

Parallel Evaluation and Laziness

Max Levatch & Stephen A. Edwards

Columbia University

Fall 2025



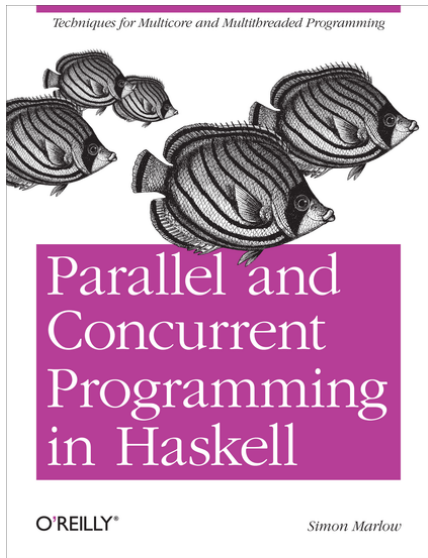
Parallelism

- ThreadScope

- Sparking Parallelism with par

- Sparks

- Limiting Granularity



This material adapted from

Simon Marlow's book

<https://simonmar.github.io/pages/pcph.html>

Mary Sheeran and John Hughes's class

http://www.cse.chalmers.se/edu/year/2018/course/DAT280_Parallel_Functional_Programming/lectures.html

Let's Speed Up a Dumb[†] Program

```
nfib1 :: Integer -> Integer
nfib1 n | n < 2 = 1
nfib1 n = nfib1 (n-1) + nfib1 (n-2) + 1

main :: IO ()
main = print (nfib1 40)
```

<i>n</i>	nfib <i>n</i>
10	177
20	21891
25	242785
30	2692537
35	29860703
40	331160281

```
$ stack ghc -- -O2 \           # Optimize
                        -threaded \ # Enable parallel execution
                        -rtsopts \  # Enable run-time system flags +RTS
                        nfib1.hs
```

[†]This should be iterative, not recursive

Running the Program

```
$ TIMEFORMAT="real %Rs"      # for bash time builtin
$ time ./nfib1
331160281
real 5.472s
$ time ./nfib1 +RTS -N1      # +RTS = Run Time System, -N1 = 1 core
331160281
real 5.462s
$ time ./nfib1 +RTS -N4      # -N4 = use 4 cores
331160281
real 5.852s
$ time ./nfib1 +RTS -N4 -ls  # -ls = Record events in nfib1.eventlog
331160281
real 5.924s
```

ThreadScope

ThreadScope: the Haskell parallel execution event log viewer

Under Ubuntu, I was able to install it using Aptitude:

```
$ sudo apt install threadscope
```

Under stack and Ubuntu, it needs GTK libraries to compile and wasn't happy with the lts-22.33 resolver, but would compile:

```
$ sudo apt-get install libgtk2.0-dev libpango1.0-dev \  
    libglib2.0-dev libcairo2-dev  
$ stack install --resolver lts-20.26 threadscope
```

A Haskell executable compiled with `-rtsopts` enables the `+RTS ... -RTS` syntax for passing arguments to the Haskell runtime system

The `-l` option enables event logging (in a binary file *executable.eventlog*); `s` includes scheduler events

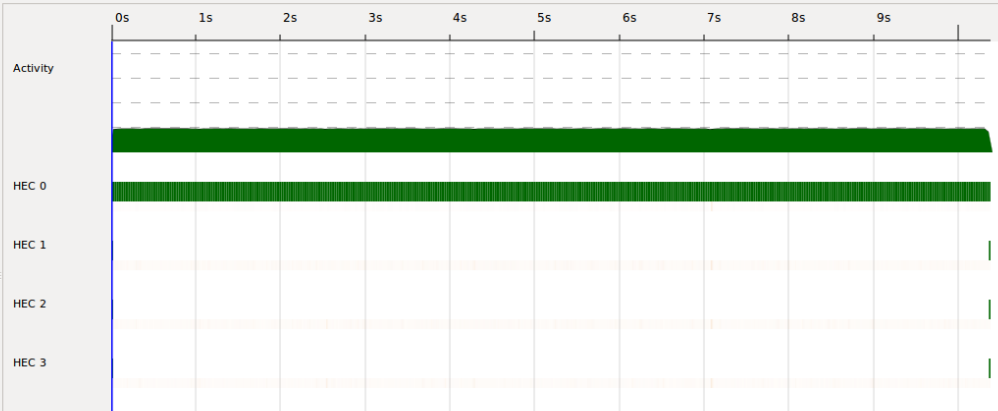
Google “Haskell Runtime Control” or look in the GHC User Guide



Key Traces Bookmarks

Timeline

- running
- GC
- create thread
- seq GC req
- par GC req
- migrate thread
- thread wakeup
- shutdown
- user message
- perf counter
- perf tracepoint
- create spark
- dud spark
- overflowed spark
- run spark
- fizzled spark
- GCed spark



Time Heap GC Spark stats Spark sizes Process info Raw events

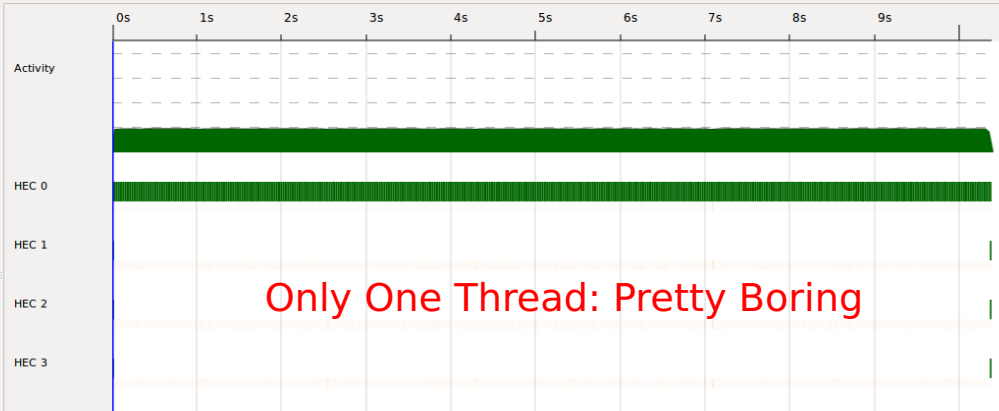
Total time: 10.37s
Mutator time: 10.21s
GC time: 0.16s
Productivity: 98.4% of mutator vs total



Key Traces Bookmarks

- running
- GC
- create thread
- seq GC req
- par GC req
- migrate thread
- thread wakeup
- shutdown
- user message
- perf counter
- perf tracepoint
- create spark
- dud spark
- overflowed spark
- run spark
- fizzled spark
- GCed spark

Timeline



Time Heap GC Spark stats Spark sizes Process info Raw events

Total time: 10.37s
Mutator time: 10.21s
GC time: 0.16s
Productivity: 98.4% of mutator vs total

Asking for Parallelism

In `Control.Parallel`, (`stack install parallel`)

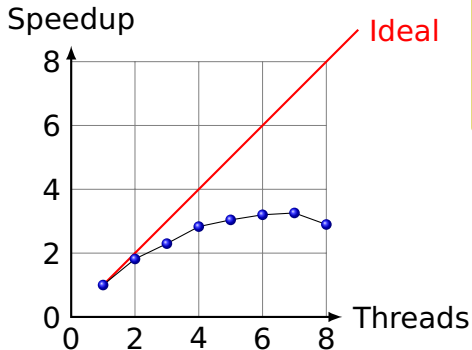
```
par : a -> b -> b
```

`par x y` “sparks” the evaluation of `x` in parallel with `y`; returns `y`.

The run-time system *may* convert a spark into work for a thread

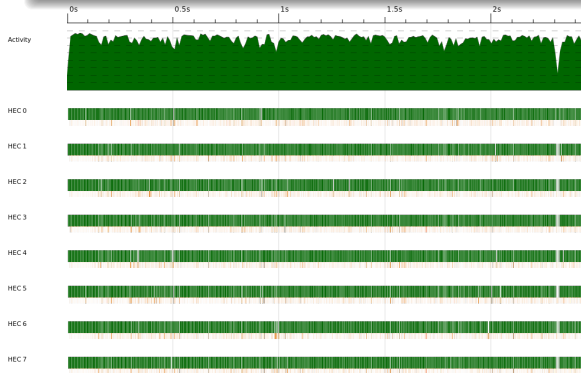
```
import Control.Parallel(par)  
  
nfib2 :: Integer -> Integer  
nfib2 n | n < 2 = 1  
nfib2 n = par nf (nf + nfib2 (n-2) + 1)  
    where nf = nfib2 (n-1)
```

Performance of nfib2



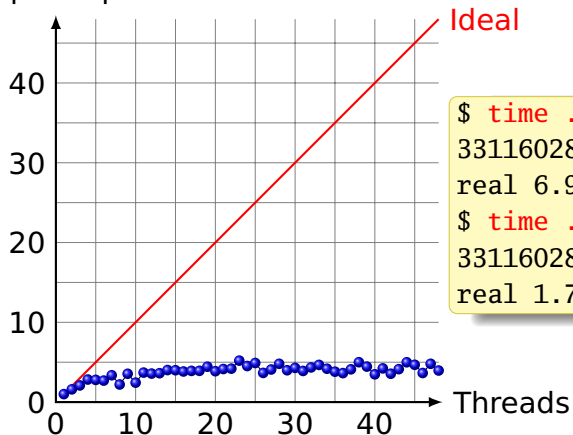
On a 8 thread (4 core)
i7-3820
A speedup of 3.2 max

```
$ time ./nfib2 +RTS -N1
331160281
real 7.312s
$ time ./nfib2 +RTS -N8
331160281
real 2.524s
```

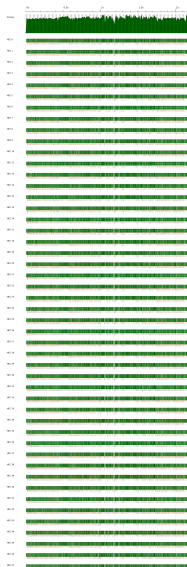


Performance of nfib2

Speedup

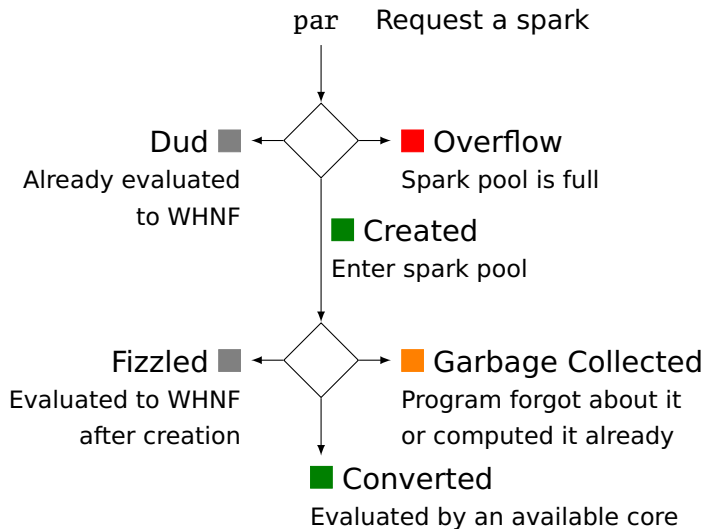


```
$ time ./nfib2 +RTS -N1
331160281
real 6.922s
$ time ./nfib2 +RTS -N48
331160281
real 1.749s
```



On a 48 thread (48 core) Xeon 4214: A speedup of 5 max

Sparks



```
$ ./nfib2 +RTS -N8 -s
331160281
SPARKS:
194364898 total
      10671 converted
121033805 overflowed
         0 dud
    2594777 GC'd
    46510093 fizzled
```

Conclusion: Far too many sparks created; over half didn't even fit in the spark pool, over half overflowed. 25% didn't even fit in the spark pool. Only 10671 (0.005%) did useful work.



Asking more precisely for parallelism

Also in Control.Parallel,

```
pseq : a -> b -> b
```

Like seq, but only strict in its first argument. pseq x y means “make sure x is evaluated before starting on y”

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

```
nfib3 :: Integer -> Integer
```

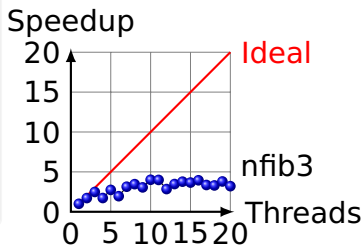
```
nfib3 n | n < 2 = 1
```

```
nfib3 n = nf1 `par` nf2 `pseq` nf1 + nf2 + 1
```

```
  where nf1 = nfib3 (n-1)
```

```
        nf2 = nfib3 (n-2)
```

No visible change in performance; the compiler may have automatically done this for us



Controlling Granularity

We are creating a *lot* of sparks, most of which are pointless:

```
$ ./nfib3 +RTS -N48 -s
SPARKS: 315889730
        (294622 converted,
        91778269 overflowed,
         0 dud,
        8191203 GC'd,
        193400740 fizzled)
```

It doesn't make sense to be creating 315 million pieces of work when we only have 48 threads capable of doing work in parallel; only 294622 ever did useful work.

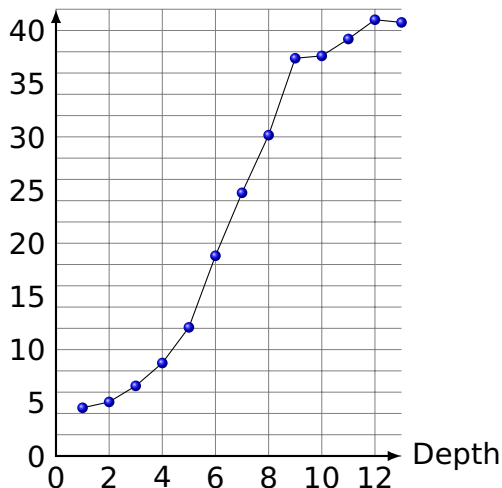
Idea: let's go parallel **only to a certain depth**

Running Parallel to a Certain Depth

```
nfib4 :: Int -> Int -> Integer
nfib4 0 n          = nfib n
nfib4 _ n | n < 2 = 1
nfib4 d n = nf1 `par` nf2 `pseq`
            nf1 + nf2 + 1
    where nf1 = nfib4 (d-1) (n-1)
          nf2 = nfib4 (d-1) (n-2)

nfib :: Int -> Integer
nfib n | n < 2 = 1
nfib n = nfib (n-1) +
         nfib (n-2) + 1
```

Speedup

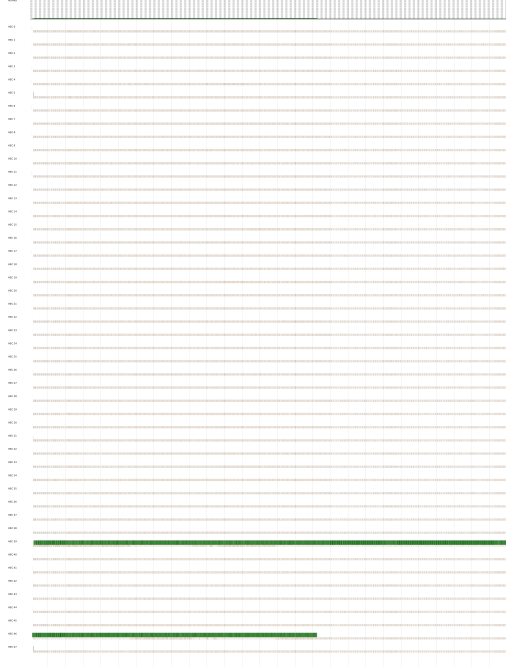


`nfib4 40` on a 48-thread Dual Xeon 4214

Depth	Sparks				Time (s)		Speed
	total	converted	GC'ed	fizzled	total	elapsed	
1	1	1	0	0	11.64	2.57	4.
2	3	3	0	0	8.64	1.70	5.
3	7	7	0	0	6.67	1.01	6.
4	15	15	0	0	5.91	0.68	8.
5	31	31	0	0	6.32	0.52	12.
6	63	61	0	1	7.42	0.39	18.
7	127	110	0	16	7.35	0.30	24.
8	257	209	1	46	7.28	0.24	30.
9	521	344	6	170	7.51	0.20	37.
10	1052	625	37	389	7.64	0.20	37.
11	2150	1246	267	636	8.36	0.21	39.
12	4455	2087	998	1370	8.75	0.21	41.
13	9405	3924	2969	2510	10.09	0.25	40.
25	54835638	585256	3992042	32348255	31.22	0.83	37.

48-thread, dual 2.2 GHz Xeon 4214 +RTS -N48 -s, 4-run averages, -02 -threaded
-rtsopts

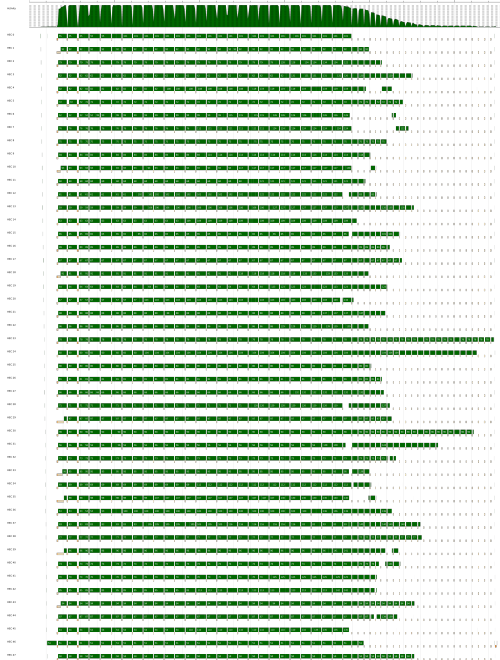
Depth = 1: Only two-way
parallelism



Depth = 4: Unbalanced 16-way parallelism



Depth = 7: 128 sparks; better
balancing



Depth = 12: 4000+ sparks;
excellent balancing

