

Simple Perl 6 Fractals and Concurrency

Steve Mynott

<https://github.com/stmuk/contalk>

Why?

- Fractals are nice and easy to calculate and look pretty
- But time consuming to run
- Supposed to run well in parallel
- Perl 6 has great support for concurrency
- Use those extra core(s)

Which Fractal?

- The Obvious One!
- Benoit B. Mandelbrot (20 November 1924 – 14 October 2010)



How?

- Perl 6 in “performance enhancement phase”
- So very small size 320x240 (80s retro!)
- On Celeron N2840 (~2.16GHz) took 100 secs
- Plot with SDL (Simple DirectMedia Layer)
- Code tested on Linux and Mac

Code Overview | of 2

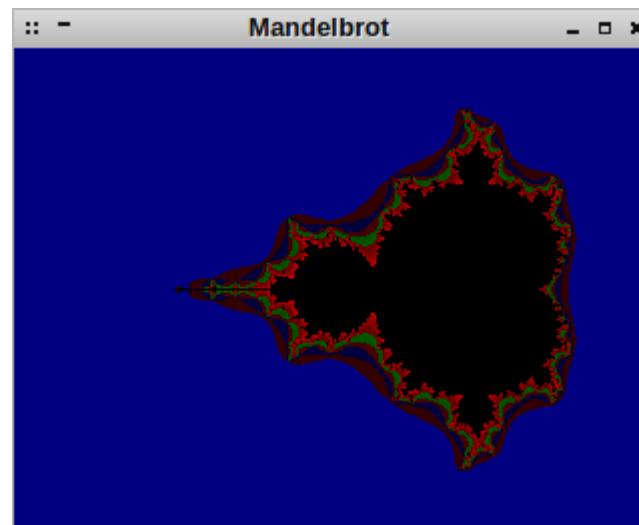
```
loop (my $y = 1; $y >= -1; $y -= 0.01) { # rows
    loop (my $x = -2; $x <= 1; $x += 0.01) { # cols

        my ($res, $i) = mandelbrot($x + $y * i);
        my $color=128;
        $color=10*$i if $i;

        if !$resdefined {
            $color=0;
            SDL_SetRenderDrawColor($render, $color, $color, $color, 0);
            SDL_RenderDrawPoint($render, ($x*($width/4)).Int+200, ($y*($height/4)).Int+120);
        } else {
            if $i < 5 {
                SDL_SetRenderDrawColor($render, 0, 0,128, 0);
            } elsif $i > 5 and $i < 7 {
                SDL_SetRenderDrawColor($render, 0, 0, $color, 0);
            }
            ...
            SDL_RenderDrawPoint($render, ($x*($width/4)).Int+200, ($y*($height/4)).Int+120);
            SDL_RenderPresent($render);
        }
    }
}
```

Code Overview 2 of 2

```
sub mandelbrot(Complex $c) {  
    my $z = $c;  
    for (1 .. 20) -> $i {  
        $z = $z * $z + $c;  
        return ($z, $i) if $z.abs > 2;  
    }  
}
```



Perl 6 Promises

```
my $p = start { say "hi" }
```

```
await $p;
```

Simple Loop

```
for ('a'..'c') -> $l {  
    print "$l";  
    for (1..3) -> $i {  
        print $i  
    }  
}
```

a123b123c123

Async Loop

```
await do for ('a'..'c') -> $l {  
    print "$l";  
    start do for (1..3) -> $i {  
        print $i  
    }  
}
```

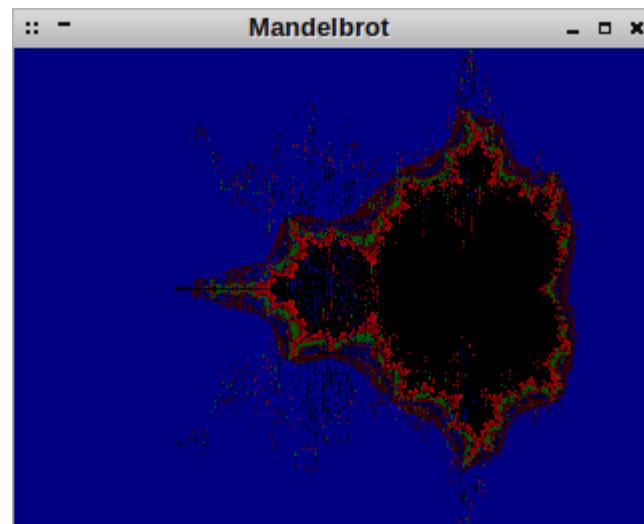
ab123c123123

mbrot2.p6

```
await do for ( 0..$width) -> int $xcoord {  
    start {  
        for ( 0..$height-1) -> int $ycoord {
```

Oh dear

- Took 80% of time (better) but



SDL2 didn't work with threads

The screenshot shows a web browser displaying the **SDL Wiki 2.0**. On the left, there's a sidebar with sections for **Search Wiki**, **Quick Links** (including FrontPage, Introduction, Support, API By Name, API By Category, Contributing, RecentChanges, FindPage, Wiki Help), and **User Actions** (Login). The main content area is titled **FAQ: Development**. It features a **Contents** sidebar with links to various FAQ topics. A red oval highlights the question **Can I call SDL video functions from multiple threads?** which has the answer: "No, most graphics back ends are not thread-safe, so you should only call SDL video functions from the main thread of your application." Below this, another question is visible: **Does SDL support 3D acceleration?**

FAQ: Development

Contents

- FAQ: Development
- Is SDL multi-threaded?
- Can I call SDL video functions from multiple threads?
- Does SDL support 3D acceleration?
- Does SDL support networking?
- Does SDL support PCX, JPG, PNG, etc...
- Does SDL support MP3, Ogg Vorbis, etc...
- Does SDL have text drawing support?
- What kind of GUIs are there for SDL?
- Do I #include <SDL.h> or <SDL/SDL.h>?
- I get the SDL_DUMMY_ENUM assertion in SDL_types.h
- I have an accelerated video card, but SDL tells me that I have zero video memory and no acceleration!
- I'm using SDL_DOUBLEBUF and I still get tearing.
- My mouse cursor disappears when in fullscreen mode!

Is SDL multi-threaded?

SDL provides simple cross-platform functions for the creation of threads and synchronization using mutexes. These are used internally by SDL for some implementations of the audio subsystem and input handling.

Can I call SDL video functions from multiple threads?

No, most graphics back ends are not thread-safe, so you should only call SDL video functions from the main thread of your application.

Does SDL support 3D acceleration?

Mac didn't like it

- Reproducible Kernel panic
- Apple's problem not the Perl 6 Team!

Your computer restarted because of a problem. Press a key or wait a few seconds to continue starting up.

Votre ordinateur a redémarré en raison d'un problème. Pour poursuivre le redémarrage, appuyez sur une touche ou patientez quelques secondes.

El ordenador se ha reiniciado debido a un problema. Para continuar con el arranque, pulse cualquier tecla o espere unos segundos.

Ihr Computer wurde aufgrund eines Problems neu gestartet. Drücken Sie zum Fortfahren eine Taste oder warten Sie einige Sekunden.

問題が起きたためコンピュータを再起動しました。このまま起動する場合は、いずれかのキーを押すか、数秒間そのままお待ちください。

电脑因出现问题而重新启动。请按一下按键，或等几秒钟以继续启动。

What's A Solution?

- <http://doc.perl6.org/language/concurrency>
- Locking!

mbrot3.p6

```
my $l = Lock.new;

await do for ( 0..$width) -> int $xcoord {
    start {
        for ( 0..$height-1) -> int $ycoord {
            ...
            $l.protect(sub{plot($render, $xcoord, $ycoord
$ccolor)});
```

Visible Threading



Locking Works But...

- Image looks fine and its only costs a little more time than not locking
- Not really supposed to use lock due to risk of deadlocks
- Better, more high level solutions in Perl6
- “Channel” – thread safe queue (multiple writers to single reader)

mbrot4.p6 (reader)

```
my $c = Channel.new;

my $plotting = start {
    my $closed = $c.closed;
    loop {
        if $c.poll -> $item {
            my ($xcoord, $ycoord, $color) = $item;
            plot($render, $xcoord, $ycoord, $color);
        }
        elsif $closed {
            last;
        }
    }
}
```

mbrot4.p6 (writers)

```
await do for ( 0..$width) -> int $xcoord {  
  
    start {  
  
        ...  
        my $item = ($xcoord, $ycoord, $color);  
        $c.send($item);  
    }  
}  
  
}  
  
$c.close;  
  
await $plotting;
```

Cheating a little

- Messing with thread scheduler

```
$PROCESS::SCHEDULER=ThreadPoolScheduler.new(initial_threads => 0, max_threads => 2,  
uncaught_handler => Callable);
```

Benchmarks

benchmarking will run 3 iteration(s)

- mbrot1.p6 – non-concurrent 100s
- mbrot2.p6 - broken concurrent 74s
- mbrot3.p6 - concurrent with lock 80s
- mbrot4.p6 - concurrent with channel 112s

Threading is slower!

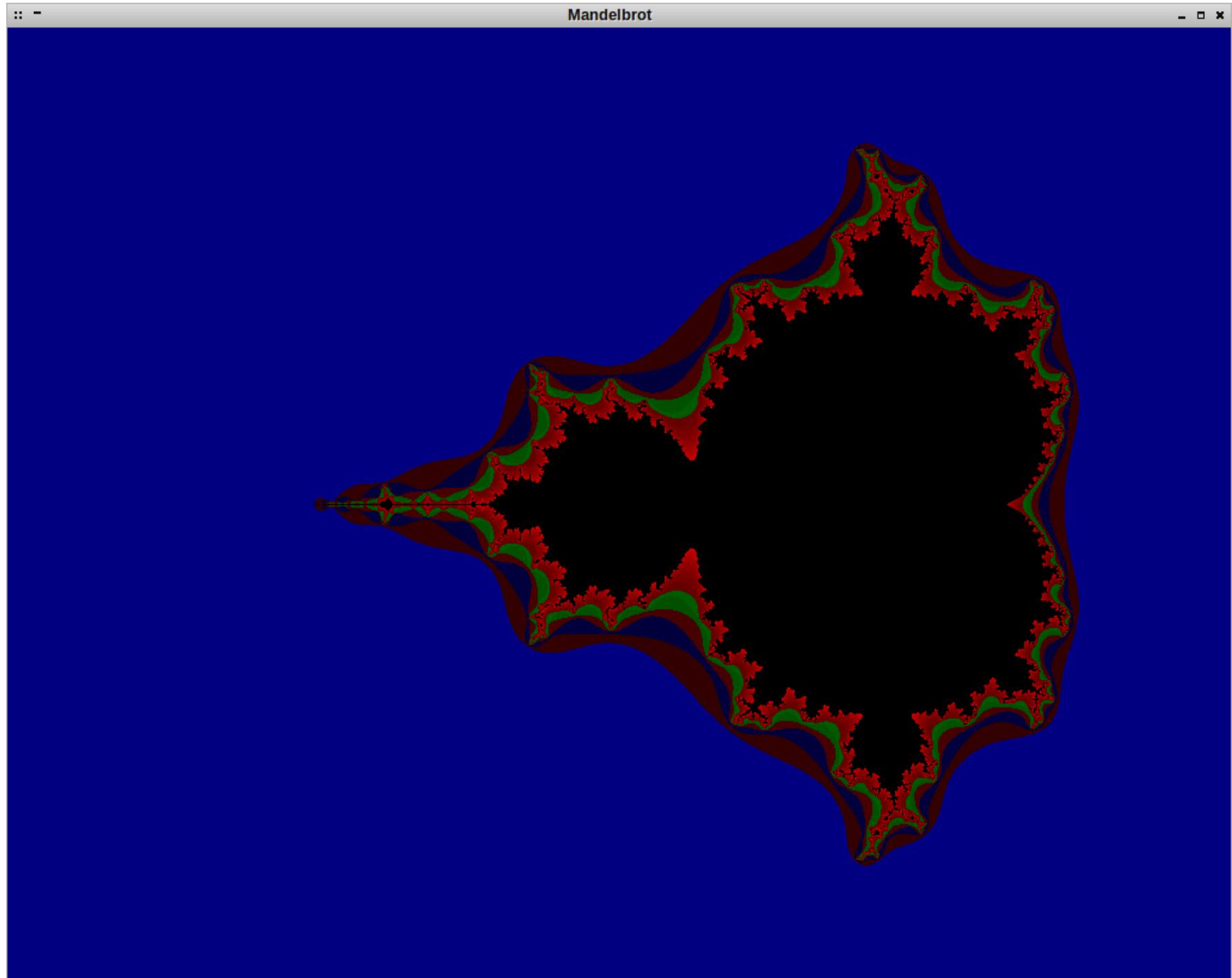
- What to do?
- Freenode/#perl6
- “Threading has a lot of overheads only improves performance with 1000-10,000s of calculation / promise” – moritz
- Increase number of pixels and wait longer

Bigger Mandelbrot!

- 1024x768 (90s retro)
- Use faster computer (2nd Gen i5 2.5Ghz)
- Wait longer

Results

- Standard Mandelbrot – 67 minutes
- Mandelbrot with channel – 62 minutes
- Not hugely different but at least faster!
- Research continuing – watch this space!



Conclusions

- Concurrency in Perl 6 is fairly easy
- It is a good selling point
- Not a “silver bullet”
- Releases this year set to improve performance

<https://github.com/stmuk/contalk>