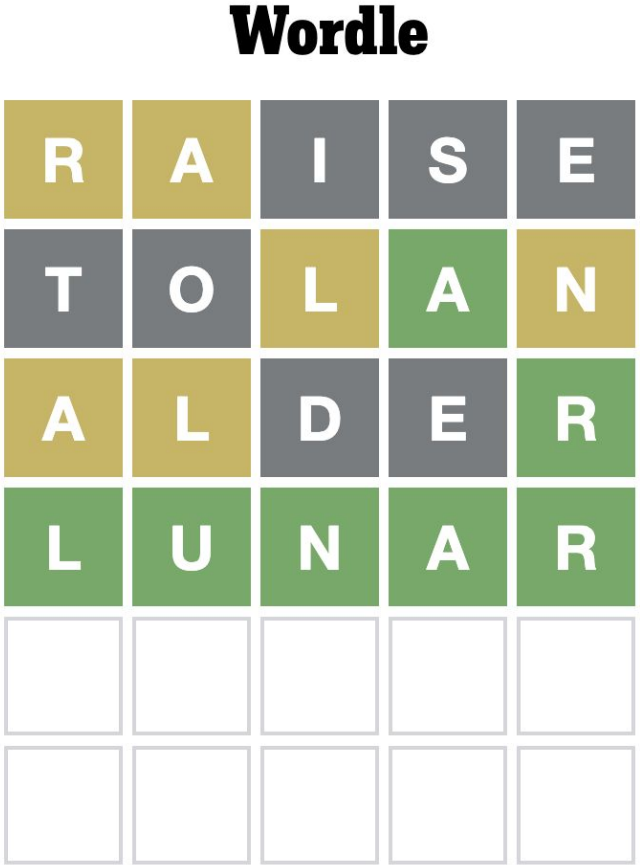
ParWordle

Sanjay Rajasekharan and Zac Coeur



Wordle Algorithm

$A = W = S$ ▷ S is the set of all 5-letter words
G = {}
initialize K ▷ K represents our knowledge about the answer
$g =$ **argmax** $\{E(w)|w \in W\}$
while $g \neq a$ **do** ▷ a is the Wordle solution
    $K \leftarrow guess(g)$
    $A \leftarrow K$ ▷ filter possible answers utilizing new knowledge
    $G = G \cup \{g\}$
    $g =$ **argmax** $\{E(w)|w \in W\}$
end while
output G

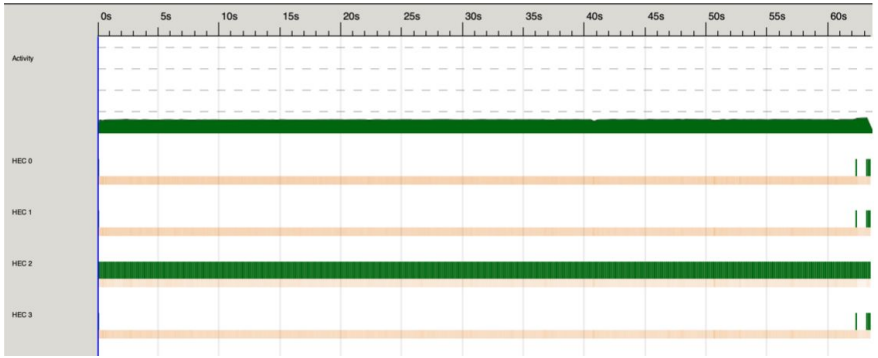
$$E(w) = \sum_x p(x) * \log_2(1/p(x))$$

Able to solve words with an average of 4.13 guesses

Parallelization

- Our main strategy was focusing on parallelizing the entropies function
  - entropies iterates over ~12,000 word guess list, calculating entropy for each

`using` parList rseq



- all sparks were converted
- but basically no parallelization
- much slower than sequential
- rseq doesn't force evaluation, as it is iterating through ByteStrings (lazy)
- use rdeepseq

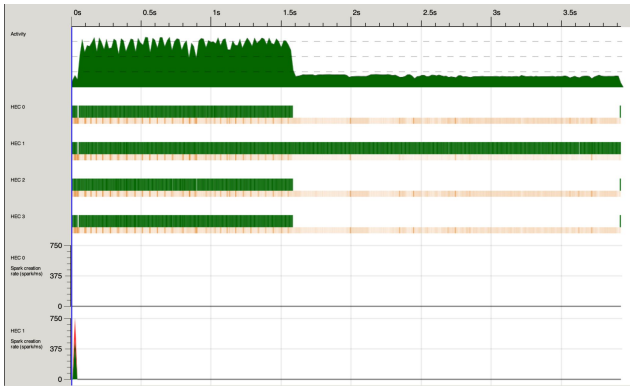
	sequential	`using` parList rseq on 4 cores
"hello"	39.17s	63.71s
"inter"	5.84s	6.69s
"shine"	2.26s	1.64s

```
SPARKS: 38910 (38910 converted, 0 overflowed, 0 dud, 0 GC'd, 0 fizzled)
```

`using` parList rdeepseq

- Running in parallel!
- But only for half of the runtime– about the same as sequential
- Sparks overflowed due to the large list size
- Try parBuffer

	sequential	`using` parList rdeepseq on 4 cores
“hello”	39.17s	36.79s
“inter”	5.84s	3.93s
“shine”	2.26s	3.03s

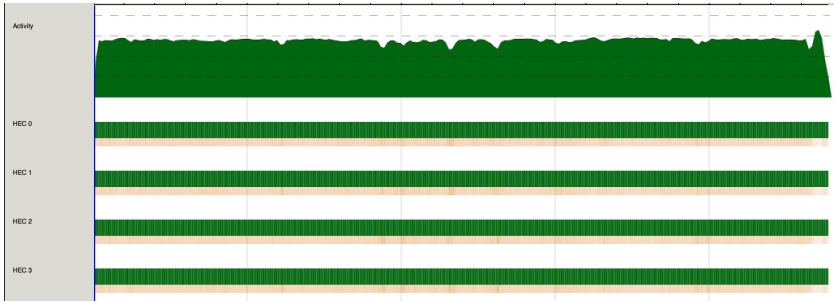



```
SPARKS: 12971 (8589 converted, 4382 overflowed, 0 dud, 0 GC'd, 0 fizzled)
```

```
SPARKS: 38910 (38906 converted, 0 overflowed, 0 dud, 0 GC'd, 4 fizzled)
```

`using` parBuffer 200 rdeepseq

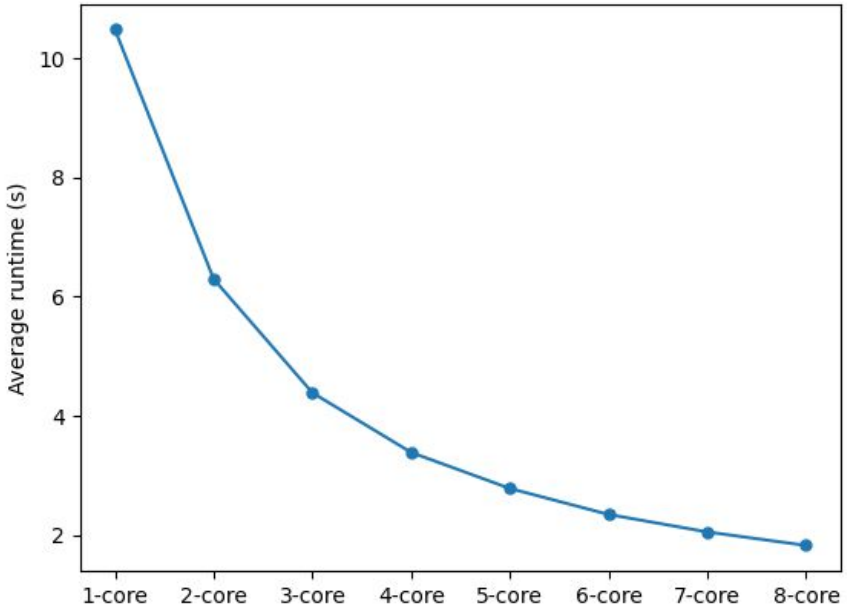
- Finally, parallel for the whole runtime
- Almost all sparks converted
- n = 200 was consistently best in our testing
- Consistent 2.5x speedup in full testing



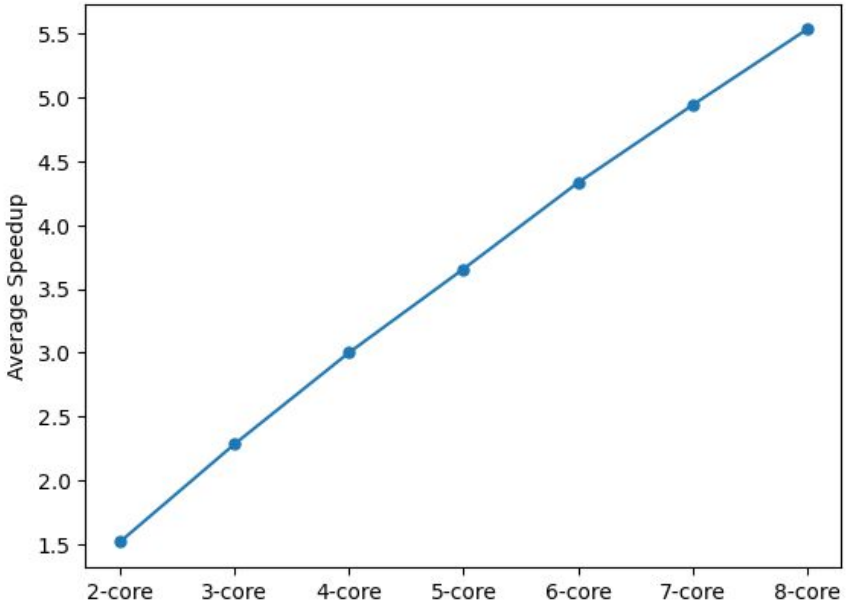
	sequential	parBuffer 100 rdeepseq	parBuffer 200 rdeepseq	parBuffer 300 rdeepseq	parBuffer 400 rdeepseq
“hello”	39.17s	20.58s	20.34s	20.83s	30.17s
“inter”	5.84s	2.66s	2.53s	3.04s	4.22s
“shine”	2.26s	1.22s	1.07s	1.02s	1.16s

parallelization tested on 4 cores

Full testing data (n=500 words)



Average runtime per core



Average speedup of multi-core performance over single core per core

Other strategies that did not work out

- Parallelizing entropy function
  - sparked an unnecessary amount– mostly GC’d and fizzled
- parList rseq/parListChunk rseq on play function
  - sparked too much– slowed parallelization of entropies
- Chunking strategy
  - split answer list into chunks and run play function over those chunks in parallel
  - again, most sparks were GC’d no matter the chunk size