A New Approach To Fizzbuzz

Introduction

- Agent-based modelling ☑
- Parallel computation ☑
- Scheduling algorithms ☑

Claim

- Revolutionary breakthrough in diactic technique

Traditional Fizzbuzz

- Print numbers from 0 to 100
- Multiples of 3: print `fizz` instead
- Multiples of 5: print `buzz` instead
- Multiples of 3 and 5: print `fizzbuzz` instead

Proof

```
Agents = { Counter, Fizzer, Buzzer, Fizzbuzzer }
```

(live demo)

Proof (code)

- Fearless concurrency

```
use std::thread;
use std::time::Duration;

fn main() {
    let counter = thread::spawn(|| {
        let mut i = 0u64;
        loop {
            i = i + 1;
            print!("\n{}", i);
            thread::sleep(Duration::from_secs(1));
        }
    });

    let fizzbuzz = thread::spawn(|| {
        loop {
            thread::sleep(Duration::from_secs(15));
            print!("\rfizzbuzz");
        }
    });

    let buzz = thread::spawn(|| {
        loop {
            thread::sleep(Duration::from_secs(5));
            print!("\rbuzz");
        }
    });

    let fizz = thread::spawn(|| {
        loop {
            thread::sleep(Duration::from_secs(3));
            print!("\rfizz");
        }
    });

    let _ = counter.join();
    let _ = fizzbuzz.join();
    let _ = buzz.join();
    let _ = fizz.join();
}
```

Proof (code)

```
➜  code git:(master) ✗ cargo run --release
   Compiling code v0.1.0
    Finished release [optimized] target(s) in 0.82 secs
     Running `target/release/code`

1
2
fizz
4
buzz
fizz
7
8
fizz
buzz
11
fizz
13
14
fizzbuzz
```

Corollary

- This diactic approach applied to the same class of algorithms

"Genius sorting algorithm"

```
#!/bin/bash

function f() {
    sleep "$1"
    echo "$1"
}
while [ -n "$1" ]
do
    f "$1" &
    shift
done
wait
```

Corollary 1.1 Sleepsort

- O(1) Sleepsort proof

  (live demo)

Q.E.D.

■