Turbo Charge CPU Utilization in Fork/Join Using the ManagedBlocker

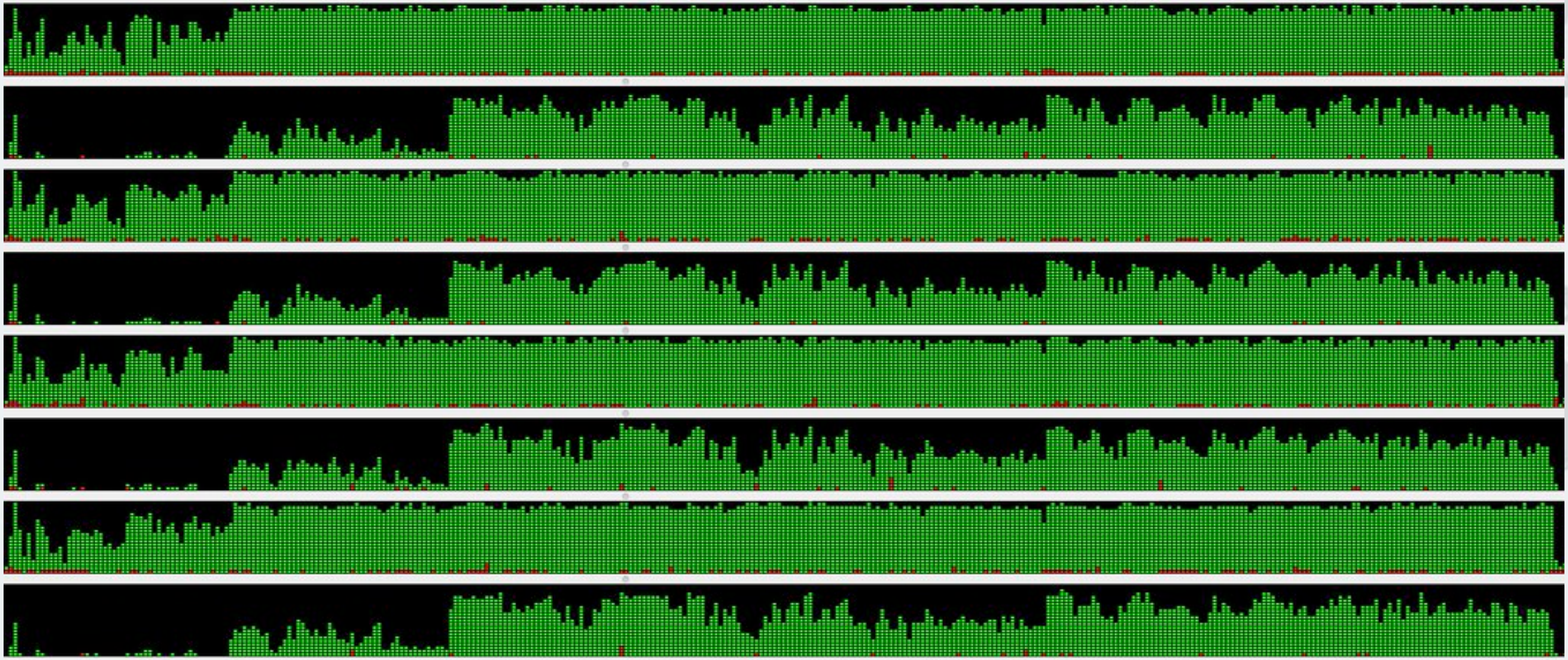
Dr Heinz M. Kabutz

Last Updated 2017-01-24

© 2017 Heinz Kabutz, All Rights Reserved

Javaspecialists.eu
Java Training

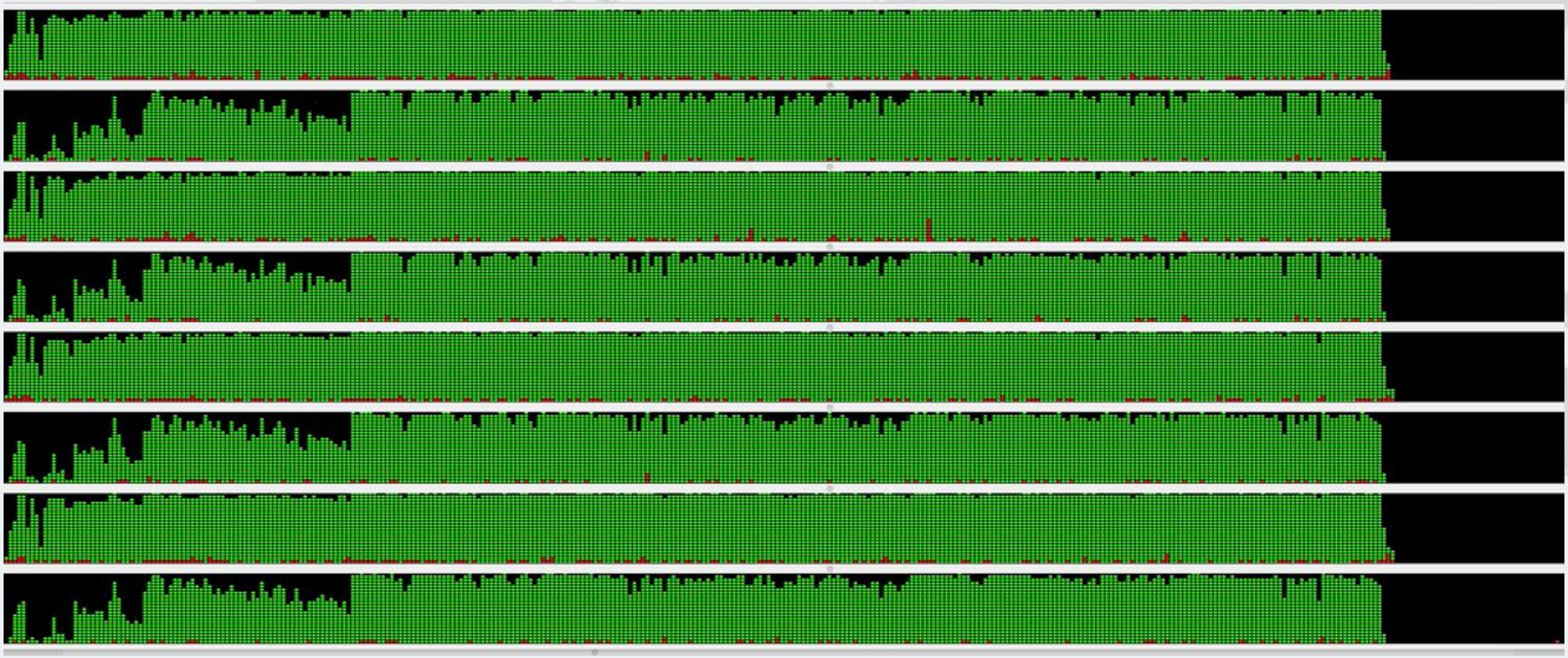
Regular



© 2012-2017 Heinz Kabutz, All Rights Reserved

Javaspecialists.eu

ManagedBlocker

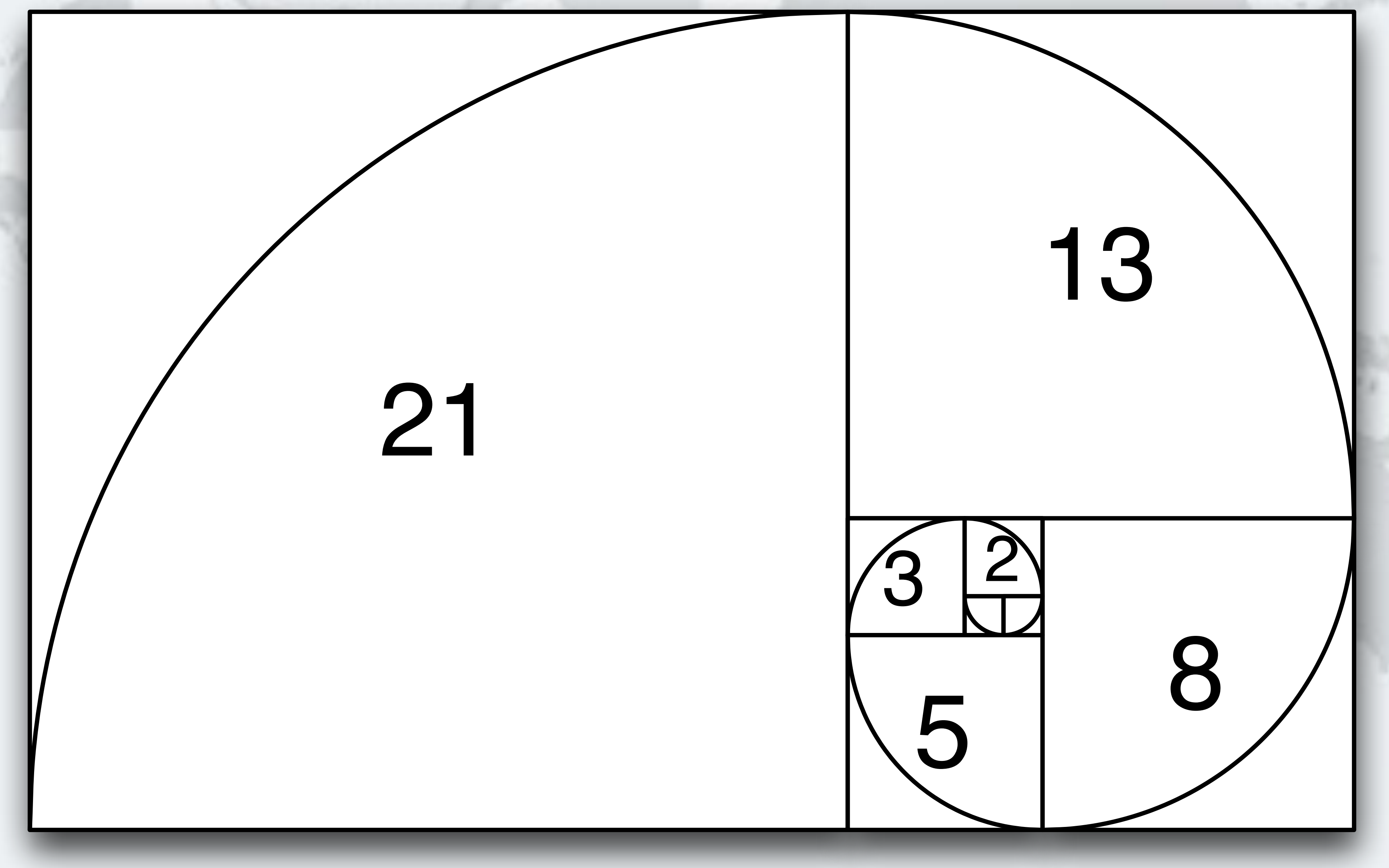


Javaspecialists.eu

© 2012-2017 Heinz Kabutz, All Rights Reserved

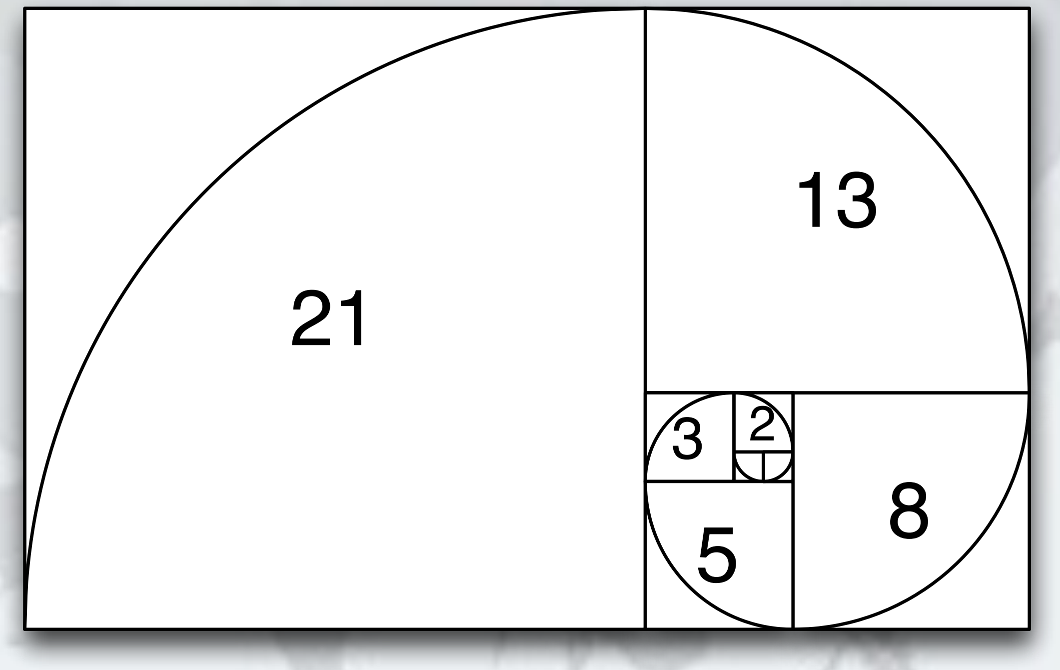
Speeding Up Fibonacci

- **Number sequence named after Leonardo of Pisa**
  - **F0 = 0**
  - **F1 = 1**
  - **Fn = Fn-1 + Fn-2**
- **Thus the next number is equal to the sum of the two previous numbers**
  - **e.g. 0, 1, 1, 2, 3, 5, 8, 13, 21, …**
- **The numbers get large very quickly**



© 2012-2017 Heinz Kabutz, All Rights Reserved

First attempt at writing a Fibonacci Method



- **Taking our recursive definition**
  - **$F_0 = 0$, $F_1 = 1$**
  - **$F_n = F_{n-1} + F_{n-2}$**
- **Our first attempt writes a basic recursive function**

```
public long f(int n) {
  if (n <= 1) return n;
  return f(n-1) + f(n-2);
}
```

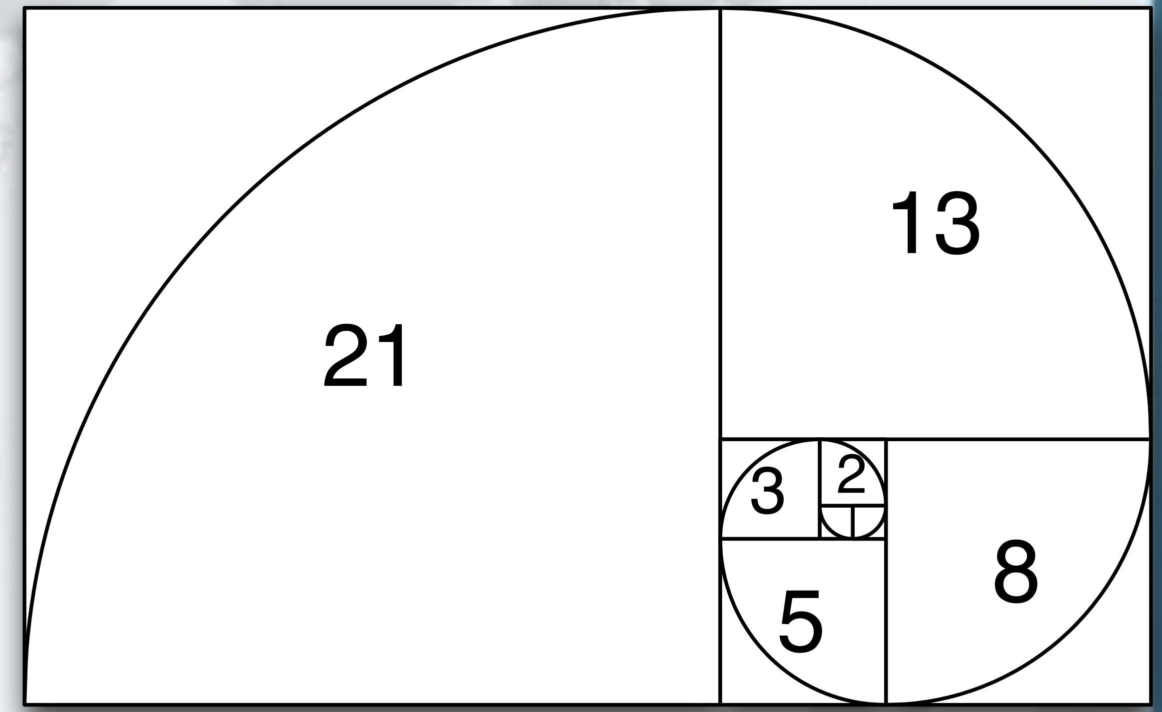
- **But this has exponential time complexity**
  - **f(n+10) is 1000 slower than f(n)**

© 2012-2017 Heinz Kabutz, All Rights Reserved

2nd Attempt at Coding Fibonacci

- **Instead of a recursive method, we use iteration:**

```
public static long f(int n) {
  long n0 = 0, n1 = 1;
  for (int i = 0; i < n; i++) {
    long temp = n1;
    n1 = n1 + n0;
    n0 = temp;
  }
  return n0;
}
```

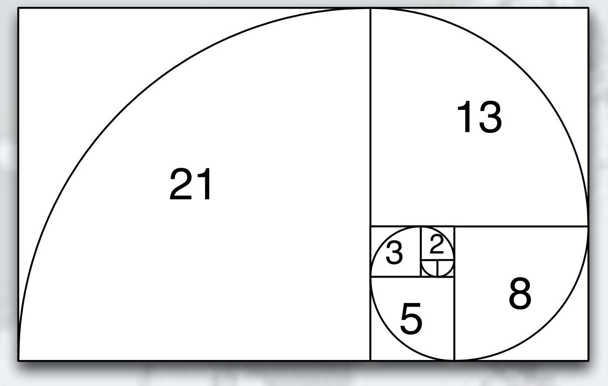


- **This algorithm has linear time complexity**
  - **Solved f(1_000_000_000) in 1.7 seconds**
    - **However, the numbers overflow so the result is incorrect**
    - **We can use BigInteger, but its add() is also linear, so time is quadratic**
    - **We need a better algorithm**

Javaspecialists.eu
© 2012-2017 Heinz Kabutz, All Rights Reserved

3rd Attempt Dijkstra's Sum of Squares

- **Dijkstra noted the following formula for Fibonacci**
  - $F_{2n-1} = F_{n-1}^2 + F_n^2$
  - $F_{2n} = (2 \times F_{n-1} + F_n) \times F_n$



- **Logarithmic time complexity and can be parallelized**
  - **Java 8 uses better BigInteger multiply() algorithms**
    - **Karatsuba complexity is $O(n^{1.585})$**
    - **3-way Toom Cook complexity is $O(n^{1.465})$**
    - **Previous versions of Java had complexity $O(n^2)$**
    - **Unfortunately multiply() in BigInteger is only available single-threaded - we'll fix that later**

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 1: Dijkstra’s Sum of Squares

- **We implement this algorithm using BigInteger**
  - $F_{2n-1} = F_{n-1}^2 + F_n^2$
  - $F_{2n} = (2 \times F_{n-1} + F_n) \times F_n$

© 2012-2017 Heinz Kabutz, All Rights Reserved

Thank You for Listening to me

- **“Heinz’s Happy Hour Recordings”**
  - **Sign up tonight for 12 months free listening worth $239.40**
- **http://tinyurl.com/javada1**

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 2: Parallelize Our algorithm

- **We can parallelize by using common Fork/Join Pool**

```
private final class FibonacciTask extends RecursiveTask<BigInteger> {
  private final int n;
  private FibonacciTask(int n) {
    this.n = n;
  }
  protected BigInteger compute() {
    return f(n);
  }
}
```

- **Next we fork() the 1st task, do the 2nd and then join 1st**

```
FibonacciTask fn_1Task = new FibonacciTask(n - 1);
fn_1Task.fork();
BigInteger fn = f(n);
BigInteger fn_1 = fn_1Task.join();
```

Javaspecialists.eu

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 3: Parallelize BigInteger

- **Using principles from demo 2, we now parallelize methods in eu.javaspecialists.performance.math.BigInteger**
  - **multiplyKaratsuba()**
  - **multiplyToomCook3()**
  - **squareKaratsuba()**
  - **squareToomCook3()**

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 4: Cache Results

- **Dijkstra's Sum of Squares needs to work out some values several times. Cache results to avoid this.**

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 5: Reserved Caching Scheme

- **We make sure we implement a “reserved caching scheme” where if one thread says he wants to calculate some value, others would wait**
  - **e.g. have a special BigInteger that signifies RESERVED**
    - **First thing a task would do is check if map contains that**
    - **If it doesn’t, it puts it in and thus reserves it**
    - **If it does, it waits until the task is done and uses that value**

Javaspecialists.eu

© 2012-2017 Heinz Kabutz, All Rights Reserved

Demo 6: ManagedBlocker

- **ForkJoinPool is configured with *desired parallelism***
  - **Number of active threads**
  - **ForkJoinPool mostly used with CPU intensive tasks**
- **If one of the FJ Threads has to block, a new thread can be started to take its place**
  - **This is done with the ManagedBlocker**
- **We use ManagedBlocker to keep parallelism high**

© 2012-2017 Heinz Kabutz, All Rights Reserved

Heinz Kabutz

heinz@kabutz.net

http://tinyurl.com/javada1

© 2017 Heinz Kabutz, All Rights Reserved