Parallelism

EDAF/N40: Functional Programming
On Concurrency and Parallelism

Jacek Malec
Dept. of Computer Science, Lund University, Sweden
May 6th, 2019

Topic of today: concurrency and parallelisation

References:
- O’Sullivan, Goerzen, Stewart: Real World Haskell, Chapter 24,
- Peyton Jones, Singh: A tutorial on parallel and concurrent programming in Haskell
- Chakravarty, Leshchinskiy, Peyton Jones, Keller, Marlow: Data Parallel Haskell: a status report
- Marlow, Parallel and Concurrent Programming in Haskell

Short note: Profiling

- Compile your program with the -prof and -fprof-auto options
- Run it with +RTS option -p

Example:

```haskell
main = print (fib 30)
fib n = if n < 2 then 1 else fib (n-1) + fib (n-2)
```

Compile and run:

```
jacek$ ghc -prof -fprof-auto -rtsopts FibProfile.hs
[1 of 1] Compiling Main ( FibProfile.hs, FibProfile.o )
Linking FibProfile ...
jacek$ ./FibProfile +RTS -p
1346269
jacek$ less FibProfile.prof
```
### Profiling, cont.

Sat May 4 00:13 2019 Time and Allocation Profiling Report (Final)

FibProfile +RTS -p -RTS

total time = 0.33 secs (332 ticks @ 1000 us, 1 processor)
total alloc = 409,314,936 bytes (excludes profiling overheads)

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>SRC</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>Main</td>
<td>FibProfile.hs:2:1-50</td>
<td>99.7</td>
<td>100.0</td>
</tr>
</tbody>
</table>

individual inherited

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>no. entries</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>MAIN</td>
<td>115</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>Main</td>
<td>229</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>fib</td>
<td>Main</td>
<td>232</td>
<td>2692537</td>
<td>99.7</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.Conc.Signal</td>
<td>209</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Encoding</td>
<td>191</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Encoding.Iconv</td>
<td>189</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Handle.FD</td>
<td>180</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Handle.Text</td>
<td>178</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>main</td>
<td>Main</td>
<td>231</td>
<td>0.3</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Jacek Malec, http://rss.cs.lth.se

---

Tue Dec 1 13:24 2015 Time and Allocation Profiling Report (Final)

FibProfile +RTS -p -RTS

total time = 0.31 secs (311 ticks @ 1000 us, 1 processor)
total alloc = 452,395,728 bytes (excludes profiling overheads)

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib</td>
<td>Main</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

individual inherited

<table>
<thead>
<tr>
<th>COST CENTRE</th>
<th>MODULE</th>
<th>no. entries</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN</td>
<td>MAIN</td>
<td>44</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>Main</td>
<td>87</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>main</td>
<td>Main</td>
<td>88</td>
<td>1</td>
<td>0.0</td>
</tr>
<tr>
<td>fib</td>
<td>Main</td>
<td>89</td>
<td>2692537</td>
<td>100.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.Conc.Signal</td>
<td>83</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Encoding</td>
<td>77</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Encoding.Iconv</td>
<td>75</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Handle.FD</td>
<td>67</td>
<td>0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF</td>
<td>GHC.IO.Handle.Text</td>
<td>65</td>
<td>0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Jacek Malec, http://rss.cs.lth.se

---

### Profiling, cont.

CAF (Constant Applicative Form) = top-level thunk

2nd example:

main = print (f 30 + g 30)

where

\[ f \ n = \text{fib} \ n \]

\[ g \ n = \text{fib} \ (n \ '\text{div}' \ 2) \]

fib n = if n < 2 then 1 else fib (n-1) + fib (n-2)
Profiling, cont.

Sat May 4 00:16 2019 Time and Allocation Profiling Report (Final)

FibProfile2 +RTS -p -RTS

total time = 0.34 secs (342 ticks @ 1000 us, 1 processor)
total alloc = 495,839,264 bytes (excludes profiling overheads)

<table>
<thead>
<tr>
<th>COST CENTRE MODULE SRC</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib Main FibProfile2.hs:6:1-50</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

individual inherited

<table>
<thead>
<tr>
<th>COST CENTRE MODULE SRC</th>
<th>no. entries</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN MAIN &lt;built-in&gt;</td>
<td>115</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF Main &lt;entire-module&gt;</td>
<td>229</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>main Main FibProfile2.hs:(1,1)-(4,25)</td>
<td>230</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>main.f Main FibProfile2.hs:3:5-15</td>
<td>234</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>main.g Main FibProfile2.hs:6:5-25</td>
<td>233</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>fib Main FibProfile2.hs:6:1-50</td>
<td>235</td>
<td>99.9</td>
<td>99.9</td>
</tr>
<tr>
<td>CAF GHC.Conc.Signal &lt;entire-module&gt;</td>
<td>209</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Encoding &lt;entire-module&gt;</td>
<td>191</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Encoding.Iconv &lt;entire-module&gt;</td>
<td>189</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Handle.FD &lt;entire-module&gt;</td>
<td>180</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Handle.Text &lt;entire-module&gt;</td>
<td>178</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>main Main FibProfile2.hs:(1,1)-(4,25)</td>
<td>233</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Jacek Malec, http://rss.cs.lth.se

Parallelism

Mon May 6 10:31 2019 Time and Allocation Profiling Report (Final)

FibProfile2 +RTS -ls -N4 -p -RTS

total time = 0.07 secs (273 ticks @ 1000 us, 4 processors)
total alloc = 495,917,456 bytes (excludes profiling overheads)

<table>
<thead>
<tr>
<th>COST CENTRE MODULE SRC</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>fib Main FibProfile2.hs:6:1-50</td>
<td>100.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

individual inherited

<table>
<thead>
<tr>
<th>COST CENTRE MODULE SRC</th>
<th>no. entries</th>
<th>%time</th>
<th>%alloc</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAIN MAIN &lt;built-in&gt;</td>
<td>115</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF Main &lt;entire-module&gt;</td>
<td>229</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>main Main FibProfile2.hs:(1,1)-(4,25)</td>
<td>230</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>main.f Main FibProfile2.hs:3:5-15</td>
<td>234</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>fib Main FibProfile2.hs:6:1-50</td>
<td>235</td>
<td>99.6</td>
<td>99.6</td>
</tr>
<tr>
<td>main.g Main FibProfile2.hs:6:5-25</td>
<td>232</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.Conc.Signal &lt;entire-module&gt;</td>
<td>209</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Encoding &lt;entire-module&gt;</td>
<td>191</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Encoding.Iconv &lt;entire-module&gt;</td>
<td>189</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Handle.FD &lt;entire-module&gt;</td>
<td>180</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>CAF GHC.10.Handle.Text &lt;entire-module&gt;</td>
<td>178</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>main Main FibProfile2.hs:(1,1)-(4,25)</td>
<td>233</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Jacek Malec, http://rss.cs.lth.se

A tool for visualisation: ThreadScope

HEC - Haskell Execution Context

 Compile your program with the -threaded and -eventlog options
 Run it with +RTS option -ls

Compile and run:

```bash
jacek$ ghc -rtsopts -threaded -eventlog FibProfile.hs
[1 of 1] Compiling Main ( FibProfile.hs, FibProfile.o )
Linking FibProfile ...

jacek$ ./FibProfile +RTS -ls -N2 1346269
jacek$ threadscope FibProfile.eventlog
```

Jacek Malec, http://rss.cs.lth.se
Threads

In Control.Concurrent module:

:t forkIO
forkIO :: IO () -> IO ThreadID
:m +System.Directory
forkIO (writeFile "foobar" "this is sth") >>
doesFileExist "foobar"

(import qualified Data.ByteString.Lazy as L)
do
  content <- L.readFile name
  forkIO (compressFile name content)
  return ()

Communication between threads using synchronising variables:

-- ch24/MVarExample.hs
import Control.Concurrent

communicate = do
  m <- newEmptyMVar
  forkIO $ do
    v <- takeMVar m
    putStrLn "sending"
    putMVar m "wake up!"

Threads, cores, etc.

- I will NOT talk about concurrency (synchronisation, etc.) any more
- Concurrent computations need to be requested *explicitly* in case you want to specify # of cores/processors, otherwise the default!
  - when compiling (-threaded), actually while linking
  - when invoking (+RTS -N<#> -RTS)
  - possibly also in the code itself
- Weighing needs to be done in an informed way (Haskell threads vs. OS threads)
- When the main thread finishes, all other get killed. Watch out!

Normal forms

- NF (RNF) – normal form (reduced normal form)
- HNF – head normal form
- WHNF – weak head normal form

Describe the amount of evaluation performed:
- NF – evaluated
- WHNF – evaluated only up to the outermost constructor

Check http://stackoverflow.com/questions/6872898/haskell-what-is-weak-head-normal-form
Some introductory stuff

Some definitions:

- **Strict vs. lazy evaluation**;
- **Space leaks (foldl vs. foldl’)**

\[
\text{foldl} \ (+) \ 0 \ (1:2:3:[]) \\
= \text{foldl} \ (+) \ (0 + 1) \ (2:3:[]) \\
= \text{foldl} \ (+) \ ((0 + 1) + 2) \ (3:[]) \\
= \text{foldl} \ (+) \ (((0 + 1) + 2) + 3) \ [] \\
= (((0 + 1) + 2) + 3)
\]

*Thunks* stored until needed.

Usage of *seq*

Problematic:

- `hiddenInside x y = someFunc (x ‘seq‘ y)`
- `hiddenByLet x y z = let a = x ‘seq‘ someFunc y in anotherFunc a z`
- `badExpression step zero (x:xs) = seq (step zero x) (badExpression step (step zero x) xs)`

Below are reasonable cases:

- `onTheOutside x y = x ‘seq‘ someFunc y`
- `chained x y z = x ‘seq‘ y ‘seq‘ someFunc z`
- `strictPair (a,b) = a ‘seq‘ b ‘seq‘ (a,b)`

Sorting examples

An ordinary quicksort:

\[
\begin{align*}
\text{sort} &: (\text{Ord } a) \Rightarrow [a] \Rightarrow [a] \\
\text{sort} \ (x:xs) &= \text{lesser} ++ x : \text{greater} \\
&\quad \text{where lesser} = \text{sort} \ [y \mid y < xs, y < x] \\
&\quad \text{greater} = \text{sort} \ [y \mid y < xs, y > x] \\
\text{sort} \ _ &= []
\end{align*}
\]
Parallelism

Sorting examples

A parallel version of quicksort:

```haskell
import Control.Parallel (par, pseq)

parSort :: (Ord a) => [a] -> [a]
parSort (x:xs) = force grtr 'par' (force lesser 'pseq'
  (lesser ++ x:grtr))
  where lesser = parSort [y | y <- xs, y < x]
    grtr = parSort [y | y <- xs, y >= x]
parSort _ = []
```

New stuff: `par`, `pseq` (from Control.Parallel) and `force` (later)

Jacek Malec, http://rss.cs.lth.se

Parallelism

Sparks

Another example to illustrate sparks:

```haskell
mkList :: Int -> [Int]
mkList n = [1..n-1]

relprime :: Int -> Int -> Bool
relprime x y = gcd x y == 1

euler :: Int -> Int
euler n = length (filter (relprime n) (mkList n))

sumEuler :: Int -> Int
sumEuler = sum . (map euler) . mkList
```

Jacek Malec, http://rss.cs.lth.se

Parallelism

Sparks, cntd.

```haskell
fib :: Int -> Int
fib 0 = 0
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

And their sum

```haskell
sumFibEuler :: Int -> Int -> Int
sumFibEuler a b = fib a + sumEuler b
sumFibEuler 38 5300 seems reasonable for tests.
```

Jacek Malec, http://rss.cs.lth.se
Parallel solutions

parSumFibEuler :: Int -> Int -> Int
parSumFibEuler a b
  = f 'par' (f + e)
  where
    f = fib a
    e = sumEuler b

Spark, but no thread. Why?

Better, but pure luck! Compiler can change a + b into b + a at will!

Parallel solutions 3

parSumFibEuler :: Int -> Int -> Int
parSumFibEuler a b
  = f 'par' (e 'pseq' (e + f))
  where
    f = fib a
    e = sumEuler b

(or (e 'pseq' (f + e)) as well)

Sorting examples

Knowing what to evaluate in parallel (possible interference with strict evaluation).

sillySort (x:xs) = grtr 'par' (lesser 'pseq'
  (lesser ++ x:grtr))
  where lesser = sillySort [y | y <- xs, y < x]
  grtr = sillySort [y | y <- xs, y >= x]
sillySort _ = []

Evaluates just one element in each sublist! Almost everything is sequential!
Parallelism: divide and conquer

The Google’s MapReduce concept: massive parallelism in the cloud.

The Haskell counterpart below.

Example: processing Apache log files

```
201.49.94.87 - - [08/Jul/2008:07:04:20 -0500] "GET / HTTP/1.1"
200 2097 "http://en.wikipedia.org/wiki/Mercurial_(software)"
Mozilla/5.0 (Windows; U; Windows XP 5.1; en-GB; rv:1.8.1.12)
Gecko/20080201 Firefox/2.0.0.12" 0 hgbook.red-bean.com
```

Parallelism

Sorting examples

Forcing the entire spine of a list to be evaluated before returning a constructor:

```
force :: [a] -> ()
force xs = go xs 'pseq' ()
  where go (_:xs) = go xs
       go [] = 1
```

Modified (for my tests):

```
force :: [a] -> ()
force xs = go xs 'seq' ()
  where go (x:xs) = x 'seq' (go xs)
       go [] = 1
```

Parallelism

Sorting examples

Fine-tuning parallelism:

```
parSort2 :: (Ord a) => Int -> [a] -> [a]
parSort2 d list@(x:xs)
  | d <= 0     = sort list
  | otherwise  = force greater 'par' (force lesser 'pseq'
                                (lesser ++ x:greater))
    where lesser = parSort2 d' [y | y <- xs, y < x]
          greater = parSort2 d' [y | y <- xs, y >= x]
            d'   = d - 1
        parSort2 _ _ = []
```

On the other hand, `par` may be used quite freely, as the compiler has the freedom to not obey it.

Parallelism

The problem

Lots of interesting questions may be asked that require processing the huge data set (log file(s)) in order to find the answer for.

Recurring issues:

- An algorithm quickly becomes obscured by the details of partitioning and communication.
- Choosing a “grain size” — the smallest unit of work parceled out to a core — can be difficult. Too large = not efficient. Too small = too much bookkeeping.
import Control.Parallel (par)

parallelMap :: (a -> b) -> [a] -> [b]
parallelMap f (x:xs) = let r = f x
                   in r `par` r : parallelMap f xs
parallelMap _ _ = []

Issue: evaluation stops at the constructor level. What if a is a list itself?

forceList :: [a] -> ()
forceList (x:xs) = x `pseq` forceList xs
forceList _ = ()

Strategies

We have encountered an evaluation strategy:

data Eval a = Seq a | Par a | Lazy a

type Strategy a = a -> Eval a

An evaluation strategy performs no computation; it simply ensures that a value is evaluated to some extent. The simplest strategy is named \( r_0 \), and does nothing at all:

\[ r_0 :: \text{Strategy } a \]

\[ r_0 _ = () \]

We have also encountered this one:

\[ rwhnf :: \text{Strategy } a \]

\[ rwhnf x = x `\text{seq}' () \]
If we want to evaluate an arbitrary type \( a \) further:

```haskell
class NFData a where
  rnf :: Strategy a
  rnf = rwhnf
```

Note that this is a default definition!

```haskell
instance NFData a => NFData (Maybe a) where
  rnf Nothing = ()
  rnf (Just x) = rnf x
```

Jacek Malec, http://rss.cs.lth.se

---

**MapReduce**

```haskell
simpleMapReduce :: (a -> b) -- map function
                -> ([b] -> c) -- reduce function
                -> [a] -- list to map over
                -> c
```

```haskell
simpleMapReduce mapFunc reduceFunc = reduceFunc . map mapFunc
```

Jacek Malec, http://rss.cs.lth.se
MapReduce with strategies

```hs
mapReduce :: Strategy b  -- evaluation strategy for mapping
    -> (a -> b)      -- map function
    -> Strategy c  -- evaluation strategy for reduction
    -> ([b] -> c)  -- reduce function
    -> [a]          -- list to map over
    -> c

mapReduce mapStrat mapFunc reduceStrat reduceFunc input =
    mapResult `pseq` reduceResult
where mapResult = parMap mapStrat mapFunc input
    reduceResult = reduceFunc mapResult `using`
                    reduceStrat
```

Jacek Malec, http://rss.cs.lth.se

37(39)

More on parallelism

MapReduce framework (Google’s and ours) is an example of (flat) data parallelism. Limiting.

Nested data parallelism is more expressive, supporting irregular and sparse data structures. Original idea may be found in the NESL language. Focus lies on multicore CPUs.

Look for Data Parallel Haskell: libraries, tutorials, compiler versions, etc.

GPH (Glasgow Parallel Haskell) is an old fork (out of 4.06) of ghc towards clusters and other types of multiprocessors. ghc supports only shared-memory multiprocessor.

Jacek Malec, http://rss.cs.lth.se

39(39)

Chunking

The textbook (RWH, Chapter 24) contains an interesting discussion on chunking the input into appropriately-sized parts amenable to parallel processing.

Lines — evidently too little. How much is OK?

Interplay with lazy reading of files - requires careful attention.

Domain-dependent.