#1BRC - THE FIRST 80%

How to Read One Billion Rows Fast Enough

René Schwietzke, Xceptance GmbH

ABOUT RENÉ SCHWIETZKE

- Master of Computer Science (Dipl.-Inf.)
- Programmer since 1989
- Java since Java 1.0, 1996
- QA and Testing since 1998
- Performance Tester since 1999
- Co-Founder of XCEPTANCE
- @ReneSchwietzke
- @reneschwietzke@foojay.social

#java #performance #tuning #qa #test #performancetest #loadtest

ABOUT XCEPTANCE

- Founded 2004
- Headquarters in Jena, Germany; Subsidiary in Cambridge, MA, USA
- Specialized in Software Testing and Quality Assurance
- Performance testing since 2004
- Over 150 performance test projects every year
- World-wide customer base including APAC and South America
- Performance Test Tool XLT®, Java-based, APL 2.0

github.com/Xceptance/XLT

LICENSE

I care and share

#1BRC - INTRODUCTION

What is 1BRC and Why the Fuzz?

THE ORIGIN

A busy January 2024 for Java Devs

- Gunnar Morling got bored during the holidays
- His wife suggested to do something nice and fun
- So he asked the community the #1brc question

... write a Java program for retrieving temperature measurement values from a text file and calculate the min, mean, and max temperatures per weather station.
... one caveat, the file has 1,000,000,000 rows!

https://github.com/gunnarmorling/1brc

DATA

Input

```
Zagreb;19.7
Havana;36.2
Mexicali;27.3
Marseille;15.4
Banjul;24.2
Canberra;0.8
Mandalay;19.3
Bridgetown;29.6
Chihuahua;10.9
```

Output

```
# This is TreeMap::toString
# Data shown here is formatted for screen
{
   Abha=5.0/18.0/27.4, Abidjan=15.7/26.0/34.1,
   Abéché=12.1/29.4/35.6, Accra=14.7/26.4/33.1,
   Addis Ababa=2.1/16.0/24.3, Adelaide=4.1/17.3/29.7,
   ...
   Willemstad=-7.9,28.2,57.7, Winnipeg=-28.1,3.0,39.4,
   Wrocław=-27.5,9.6,50.2, Xi'an=-22.5,14.0,49.1,
```

```
Honiara;35.3
Gangtok;-1.7
Houston;32.4
Zürich;13.8
Kolkata;20.6
Tripoli;8.9
Karachi;28.3
Zagreb;13.2
Palm Springs;20.3
Lhasa;6.2
Sacramento;28.3
Edmonton;5.6
Toluca;14.0
El Paso;24.0
Antsiranana;24.9
Tromsø;-3.9
```

```
    Yakutsk=-42.2,-8.8,26.4, Yangon=-2.6,27.6,59.0,
    Yaoundé=-10.1,23.6,58.3, Yellowknife=-38.4,-4.0,40.0,
    Yerevan=-18.4,12.4,49.6, Yinchuan=-25.4,9.5,42.0,
    Zagreb=-25.3,10.6,48.4, Zanzibar City=-5.0,26.1,61.2,
    Zürich=-20.9,9.4,43.5, Ürümqi=-26.8,7.4,43.7,
    İzmir=-15.8,18.0,50.1
}
```

RULES

- Java Only
- JDKs via SDKMAN only, any version!
- Any CLI parameters permitted
- No external dependencies
- Results measured on bare metal
- File in RAM (RAMdisk or file cache)
- No preprocessing of the data permitted
- Max 10k stations - initial version had 413
- Only valid data in the file, UTF-8
- Line ending is `\n`
- City names max 100 bytes (unicode) long
- Temperatures -99.9 to 99.9 (e.g. 5.0, 0.0)
- IEEE 754 rounding-direction "roundTowardPositive"
- Five measurements, middle three averaged

THE RESULTS

Pretty unbelievable

Test	Cores	Time

Baseline	1	200.7 s
Baseline	8	99.8 s
Thomas Wuerthinger	1	8.9 s
Thomas Wuerthinger	8	1.2 s
Today's Goal	1	40.0 s

- Digital Ocean, Amsterdam
- Dedicated Machine, CPU-optimized
- 16x Intel Xeon Platinum 8358 CPU @ 2.60GHz
- Cache 16x 4 MB
- Memory 32 GB

LET'S CHALLENGE OURSELVES

What we will do and what we won't

SOME UPDATED RULES

Just to Make Things Easier

- Permit 1M, 10M, 100M and 1000M files
- 10M is 1M plus new 9M and so on
- Measure without process startup
- Create small simple Benchmark framework
- One Simple Benchmark Framework - 1SBF
- Supports warmup and measurements rounds
- Calculates mean of measurements
- New instance of test class for each round
- SHA-256 of result to discover differences
- Supports batchmode

github.com/rschwietzke/1brc-the-first-80-meters.git

LIMITATIONS

Some limitations apply today

- 50 min are not enough
- 80% reach doesn't require all tricks
- Über-code becomes hard to understand
- Real world gains vary a lot

Goal: All code is still easily understandable.

- No Unsafe
- No Vector API
- No Graal
- No Bit-Magic
- No Branchless Code
- No Memory-Mapped Files
- No JNI/Panama

github.com/rschwietzke/1brc-the-first-80-meters.git

THE BASELINE

The original code

```
public String run(final String fileName) throws IOException {
    Collector<Measurement, MeasurementAggregator, ResultRow> collector = Collector.of(MeasurementAggregator::new,
            (agg, m) -> {
                agg.min = Math.min(agg.min, m.value);
                agg.max = Math.max(agg.max, m.value);
                agg.sum += m.value;
                agg.count++;
            }, (agg1, agg2) -> {
                var res = new MeasurementAggregator();
                res.min = Math.min(agg1.min, agg2.min);
                res.max = Math.max(agg1.max, agg2.max);
                res.sum = agg1.sum + agg2.sum;
                res.count = agg1.count + agg2.count;

                return res;
            }, agg -> {
                return new ResultRow(agg.min, (Math.round(agg.sum * 10.0) / 10.0) / agg.count, agg.max);
            });

    var result = Files.lines(Paths.get(fileName)).parallel() // <-- optional
            .map(l -> l.split(";"))
            .map(l -> new Measurement(l))
            .collect(groupingBy(m -> m.station(), collector));
```

```
    return new TreeMap<>(result).toString();
}
```

org.rschwietzke.devoxxpl24.BRC01_BaselineST

github.com/rschwietzke/1brc-the-first-80-meters.git

THE INSIGHTS

BRC01_BaselineST

- Java Streams
- `File.lines` for IO
- `String::split` for CSV
- `Double::parseDouble(String)`
- `groupingBy`
- Records as intermediate storage

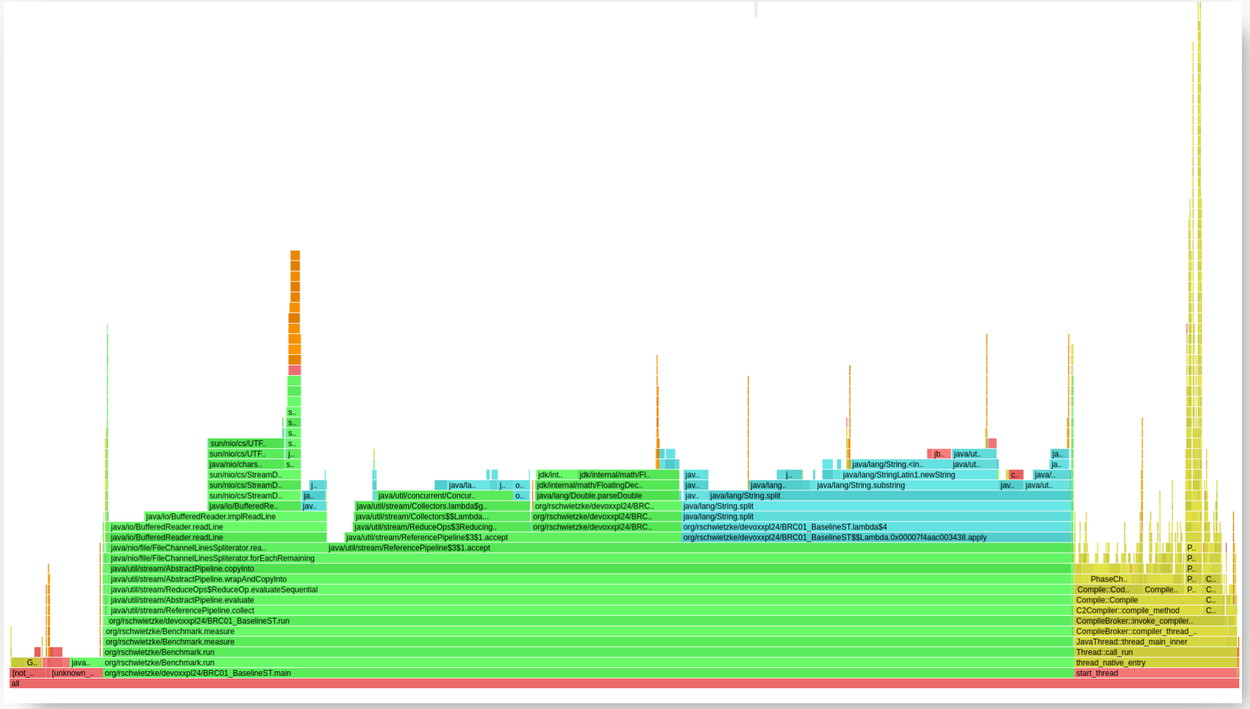
THE RESULTS

Test	Cores	Time
Baseline	1	205.0 s
Baseline	8	96.8 s

Numbers between slides might vary.

github.com/rschwietzke/1brc-the-first-80-meters.git

FLAMEGRAPH



BRC01_BaselineST CPU Flamegraph

run	78.4 %
readLine	17.7 %
split	31.7 %
parseDouble	11.7 %
store/reduce	14.4 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC01_BaselineST measurements.txt 2 2 \
```

github.com/rschwietzke/1brc-the-first-80-meters.git

THE SECOND BASELINE

First is too complicated for tuning

```java
/**
 * Our non-stream version
 */
public String run(final String fileName) throws IOException
{
    final Map<String, Temperatures> cities = new HashMap<>();

    try (var reader = Files.newBufferedReader(Paths.get(fileName)))
    {
        String line;
        while ((line = reader.readLine()) != null)
        {
            // split the line
            final String[] cols = line.split(";");

            // get us the data to store
            final String city = cols[0];
            final double temperature = Double.parseDouble(cols[1]);

            // store and sum up the data
            cities.merge(city, new Temperatures(temperature), (t1, t2) -> t1.merge(t2));
        }
    }
```

```
    // ok, we got everything, now we need to order it
    return new TreeMap<String, Temperatures>(cities).toString();
}
```

org.rschwietzke.devoxxpl24.BRC03_NoStreamST

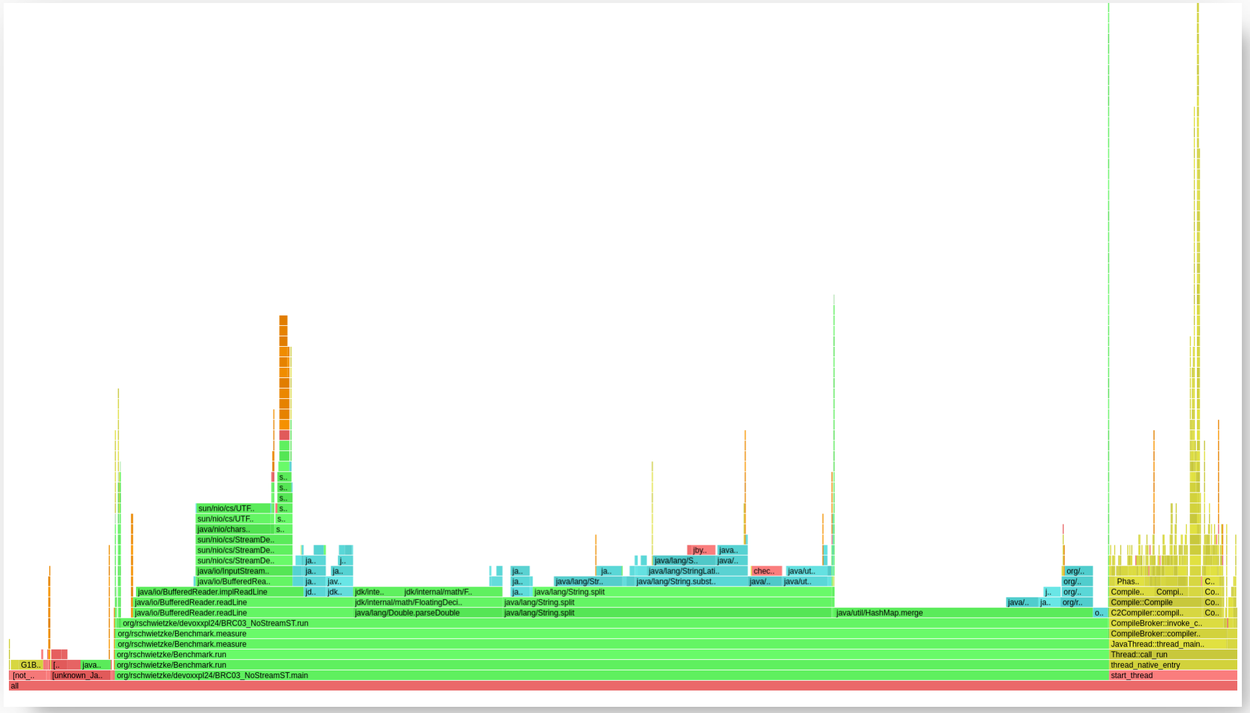
github.com/rschwietzke/1brc-the-first-80-meters.git

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %

github.com/rschwietzke/1brc-the-first-80-meters.git

FLAMEGRAPH



BRC03_NoStreamST CPU Flamegraph

Area	01	03
run	**78.4 %**	**80.4 %**
readLine	17.7 %	17.9 %
split	31.7 %	26.8 %
parseDouble	11.7 %	12.2 %
merge	14.4 %	21.0 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC03_NoStreamST measurements.txt 2 2 \
```

github.com/rschwietzke/1brc-the-first-80-meters.git

SPLIT IS SLOW

Get rid off `split`

```
public String run(final String fileName) throws IOException
{
    // our cities with temperatures
    final Map<String, Temperatures> cities = new HashMap<>();

    try (var reader = Files.newBufferedReader(Paths.get(fileName)))
    {
        String line;
        while ((line = reader.readLine()) != null)
        {
            // split the line
            final int pos = line.indexOf(';');

            // get us the city
            final String city = line.substring(0, pos);
            final String temperatureAsString = line.substring(pos + 1);

            // parse our temperature
            final double temperature = Double.parseDouble(temperatureAsString);

            // merge the data into the captured data
            cities.merge(city, new Temperatures(temperature), (t1, t2) -> t1.merge(t2));
        }
```

```
        }

        // ok, we got everything, now we need to order it and print it
        return new TreeMap<String, Temperatures>(cities).toString();
    }
```

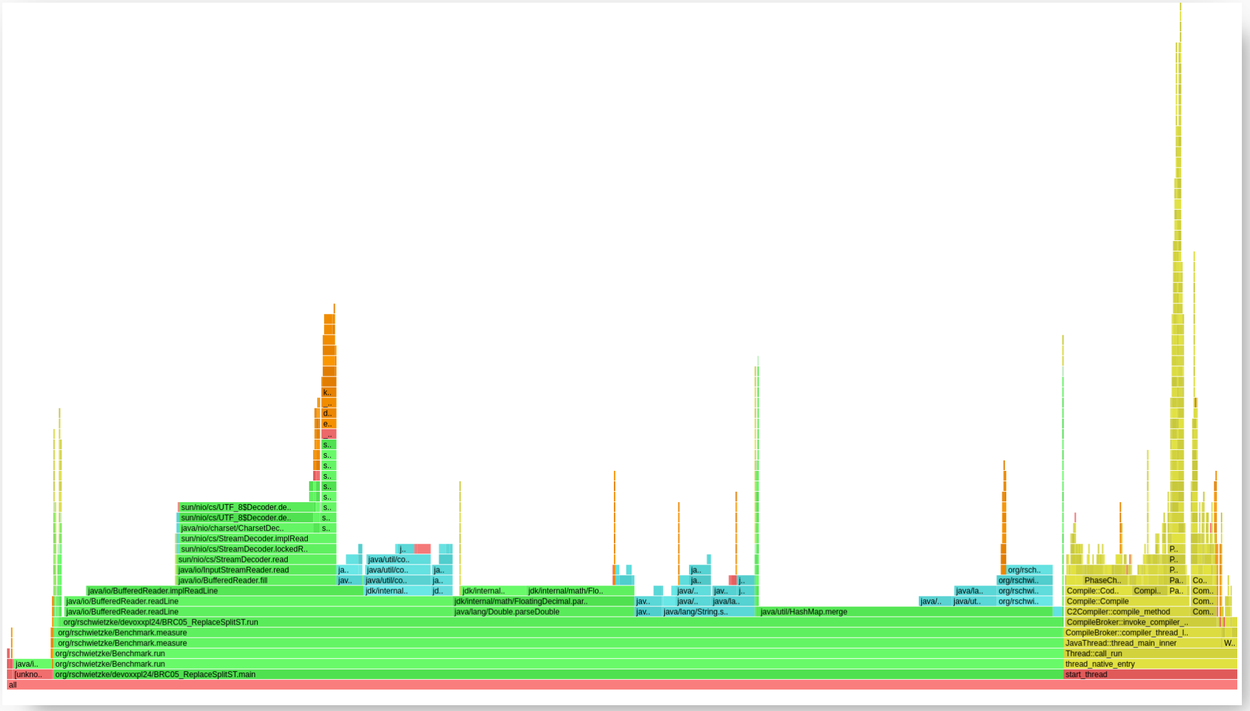
org.rschwietzke.devoxxpl24.BRC05_ReplaceSplitST

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %

github.com/rschwietzke/1brc-the-first-80-meters.git

FLAMEGRAPH



BRC05_ReplaceSplitST CPU Flamegraph

Area	01	03	05
run	**78.4 %**	**80.4 %**	**81.3 %**
readLine	17.7 %	17.9 %	31.6 %
split	31.7 %	26.8 %	n/a
indexOf	n/a	n/a	2.2 %
subString	n/a	9.2 %	7.6 %
parseDouble	11.7 %	12.2 %	14.8 %
merge	14.4 %	21.0 %	23.9 %

github.com/rschwietzke/1brc-the-first-80-meters.git

DOUBLE PARSING

Make double parsing less generic

```
public static double parseDouble(final String s) {
    return parseDouble(s, 0, s.length() - 1);
}

private static final double[] multipliers = {
    1, 1, 0.1, 0.01, 0.001, 0.000_1, 0.000_01, 0.000_001, 0.000_000_1, 0.000_000_01,
    0.000_000_001, 0.000_000_000_1, 0.000_000_000_01, 0.000_000_000_001, 0.000_000_000_000_1,
    0.000_000_000_000_01, 0.000_000_000_000_001, 0.000_000_000_000_000_1, 0.000_000_000_000_000_01};

public static double parseDouble(final String s, final int offset, final int end) {
    final int negative = s.charAt(offset) == '-' ? offset + 1 : offset;

    long value = 0;
    int decimalPos = end;

    for (int i = negative; i <= end; i++) {
        final int d = s.charAt(i);
        if (d == '.') {
            decimalPos = i;
            continue;
        }
        final int v = d - DIGITOFFSET;
        value = ((value << 3) + (value << 1));
```

```
            value += v;
        }

        // adjust the decimal places
        value = negative != offset ? -value : value;
        return value * multipliers[end - decimalPos + 1];
    }
```

org.rschwietzke.devoxxpl24.BRC06_NewDoubleParsingST

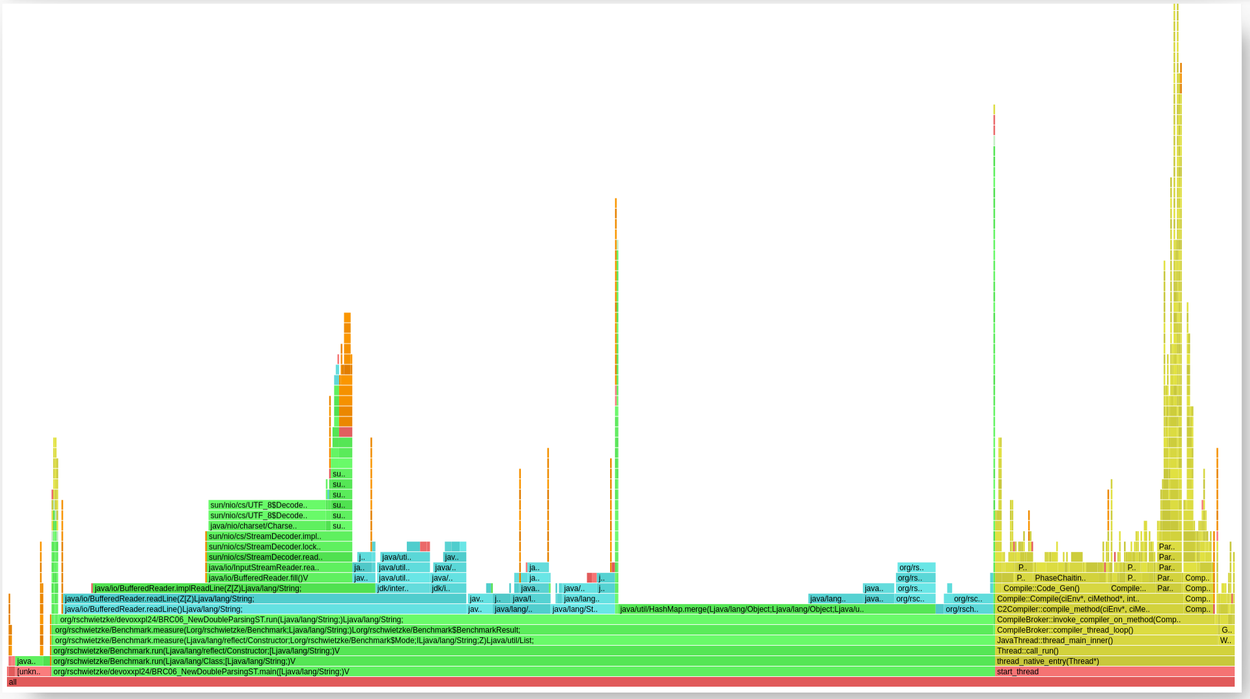
github.com/rschwietzke/1brc-the-first-80-meters.git

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %

github.com/rschwietzke/1brc-the-first-80-meters.git

FLAMEGRAPH



BRC06_NewDoubleParsingST CPU Flamegraph

Area	05	06
run	**81.3 %**	**76.2 %**
readLine	31.6 %	32.8 %
indexOf	2.2 %	2.2 %
subString	7.6 %	9.9 %
parseDouble	14.8 %	4.4 %
merge	23.9 %	25.8 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC06_NewDoubleParsingST measurements.txt 2 2 \
```

SUBSTRING

One less String

```
while ((line = reader.readLine()) != null)
{
    // split the line
    final int pos = line.indexOf(';');

    // get us the city
    final String city = line.substring(0, pos);

    // parse our temperature inline without an instance of a string for temperature
    final double temperature = ParseDouble.parseDouble(line, pos + 1, line.length() - 1);

    // merge the data into the captured data
    cities.merge(city, new Temperatures(temperature), (t1, t2) -> t1.merge(t2));
}
```

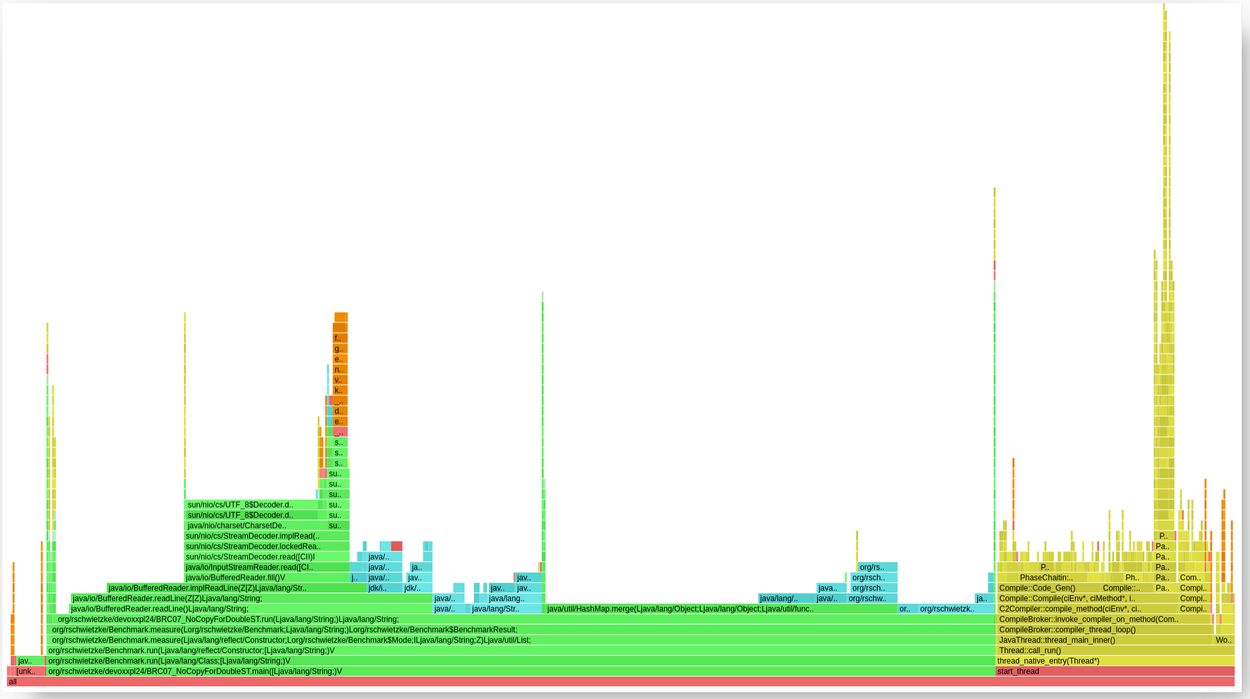
org.rschwietzke.devoxxpl24.BRC07_NoCopyForDoubleST

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %

github.com/rschwietzke/1brc-the-first-80-meters.git

FLAMEGRAPH



BRC07_NoCopyForDoubleST CPU Flamegraph

Area	05	06	07
run	**81.3 %**	**76.2 %**	**76.4 %**
readLine	31.6 %	32.8 %	29.6 %
indexOf	2.2 %	2.2 %	2.6 %
subString	7.6 %	9.9 %	5.8 %
parseDouble	14.8 %	4.4 %	6.0 %
merge	23.9 %	25.8 %	28.7 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC07_NoCopyForDoubleST measurements.txt 2 2 \
```

---

github.com/rschwietzke/1brc-the-first-80-meters.git

DOUBLE IS DOUBLE THE WORK

Ignore double as ~~long~~int as possible

```
/**
 * Parses a double but ends up with an int, only because we know
 * the format of the results -99.9 to 99.9
 */
public static int parseInteger(final String s, final int offset, final int end)
{
    final int negative = s.charAt(offset) == '-' ? offset + 1 : offset;

    int value = 0;

    for (int i = negative; i <= end; i++)
    {
        final int d = s.charAt(i);
        if (d == '.')
        {
            continue;
        }
        final int v = d - DIGITOFFSET;
        value = ((value << 3) + (value << 1));
        value += v;
    }

    value = negative != offset ? -value : value;
```

```
    return value;
}
```

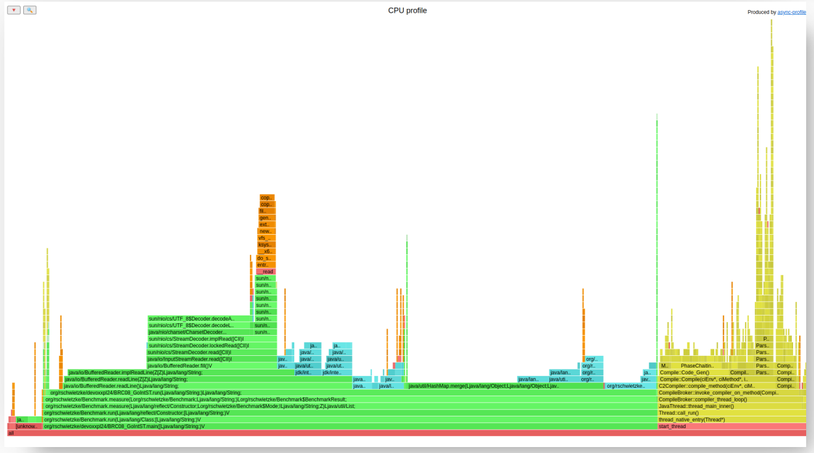
org.rschwietzke.devoxxpl24.BRC08_GoIntST

THE RESULTS

Measured on Digital Ocean

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %

FLAMEGRAPH



CPU Flamegraph

Area	05	06	07	08
run	**81.3 %**	**76.2 %**	**76.4 %**	**79.3 %**
readLine	31.6 %	32.8 %	29.6 %	34.4 %
indexOf	2.2 %	2.2 %	2.6 %	2.5 %
subString	7.6 %	9.9 %	5.8 %	6.0 %
parseDouble	14.8 %	4.4 %	6.0 %	8.2 %
merge	23.9 %	25.8 %	28.7 %	24.2 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC08_GoIntST measurements.txt 2 2 \
```

REMOVE LAMBDA

Explicit Merging

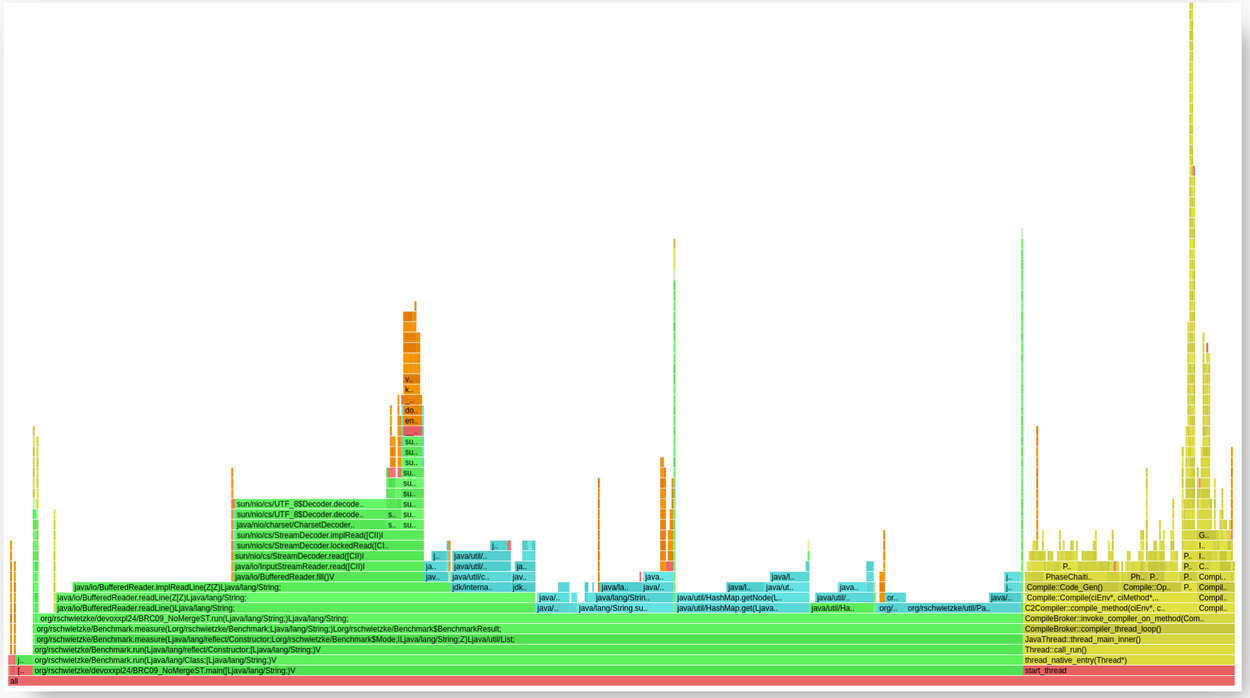
```
while ((line = reader.readLine()) != null)
{
    // split the line
    final int pos = line.indexOf(';');

    // get us the city
    final String city = line.substring(0, pos);

    // parse our temperature inline without an instance of a string for temperature
    final int temperature = ParseDouble.parseInteger(line, pos + 1, line.length() - 1);

    // get city and update
    final var v = cities.get(city);
    final var t = new Temperatures(temperature);
    cities.put(city, v != null ? v.merge(t) : t);
}
```

org.rschwietzke.devoxxpl24.BRC09_NoMergeST

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %

FLAMEGRAPH



CPU Flamegraph

run	80.1 %
readLine	39.1 %
indexOf	2.8 %
substring	7.9 %
parseInteger	6.3 %
get	10.9 %
merge	2.3 %
put	5.2 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC09_NoMergeST measurements.txt 2 2 \
```

MUTABILITY TO THE RESCUE

MUTABILITY TO THE RESCUE

Less Memory Churn

```
while ((line = reader.readLine()) != null)
{
    ...
    // get city
    final Temperatures v = cities.get(city);
    if (v != null)
    {
        // know it, put both together
        v.add(temperature);
    }
    else
    {
        // we have not seen that city yet, create a container and store it
        cities.put(city, new Temperatures(temperature));
    }
}
```

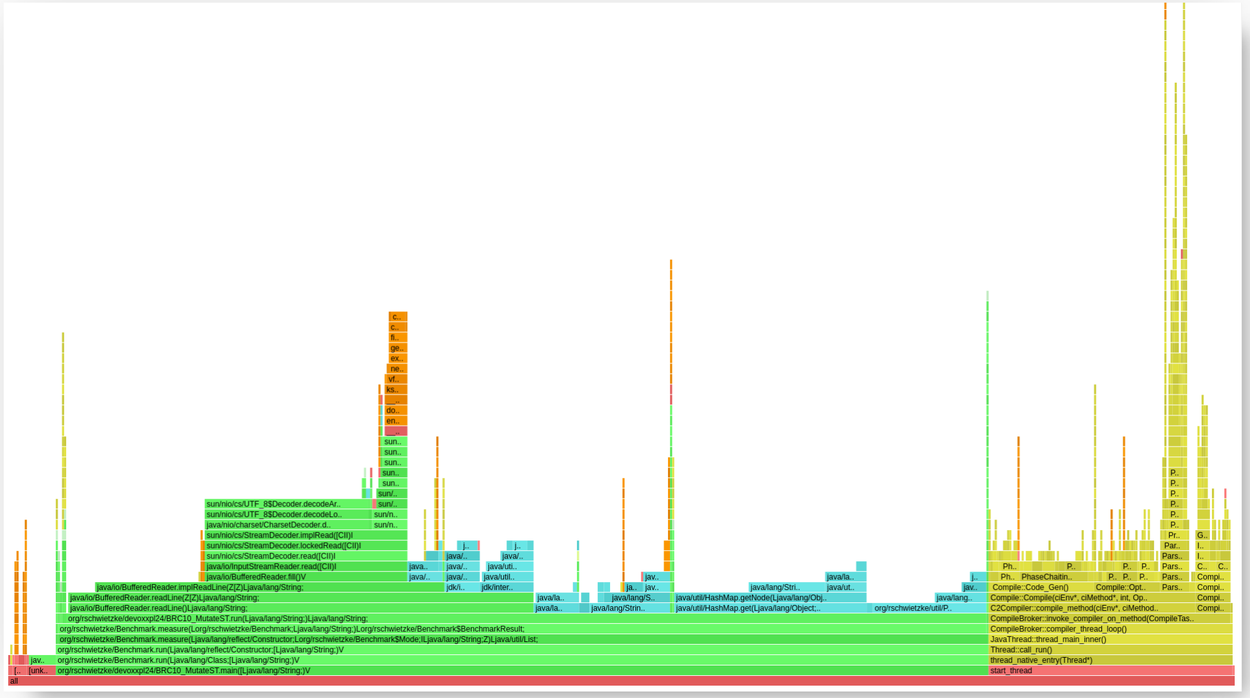
org.rschwietzke.devoxxpl24.BRC10_MutateST

THE RESULTS

Test	Time	%

BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %

FLAMEGRAPH



CPU Flamegraph

Area	09	10
run	80.3 %	71.3 %
readLine	39.1 %	38.1 %
indexOf	2.8 %	2.4 %
substring	7.9 %	6.2 %
parseInteger	6.3 %	6.6 %
get	10.9 %	15.7 %
merge	2.3 %	0.0 %
put	5.2 %	0.0 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC10_MutateST measurements.txt 2 2 \
```

OPEN HASHING

OPEN HASHING

Simplify the Map

```
// https://github.com/mikvor/hashmapTest/blob/master/src/main/java/map/objobj/ObjObjMap.java
public class FastHashMap<K, V> {
    private static final Object FREE_KEY = new Object();
    private Object[] m_data;
...
    public V get( final K key ) {
        int ptr = (key.hashCode() & m_mask) << 1;
        Object k = m_data[ ptr ];

        if ( k == FREE_KEY ) {
            return null;  //end of chain already
        }

        if ( k.hashCode() == key.hashCode() && k.equals( key ) ) {
            return (V) m_data[ ptr + 1 ];
        }

        while ( true ) {
            ptr = (ptr + 2) & m_mask2; //that's next index
            k = m_data[ ptr ];
            if ( k == FREE_KEY ) {
                return null;
            }
            if (k.equals( key )) {
                return (V) m_data[ ptr + 1 ];
            }
        }
    }
```

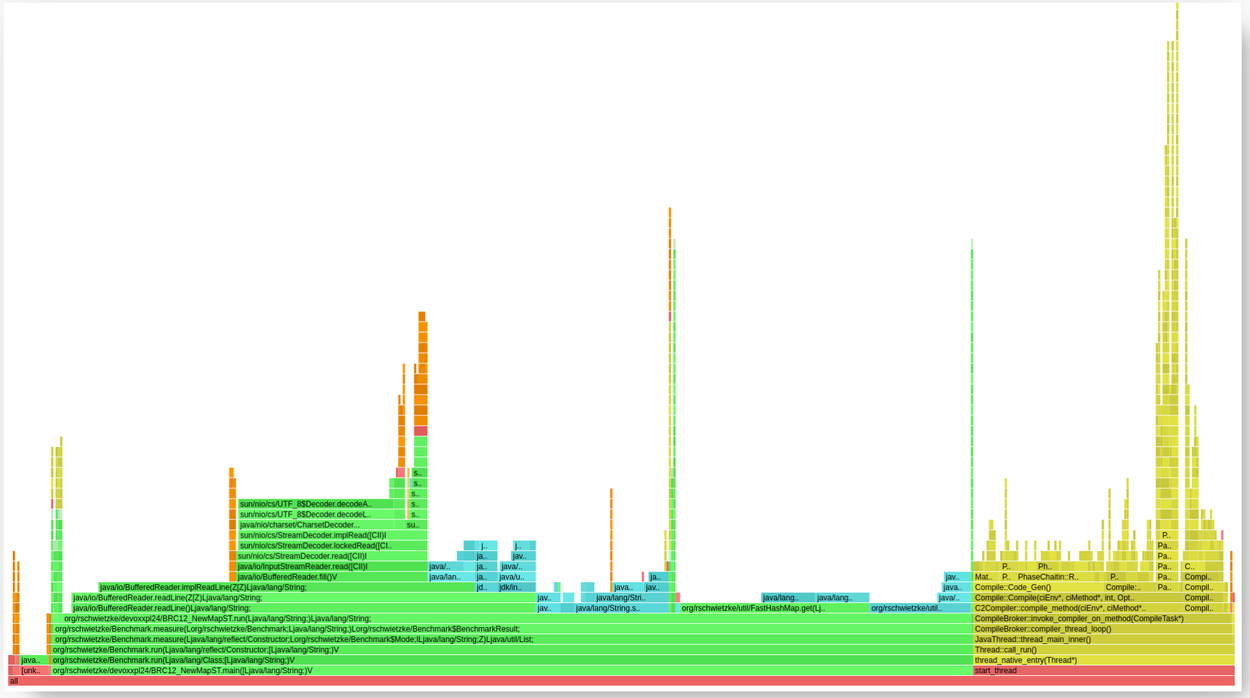
```
...
}
```

org.rschwietzke.devoxxpl24.BRC12_NewMapST

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %
BRC12_NewMapST	105.0 s	47.9 %

FLAMEGRAPH



CPU Flamegraph

run	74.1 %
readLine	37.9 %
indexOf	2.0 %
substring	7.7 %
parseInteger	8.3 %
get	15.4 %
merge	0.0 %
put	0.0 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC12_NewMapST measurements.txt 2 2 \
```

SET INSTEAD OF MAP

SET INSTEAD OF MAP

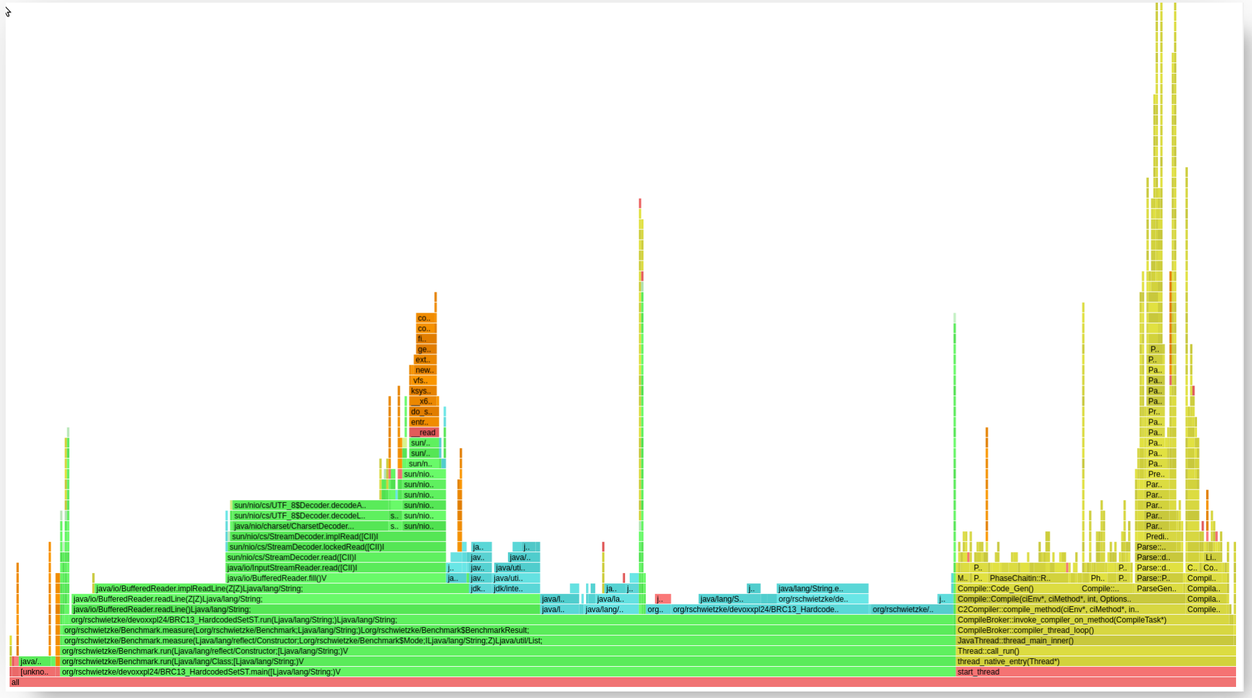
Eliminate Slots

```
public void getPutOrUpdate( final String city, int value ) {
    final int hash = city.hashCode();
    int ptr = hash & m_mask;
    Temperatures k = m_data[ ptr ];

    if ( k == FREE_KEY ) {
        put(new Temperatures(city, value));
        return;
    }

    if ( k.equals( city ) ) {
        k.add(value);
        return;
    }

    while ( true ) {
        ptr = (ptr + 1) & m_mask; //that's next index
        k = m_data[ ptr ];
        if ( k == FREE_KEY ) {
            put(new Temperatures(city, value));
            return;
        }
        if ( k.equals( city ) ) {
            k.add(value);
            return;
        }
    }
}
```

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %
BRC12_NewMapST	105.0 s	47.9 %
BRC13_HardcodedSetST	102.5 s	46.8 %

FLAMEGRAPH



CPU Flamegraph

run	72.1 %
readLine	38.2 %
indexOf	3.2 %
substring	4.5 %
parseInteger	6.7 %
putOrUpdate	16.1 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC13_HardcodedSetST measurements.txt 2 2 \
```

HEAVY LIFTING

HEAVY LIFTING

Approach IO Differently

```
try (var raf = new RandomAccessFile(fileName, "r");
     var channel = raf.getChannel();)
{
    final Line line = new Line(channel);

    while (true)
    {
        line.readFromChannel();

        if (line.hasNewLine)
        {
            // parse our temperature inline without an instance of a string for temperature
            final int temperature = ParseDouble.parseInteger(line.data, line.semicolonPos + 1, line.newlinePos - 1);

            // find and update
            cities.getPutOrUpdate(line, temperature);
        }
        if (line.EOF)
        {
            break;
        }
    }
}
```

---

org.rschwietzke.devoxxpl24.BRC14_ReadBytesST

```
private void readFromChannel()
{
    hashCode = -1;
```

```
    lineStartPos = pos;

    // look for semicolon and new line
```

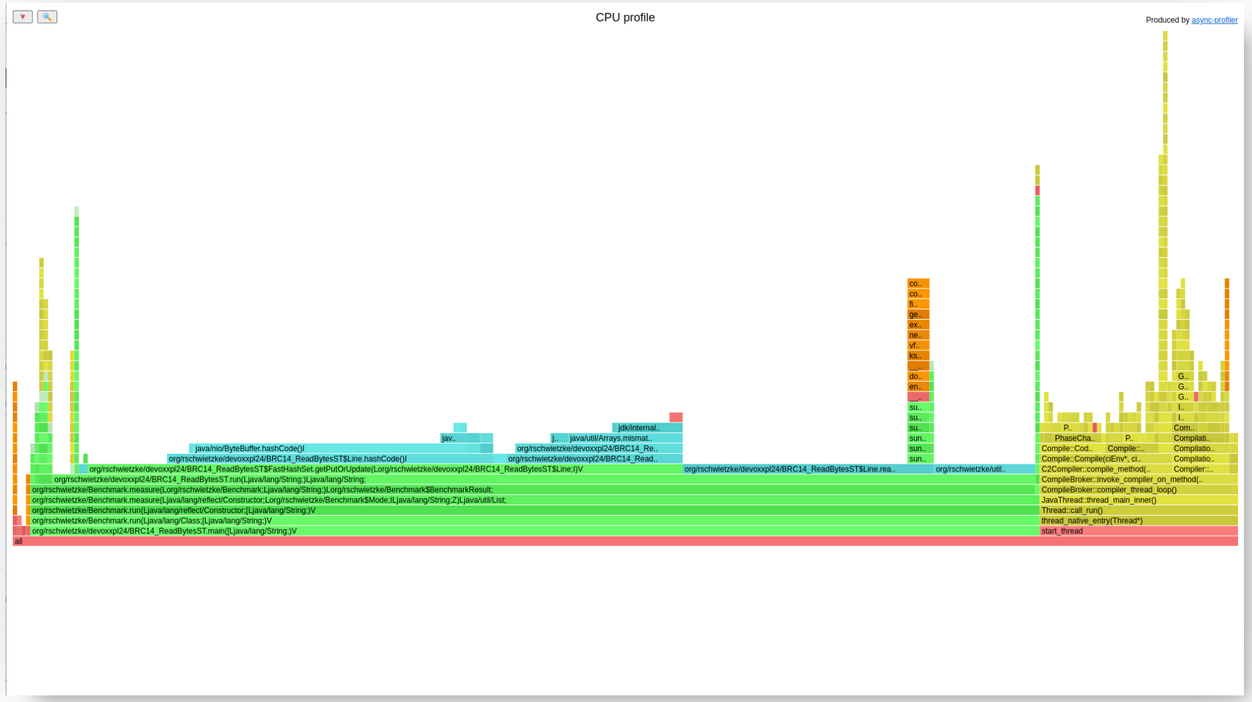
```
    hasNewLine = false;

    try
    {
        // do we near the end of the buffer?
        if (end - pos < REMAINING_MIN_BUFFERSIZE)
        {
            // we move the buffer indirectly, because the Byt
            // wraps our array, nothing for the tenderheartec
            System.arraycopy(data, pos, data, 0, data.length
            end = end - pos;
            pos = 0;
            buffer.position(end);

            // fill the buffer up
            final int readBytes = channel.read(buffer);
            if (readBytes == -1)
            {
                EOF = true;
            }

            end = buffer.position();
        }
    }
    catch (IOException e)
    {
        e.printStackTrace();
        EOF = true;
        throw new RuntimeException(e);
    }
```

```
        int i = pos;
        for (; i < end; i++)
        {
            final byte b = data[i];
            if (b == ';')
            {
                semicolonPos = i;
                break;
            }
        }

        i++;
        for (; i < end; i++)
        {
            final byte b = data[i];
            if (b == '\n')
            {
                newlinePos = i;
                pos = i + 1;
                hasNewLine = true;
                return;
            }
        }
    }
```


THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %
BRC12_NewMapST	105.0 s	47.9 %
BRC13_HardcodedSetST	102.5 s	46.8 %
BRC14_ReadBytesST	53.7 s	24.5 %

FLAMEGRAPH



CPU Flamegraph

run	80.2 %
read	2.1 %
parse line	18.4 %
parseInteger	8.3 %
putOrUpdate	**48.6 %**
hashCode	26.6 %
equals	14.4 %
put/update	7.6 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC14_ReadBytesST measurements.txt 2 2 \
```

A SURPRISE

A SURPRISE

Enjoy a treat

NO CHURN ANYMORE

A benefit we have not asked for

```
# Run with max heap 4 GB for 10 M lines
java -Xlog:gc*=debug \
     -Xms4m -Xmx4g \
     -XX:+UnlockExperimentalVMOptions \
     -XX:+UseEpsilonGC \
     -cp target/classes/ \
     org.rschwietzke.devoxxpl24.BRC01_BaselineST \
     data-10m.txt 0 1

[3.726s][info][gc] Heap: 4096M reserved,
                         3204M (78.22%) committed,
                         3199M (78.11%) used
```

1000 M lines = 100x 3.2 GB = **320 GB**

```
# Run with max heap 4 MB for 1000 M lines
java -Xlog:gc*=debug \
     -Xms4m -Xmx4m \
     -XX:+UnlockExperimentalVMOptions \
     -XX:+UseEpsilonGC \
     -cp target/classes/ \
     org.rschwietzke.devoxxpl24.BRC14_ReadBytesST \
     data-1000m.txt 0 1

[55.401s][info][gc] Heap: 4096K reserved,
                          4096K (100.00%) committed,
                          3634K (88.73%) used
```

1000 M lines = **3.7 MB**

We can process 13 GB of CSV file data with just 4 MB of heap!

SOME CHEAPISH TUNING

Attempting to save on number parsing

```
/**
 * Parses a double but ends up with an int, only because we know
 * the format of the results -99.9 to 99.9
 */
public static int parseIntegerFixed(final byte[] b, final int offset, final int end) {
    final int length = end - offset; // one is missing, we care for that later

    // we know the first three pieces already 9.9
    int p0 = b[end];
    int p1 = b[end - 2] * 10;
    int value = p0 + p1 - (DIGITOFFSET + DIGITOFFSET * 10);

    // we are 9.9
    if (length == 2) {
        return value;
    }

    // ok, we are either -9.9 or 99.9 or -99.9
    if (b[offset] != (byte)'-') {
        // we are 99.9
        value += b[end - 3] * 100 - DIGITOFFSET * 100;
        return value;
    }

    // we are either -99.9 or -9.9
```

```
    if (length == 3) {
        // -9.9
        return -value;
    }

    // -99.9
    value += b[end - 3] * 100 - DIGITOFFSET * 100;
    return -value;
}
```

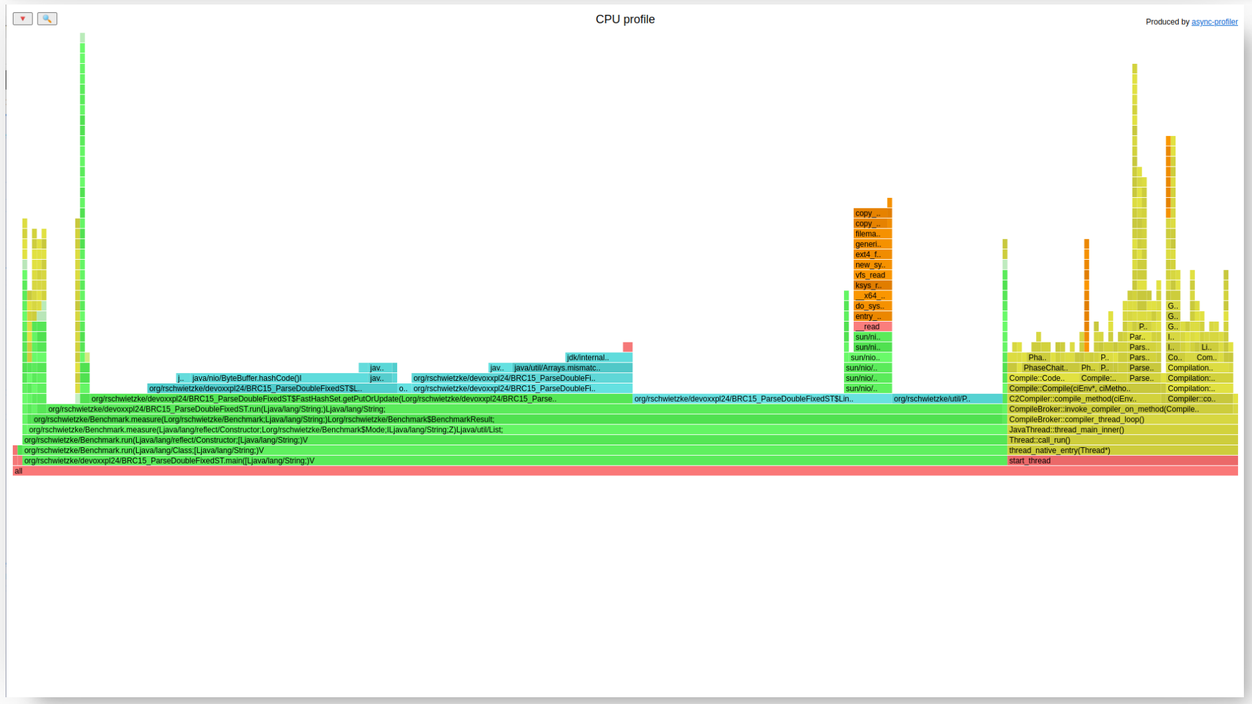
org.rschwietzke.devoxxpl24.BRC15_ParseDoubleFixedST

THE RESULTS

Test	Time	%
BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %
BRC12_NewMapST	105.0 s	47.9 %
BRC13_HardcodedSetST	102.5 s	46.8 %

BRC14_ReadBytesST	53.7 s	24.5 %
BRC15_ParseDoubleFixedST	51.1 s	23.3 %

FLAMEGRAPH



CPU Flamegraph

Area	14	15
run	**80.2 %**	**78.0 %**
read	2.1 %	3.9 %
parse line	18.4 %	18.3 %
parseInteger	8.3 %	9.0 %
putOrUpdate	**48.6 %**	**44.3 %**
hashCode	26.6 %	26.6 %
equals	14.4 %	14.4 %
put/update	7.6 %	7.6 %

```
java \
    -agentpath:async-profiler/lib/libasyncProfiler.so=start,event=cpu,sig,flamegraph,file=$1-cpu.html \
    -cp target/classes/ \
    org.rschwietzke.devoxxpl24.BRC15_ParseDoubleFixedST measurements.txt 2 2 \
```

CHANGING GEARS

Tuning is not that simple anymore

MORE IDEAS

Hot but not really hot, because we need that code

- `hashCode` still on top
- `equals` for the byte array
- `readFromChannel` w/o real IO
- Think harder first
- Resort to other metrics for further tuning

Simpler Min/Max

```
public void add(final int value)
{
```

```
public void add(final int value)
{
```

```
    this.min = Math.min(this.min, value);
    this.max = Math.max(this.max, value);
    this.total += value;
    this.count++;
}
```

```
    if (value < this.min)
    {
        this.min = value;
    }
    else if (value > this.max)
    {
        this.max = value;
    }
    this.total += value;
    this.count++;
}
```

org.rschwietzke.devoxxpl24.BRC21_ManualMinMaxST

HashCode

```
// look for semicolon and new line
int i = pos;
for (; i < end; i++)
{
    final byte b = data[i];
    if (b == ';')
    {
        semicolonPos = i;
        break;
    }
}
```

```
// We loop twice, when searching the ; and when
// calculating the hashcode later.
// Do it in one go, will safe also a comparison

// look for semicolon and new line
// when checking for semicolon, we do the hashcode right away
int h = 1;
int i = pos;
for (; i < end; i++)
{
    final byte b = data[i];
    if (b == ';')
    {
        semicolonPos = i;
        break;
    }
    h = 31 * h + b;
```

```
}
this.hashCode = h;
```

org.rschwietzke.devoxxpl24.BRC22_EarlyHashCodeST

HashCode

```
// we loop twice, when searching the ; and when calculating t
// do it in one go, will safe also a comparison

// look for semicolon and new line
// when checking for semicolon, we do the hashcode right away
int h = 1;
int i = pos;
for (; i < end; i++)
{
    final byte b = data[i];
    if (b == ';')
    {
        semicolonPos = i;
        break;
    }
    h = 31 * h + b;
}
this.hashCode = h;
```

```
// remove the multiplication and
// use bit shift and subtraction

// look for semicolon and new line
// when checking for semicolon, we do the hashcode right away
int h = 1;
int i = pos;
for (; i < end; i++)
{
    final byte b = data[i];
    if (b == ';')
    {
        semicolonPos = i;
        break;
    }
    h = (h << 5) - h + b;
}
this.hashCode = h;
```

org.rschwietzke.devoxxpl24.BRC23_NoMulST

THE RESULTS

Test	Time	%

BRC01_BaselineST	219.1 s	100.0 %
BRC03_NoStreamST	197.9 s	90.3 %
BRC05_ReplaceSplitST	151.1 s	69.0 %
BRC06_NewDoubleParsingST	127.9 s	58.4 %
BRC07_NoCopyForDoubleST	120.4 s	55.0 %
BRC08_GoIntST	117.4 s	53.6 %
BRC09_NoMergeST	121.3 s	55.4 %
BRC10_MutateST	100.6 s	45.9 %
BRC12_NewMapST	105.0 s	47.9 %
BRC13_HardcodedSetST	102.5 s	46.8 %
BRC14_ReadBytesST	53.7 s	24.5 %
BRC15_ParseDoubleFixedST	51.1 s	23.3 %
BRC20_UseArrayNoBufferST	50.4 s	23.0 %
BRC21_ManualMinMaxST	52.0 s	23.7 %
BRC22_EarlyHashCodeST	**40.0 s**	**18.3 %**
BRC23_NoMulST	40.3 s	18.4 %
BRC24_DifferentBranchInHashCodeST	40.4 s	18.5 %
BRC25_SmallAddReordingST	39.9 s	18.2 %
BRC26_MoreMapSpaceST	39.0 s	17.8 %
BRC27_SmallPutST	37.9 s	17.3 %
BRC30_DangerNoEqualsST	28.4 s	13.0 %

THE RESULTS - MORE FIDDLING

Test	Time	%

BRC01_BaselineST	228.07 s	100.0 %
BRC26_MoreMapSpaceST	39.93 s	17.5 %

BRC27_SmallPutST	39.84 s	17.5 %
BRC28_FineTuningST	38.70 s	17.0 %
BRC29a_ParseDoubleTuningST	38.83 s	17.0 %
BRC29b_ParseDoubleTuning2ST	38.68 s	17.0 %

FINAL THOUGHTS AND NUMBERS

A few last data tables

DIFFERENT MACHINES

Test	What	Machine	%

BRC01_BaselineST	Thinkpad T14s	218.8 s	100.0 %
BRC26_MoreMapSpaceST	Thinkpad T14s	35.3 s	16.1 %
BRC01_BaselineST	GCP-C2-4c-32g	278.4 s	100.0 %
BRC26_MoreMapSpaceST	GCP-C2-4c-32g	43.4 s	15.6 %
BRC01_BaselineST	GCP-C2D-4c-32g	172.4 s	100.0 %
BRC26_MoreMapSpaceST	GCP-C2D-4c-32g	34.8 s	20.2 %

Your milage will vary.

DIFFERENT JDKS

Test	What	Time	%
BRC01_BaselineST	21.0.3-tem	215.5 s	100.0 %
BRC26_MoreMapSpaceST	21.0.3-tem	38.4 s	17.8 %
BRC01_BaselineST	21.0.2-graal	212.7 s	100.0 %
BRC26_MoreMapSpaceST	21.0.2-graal	39.8 s	17.6 %
BRC26_MoreMapSpaceST	graal-native-o3	41.4 s	
BRC26_MoreMapSpaceST	graal-native-pgo	44.7 s	

P.S: Alina, I have no clue why the native images perform so badly.

DISCLAIMER AND WARNING

The cloud sucks when benchmarking

- Shared hardware is bad for small numbers
- Burst capacity might be a problem
- Other users running stuff on that host
- Some tools fail aka `perf`

```
# BRC01_BaselineST - DO CPU 16c
Warmup Runtime:       219,665 ms
Warmup Runtime:       220,341 ms
Warmup Runtime:       219,312 ms
Measurement Runtime: 219,020 ms
Measurement Runtime: 223,180 ms
Measurement Runtime: 226,418 ms

# BRC27_SmallPutST - DO CPU 16c
Warmup Runtime:       38,792 ms
Warmup Runtime:       37,929 ms
Warmup Runtime:       39,136 ms
Measurement Runtime: 39,197 ms
Measurement Runtime: 39,103 ms
Measurement Runtime: 39,106 ms
```

DISCLAIMER AND WARNING

The cloud sucks when benchmarking

```
# Google Cloud C2 - 4c - 32 GB
# BRC01_BaselineST - DO CPU 16c
Measurement Runtime: 281,448 ms
Measurement Runtime: 278,622 ms
Measurement Runtime: 281,130 ms
Measurement Runtime: 280,942 ms
Measurement Runtime: 281,287 ms

# BRC26_MoreMapSpaceST - DO CPU 16c
Measurement Runtime:  43,390 ms
Measurement Runtime:  44,003 ms
Measurement Runtime:  43,634 ms
Measurement Runtime:  43,682 ms
Measurement Runtime:  44,305 ms
```

```
# Google Cloud C2D - 4c - 32 GB
# BRC01_BaselineST - DO CPU 16c
Measurement Runtime: 172,498 ms
Measurement Runtime: 197,795 ms
Measurement Runtime: 195,888 ms
Measurement Runtime: 195,546 ms
Measurement Runtime: 195,896 ms

# BRC26_MoreMapSpaceST - DO CPU 16c
Measurement Runtime:  38,891 ms
Measurement Runtime:  34,864 ms
Measurement Runtime:  35,682 ms
Measurement Runtime:  35,691 ms
Measurement Runtime:  35,712 ms
```

BEHIND THE SCENES

```
# org.rschwietzke.devoxxpl24.BRC01_BaselineST
# 21.0.2-tem
# Mean Measurement Runtime: 225,401 ms

      228.323,41 ms task-clock                #   1,011 CPUs utilized
          119.153    context-switches         # 521,861 /sec
            7.906    cpu-migrations           #  34,626 /sec
           92.233    page-faults              # 403,958 /sec
  862.082.804.553    cycles                   #   3,776 GHz
    5.600.297.153    stalled-cycles-frontend  #   0,65% frontend cycles idle
```

```
# org.rschwietzke.devoxxpl24.BRC29b_ParseDoubleTuning2ST
# 21.0.2-tem
# Mean Measurement Runtime: 34,543 ms

      34.963,17 ms task-clock                #   1,009 CPUs utilized
           5.663    context-switches         # 161,970 /sec
             104    cpu-migrations           #   2,975 /sec
           9.110    page-faults              # 260,560 /sec
 140.577.081.498    cycles                   #   4,021 GHz
   1.629.722.236    stalled-cycles-frontend  #   1,16% frontend cycles idle
```

```
  5.000.297.153     stalled-cycles-frontend #   0,65% frontend cycles idle
  283.745.763.208     stalled-cycles-backend  #  32,91% backend cycles idle
2.314.668.391.875     instructions            #    2,68 insn per cycle
                                              #    0,12 stalled cycles per ins
  527.949.461.731     branches                #   2,312 G/sec
    2.272.060.098     branch-misses           #   0,43% of all branches

    222,936074000 seconds user
      5,665090000 seconds sys

# 21.0.2-graal
# Mean Measurement Runtime: 197,274 ms

       200.396,37 ms task-clock               #   1,015 CPUs utilized
          103.682     context-switches        # 517,385 /sec
            7.450     cpu-migrations          #  37,176 /sec
          125.728     page-faults             # 627,397 /sec
  791.749.963.199     cycles                  #   3,951 GHz
    5.159.985.458     stalled-cycles-frontend #   0,65% frontend cycles idle
  280.786.721.793     stalled-cycles-backend  #  35,46% backend cycles idle
2.075.246.381.763     instructions            #    2,62 insn per cycle
                                              #    0,14 stalled cycles per ins
  430.523.391.358     branches                #   2,148 G/sec
    2.321.436.824     branch-misses           #   0,54% of all branches

    195,555976000 seconds user
      5,031498000 seconds sys
```

```
  1.629.722.236     stalled-cycles-frontend #   1,16% frontend cycles idle
 51.856.701.946     stalled-cycles-backend  #  36,89% backend cycles idle
375.818.774.170     instructions            #    2,67 insn per cycle
                                            #    0,14 stalled cycles per insn
 59.757.923.585     branches                #   1,709 G/sec
  1.380.896.709     branch-misses           #   2,31% of all branches

     33,348061000 seconds user
      1,620892000 seconds sys

# 21.0.2-graal
# Mean Measurement Runtime: 28,154 ms

      29.217,74 ms task-clock               #   1,034 CPUs utilized
           5.975     context-switches        # 204,499 /sec
              84     cpu-migrations          #   2,875 /sec
          38.589     page-faults             #   1,321 K/sec
 120.424.476.008     cycles                  #   4,122 GHz
   2.187.333.370     stalled-cycles-frontend #   1,82% frontend cycles idle
  32.222.439.801     stalled-cycles-backend  #  26,76% backend cycles idle
 342.431.349.170     instructions            #    2,84 insn per cycle
                                            #    0,09 stalled cycles per insn
  67.659.979.890     branches                #   2,316 G/sec
   1.418.694.789     branch-misses           #   2,10% of all branches

    27,579431000 seconds user
     1,647132000 seconds sys
```

WHAT IS LEFT?

There is still room

- `byte` limits us, read only 8 bit at a time but could read 64 bit
- `int` or `long` would be better (need for UNSAFE or `ByteBuffer`)
- `mmap` a file instead of filling a buffer
- Multi-threading
- Tuning for inlining and CPU pipeling still possible
- Reduce branching

The first 80% are not that hard.

Most of the concepts are applicable to other problems including the concept

of changing one thing at a time.

Most expensive first, unless something is simpler to fix.