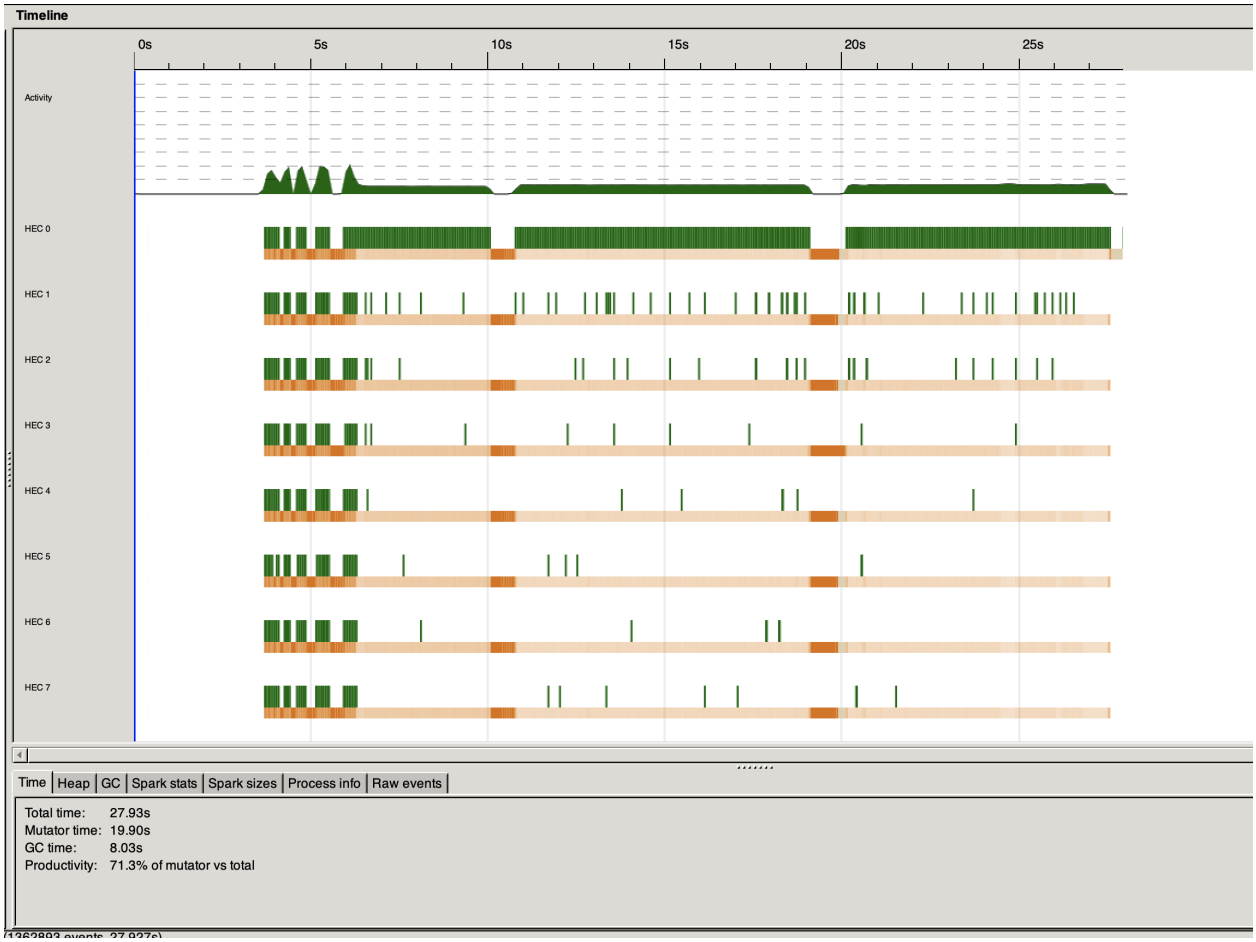
1674 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 18 (1 bound, 17 peak workers (17 total), using -N8)

SPARKS: 861552(429680 converted, 215950 overflowed, 0 dud, 207237 GC'd, 8685 fizzled)

Total time 84.043s (27.927s elapsed)

Eventlog:



Observation for 50 file tests:

Again the parallel improved the total time and run faster than the no parallel.
In this case you can see an increase of improvement in the time when adding cores,
but only up to 6 cores. When running 8 cores there was decrease (it takes longer than 6 cores)
Even though there are more converted sparks in 8 cores than 6 cores.

100 Files Test

Non parallel:

["absolute","abuelita","accesses","actually","addiction"]

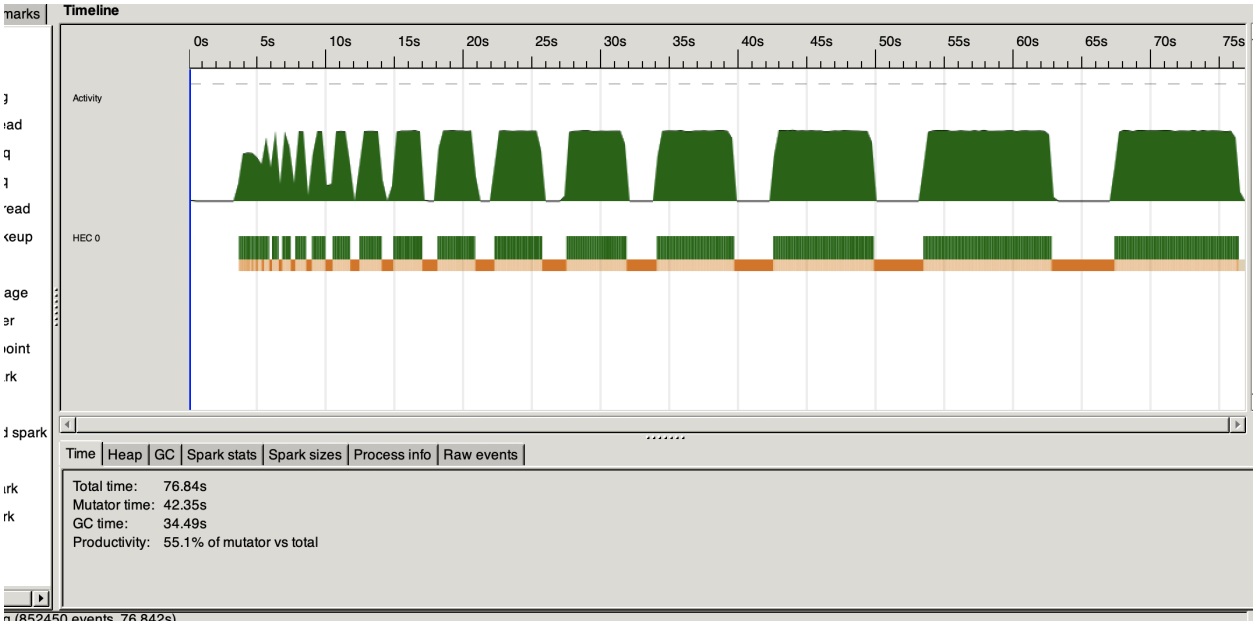
2707 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 4 (1 bound, 3 peak workers (3 total), using -N1)

SPARKS: 0(0 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)

Total time 70.581s (76.842s elapsed)

Eventlog:



Parallel 2 cores:

["absolute","abuelita","accesses","actually","addiction"]

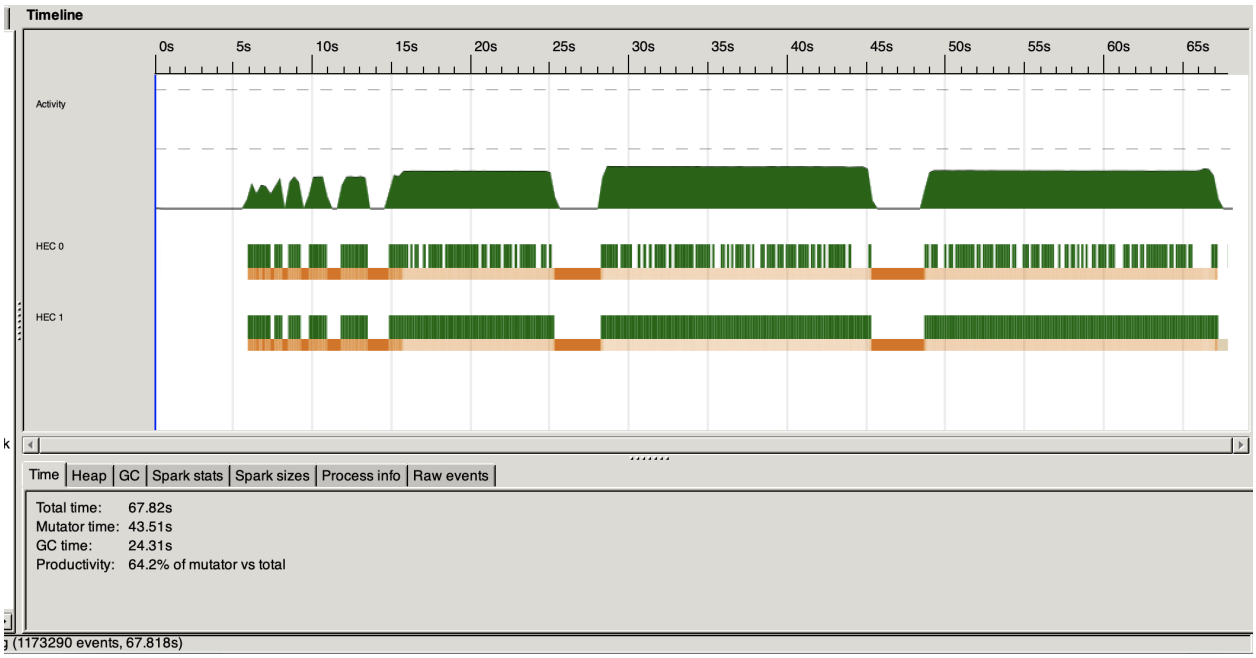
3103 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 6 (1 bound, 5 peak workers (5 total), using -N2)

SPARKS: 1732844(459346 converted, 636766 overflowed, 0 dud, 617172 GC'd, 19560 fizzled)

Total time 86.198s (67.817s elapsed)

Eventlog:



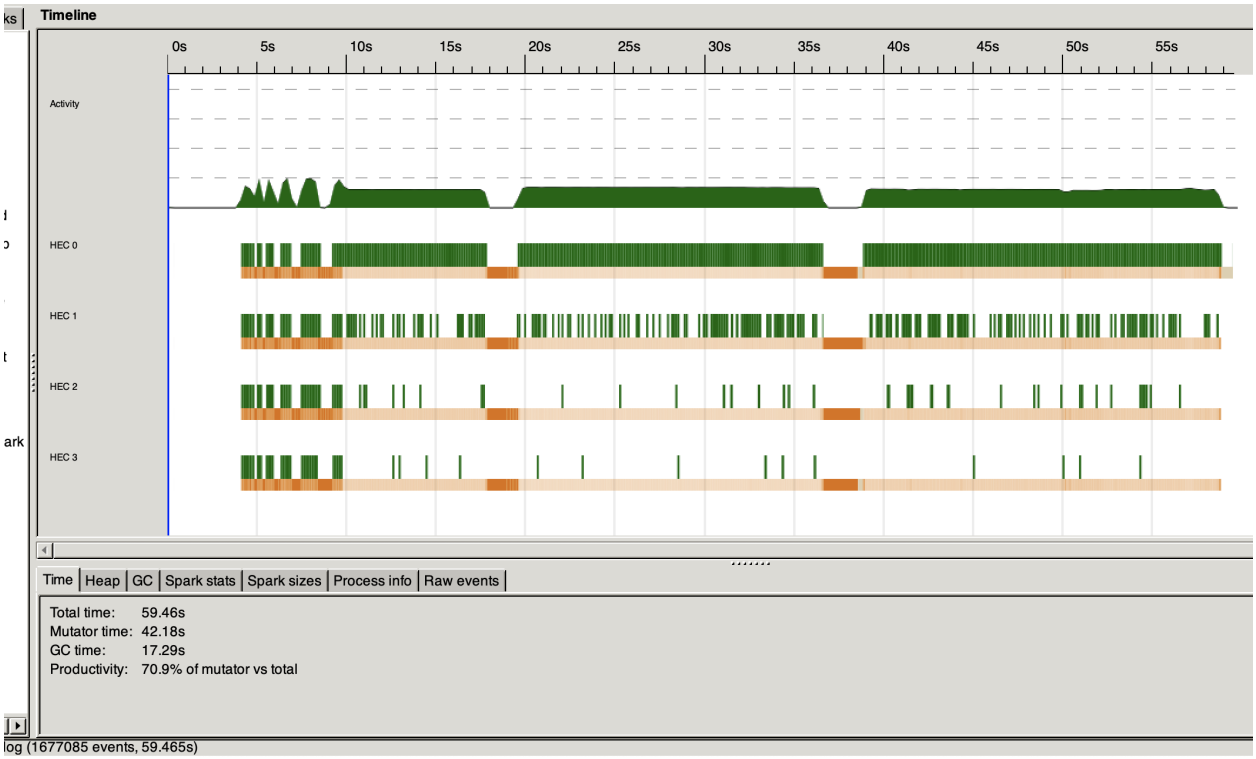
Parallel 4 cores:
["absolute","abuelita","accesses","actually","addiction"]
3128 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 10 (1 bound, 9 peak workers (9 total), using -N4)

SPARKS: 1732844(687702 converted, 522588 overflowed, 0 dud, 503350 GC'd, 19204 fizzled)

Total time 115.156s (59.464s elapsed)

Eventlog:



Parallel 6 cores:

["absolute","abuelita","accesses","actually","addiction"]

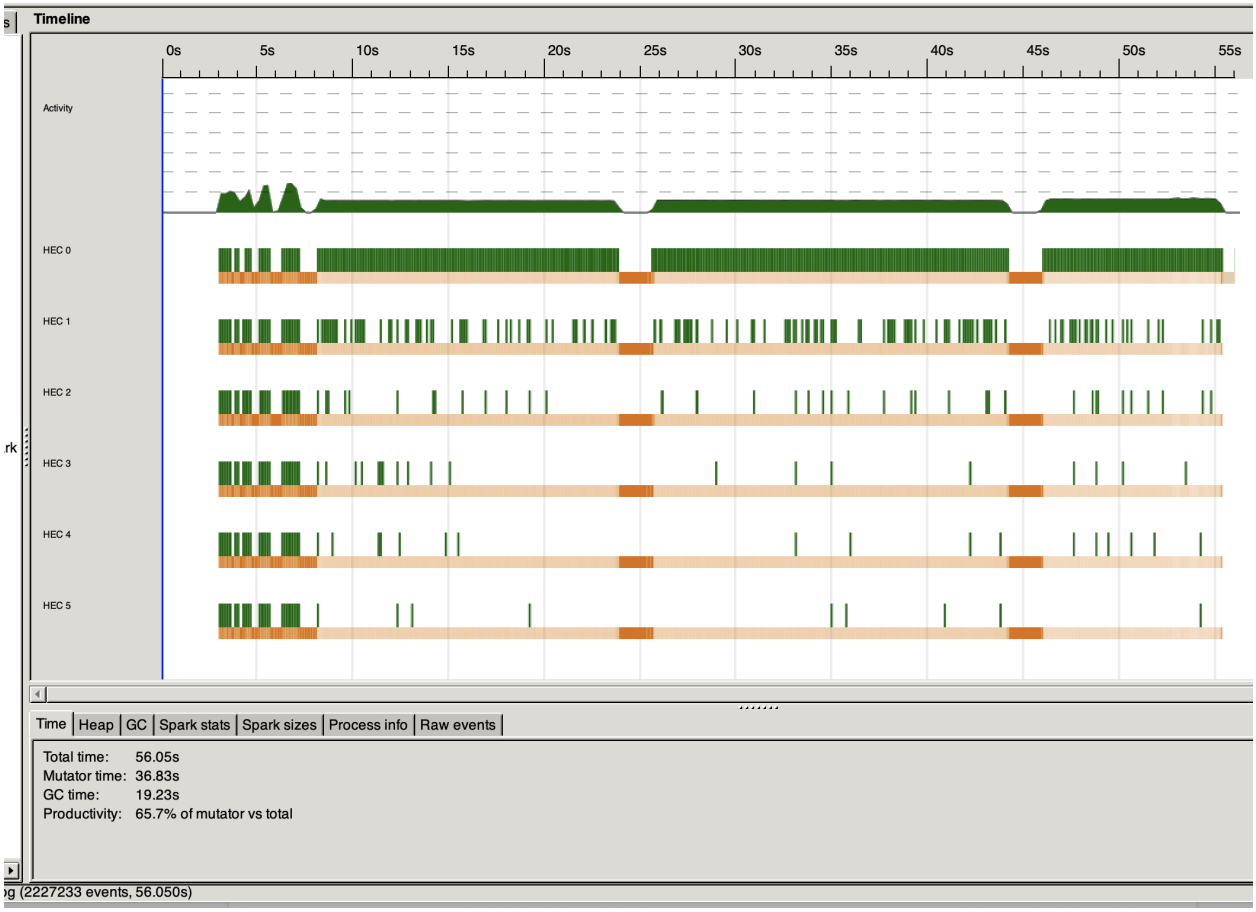
3267 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 14 (1 bound, 13 peak workers (13 total), using -N6)

SPARKS: 1732844(770594 converted, 481146 overflowed, 0 dud, 465095 GC'd, 16009 fizzled)

Total time 151.988s (56.050s elapsed)

Eventlog :

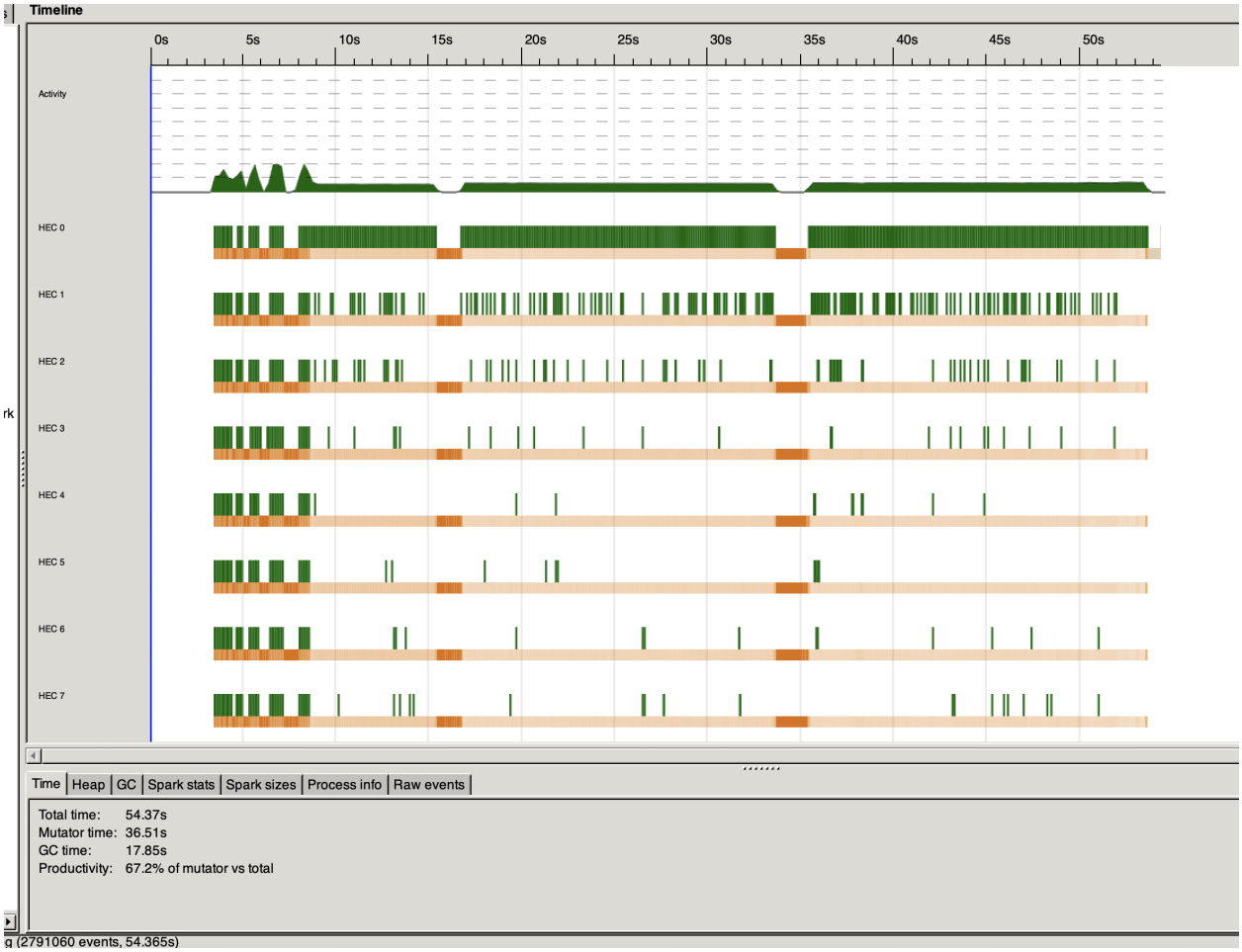


Parallel 8 cores:

["absolute","abuelita","accesses","actually","addiction"]
3230 MB total memory in use (0 MB lost due to fragmentation)

TASKS: 18 (1 bound, 17 peak workers (17 total), using -N8)
SPARKS: 1732844(819232 converted, 456825 overflowed, 0 dud, 437771 GC'd, 19016 fizzled)
Total time 178.106s (54.365s elapsed)

Eventlog:



Observation for test 100:

In the 100 tests file it is no surprise that it runs the longest time and uses the largest memory. The total time of no parallel is 76.84 seconds while all the parallel run faster than no parallel and use 2707 mb of fmemory

Each time I add 2 cores it improves the result - the best result is 54.365.

The memory of using parallel is more than in no parallel but it does not increase a lot.

Regarding the sparks, as I increase the number of cores the converted sparks increase.

Overall the best result was 8-cores in terms of converted sparks and terms of speed (54.3 seconds)

Discussion and conclusion:

Overall, in all the tests parallel running received the best results.
In the two files it is the only time where the no parallel was run
faster than 2 and 4 cores parallel.

This is not a surprise because we can assume that to run in parallel takes
some time and only in bigger tasks we can see a significant improve between
parallel and no parallel (first conclusion).

Second, the tasks show that adding more cores does not guarantee always
the best result but 3 out of 5 tasks the more cores you add the faster
the tasks are completed.

Third, converting more sparks does not necessarily improve the result
– as I saw in a few tasks – the one that runs faster
not always has the highest number of converted sparks.

In conclusion, after running all the tests using parallel improve the result,
but I can see a much bigger difference when I run a more complex/heavy tasks –
in small tasks the difference is almost does not exist because it takes a little
time to run in parallel and in short tasks that run very small – the time it
takes to run in parallel impact the result. Our smartphones and computers have
already more than one core, running my project shows that auto suggest can use
and benefit the users by getting a result faster.