Java Performance - TP2

October 23, 2018

Thomas COUCHOUD
thomas.couchoud@etu.univ-tours.fr
Victor COLEAU
victor.coleau@etu.univ-tours.fr

Chapter 1

Etude de JMH

1.1 Annotations

JMH fonctionne principalement avec des annotations. Parmi celles-ci nous retrouvons notamment:

- **Benchmark**: Permet de définir une méthode comme étant une méthode à tester
- **BenchmarkMode**: Permet d'indiquer quelles métriques nous voulons étudier, nous avons:
  - Throughput: Mesure le nombre d'opérations par seconde
  - Average Time: Mesure le temps moyen d'exécution de la méthode
  - Sample Time: Mesure les temps d'exécution de la méthode: min, max, ...
  - Single Shot Time: Mesure le temps d'exécution pour un run unique du benchmark. Cela peut être utile pour voir la performance de la méthode sans le hotspot.
- **OutputTimeUnit**: Permet de définir les unités de temps utilisées pour les rapports
- **Fork**: Permet de définir le nombre de JVM qui sont forkées. En effet par défaut la JVM est forkée à chaque "trial" de la méthode à benchmarker, cependant dans certains cas il peut être utile de changer ce comportement. Il faut tout de même faire attention lors de l'utilisation de cette annotation car des comportements anormaux peuvent apparaître (si l'un a deux classes implémentant la même interface par exemple, en effet dans ce cas la première sera plus rapide car la JVM remplace les appels méthodes).
- **Group**: Cette annotation permet de définir à quel groupe appartient le test. Cela permet de lancer tous les tests d'un même groupe en même temps (voir annotation suivante). Par défaut chaque méthode est dans un groupe unique.
- **GroupThread**: Définit le nombre de threads qui sera utilisé pour lancer la méthode. Dans le cas d'un groupe il y aura un nombre de threads égal à la somme de tous les GroupThreads du groupe.
- **Measurement**: Permet de définir les valeurs par défaut pour le benchmark de la méthode:
  - Nombre d'itérations
  - Temps de mesure
  - Taille d'un batch
- **State**: Annotation à mettre sur une classe afin de définir dans quel état cette dernière sera utilisée:
  - Thread: Valeur par défaut. Une instance de cette classe sera utilisée pour chaque thread qui exécute le benchmark.
  - Benchmark: Une instance unique sera partagée entre tous les threads qui exécutent le même benchmark. Peut être utile dans le cas de méthodes exécutées en parrallèle.
  - Group: Une instance sera allouée pour chaque groupe.
- **TearDown / Setup**: Cette annotation permet de marquer les méthodes à lancer avant et après les benchmark. L'appel à ces méthodes peuvent être précisés grâce a une paramètre de l'annotation:
  - Trial: Valeur par défaut. La méthode est appelée après/avant un benchmark entier.
  - Iteration: La méthode est appelée avant/après une itération (plusieurs invocations).

  - Invocation: La méthode est appelée avant/après une invocation (chaque run).

- **Timeout**: Définit le temps maximum que le benchmark peut prendre.
- **Warmup**: Définit les paramètres pour la phase de chauffe. Similaire à Measurement mais concerne juste la phase de chauffe.
- **CompilerControl**: Définit les options de compilation lors d'un fork de la JVM pour le benchmark de cette méthode. Cependant il faut faire attention car le compilateur peut ignorer ces derniers.

1.2 BalckHole

Le BlackHole permet de consommer, sans donner d'informations si la valeur est réellement utilisée ou non à JIT (compilation à la volée), des variables ou bien du temps CPU.

Dans le cas de variables cela est utile car cela évite que le compilateur réalise des optimisations où la variable ne serait pas calculée.

1.3 Résultats de MyBenchmark

(Code disponible en Appendix A)

Voici la sortie pour un benchmark:

Listing 1.1 – out.log

```
# JMH version: 1.21
# VM version: JDK 1.8.0_171, OpenJDK 64-Bit Server VM, 25.171-b11
# VM invoker: /usr/lib/jvm/java-8-openjdk-amd64/jre/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 5 iterations, 10 s each
# Measurement: 5 iterations, 10 s each
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark.fact

# Run progress: 0.00% complete, ETA 00:16:40
# Fork: 1 of 5
# Warmup Iteration 1: 84603912.755 ops/s
# Warmup Iteration 2: 82896037.779 ops/s
# Warmup Iteration 3: 97368528.704 ops/s
# Warmup Iteration 4: 97875393.382 ops/s
# Warmup Iteration 5: 97872525.925 ops/s
Iteration  1: 97819325.439 ops/s
Iteration  2: 98012719.194 ops/s
Iteration  3: 97591403.605 ops/s
Iteration  4: 97814943.829 ops/s
Iteration  5: 96757314.458 ops/s

# Run progress: 10.00% complete, ETA 00:15:06
# Fork: 2 of 5
# Warmup Iteration 1: 85290497.139 ops/s
# Warmup Iteration 2: 82777393.661 ops/s
# Warmup Iteration 3: 97806785.254 ops/s
# Warmup Iteration 4: 97951394.470 ops/s
# Warmup Iteration 5: 97921213.373 ops/s
Iteration  1: 97939927.352 ops/s
Iteration  2: 97868584.771 ops/s
Iteration  3: 98095783.580 ops/s
Iteration  4: 98099712.187 ops/s
Iteration  5: 97887580.667 ops/s

# Run progress: 20.00% complete, ETA 00:13:24
# Fork: 3 of 5
# Warmup Iteration 1: 85419408.056 ops/s
# Warmup Iteration 2: 83225598.451 ops/s
# Warmup Iteration 3: 97896644.463 ops/s
# Warmup Iteration 4: 97752422.955 ops/s
```

```
# Warmup Iteration 5: 97583200.816 ops/s
Iteration  1: 97456878.760 ops/s
Iteration  2: 97658458.826 ops/s
Iteration  3: 97869433.122 ops/s
Iteration  4: 97825488.380 ops/s
Iteration  5: 97631526.478 ops/s

# Run progress: 30.00% complete, ETA 00:11:43
# Fork: 4 of 5
# Warmup Iteration 1: 85280330.363 ops/s
# Warmup Iteration 2: 83234718.351 ops/s
# Warmup Iteration 3: 97957582.498 ops/s
# Warmup Iteration 4: 97359323.794 ops/s
# Warmup Iteration 5: 97928819.893 ops/s
Iteration  1: 97942649.273 ops/s
Iteration  2: 97913140.638 ops/s
Iteration  3: 98081409.370 ops/s
Iteration  4: 97774009.651 ops/s
Iteration  5: 97438788.773 ops/s

# Run progress: 40.00% complete, ETA 00:10:03
# Fork: 5 of 5
# Warmup Iteration 1: 85336199.983 ops/s
# Warmup Iteration 2: 83213631.656 ops/s
# Warmup Iteration 3: 97838773.067 ops/s
# Warmup Iteration 4: 97956246.539 ops/s
# Warmup Iteration 5: 98024030.802 ops/s
Iteration  1: 98029100.103 ops/s
Iteration  2: 97628740.314 ops/s
Iteration  3: 97603188.114 ops/s
Iteration  4: 97926136.932 ops/s
Iteration  5: 98051650.436 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark.fact":
  97788715.770 +-(99.9%) 216107.541 ops/s [Average]
  (min, avg, max) = (96757314.458, 97788715.770, 98099712.187), stdev = 288497.384
  CI (99.9%): [97572608.229, 98004823.311] (assumes normal distribution)
```

On retrouve entre les lignes 1 à 10 les différentes informations concernant le test. Le reste du fichier (L12-75) contient tous les run. A la fin (L78-81) nous obtenons les résultats finaux.

1.3.1 Informations

On retrouve les informations concernant la JVM (version JDK, version VM, options, ...) ainsi que la version de JMH. De plus nous avons les paramètres de JMH concernant le nombre d'itérations et le temps pour les phases de chauffe et de test (voir les annotations Measurement et Warmup). Les valeurs des annotations Timeout, BenchmarkMode, Thread sont aussi présentes.

1.3.2 Runs

Pour chaque run on nous indique la progression totale du benchmark ainsi qu'une estimation du temps restant. Suit le numéro du run pour la méthodes testée. Puis on retrouve les différentes itérations du warmup avec les valeurs demandées du benchmark pour l'itération donnée. De la même façon suivent les itérations de test.

1.3.3 Résultats

Les résultats nous affichent (dans notre cas) le nombre d'opérations moyen par seconde avec un taux de fiabilité. De plus les valeurs min et max sont aussi présentes avec leur ecart-type.

1.3.4 Analyse

Problème pour l'analyse car on a benchmark que la factorielle non récursive. Cependant la semaine suivante JMH se lançait avec un JDK9. Afin de ne pas falsifier les résultats nous sommes passés au MyBenchmark2.

De plus la factorielle fait rapidement exploser les int et ne nous permet donc pas de tester avec des paramètres d'entré très différents.

1.4 Résultats - MyBenchmark2

(Code disponible en Appendix B)

1.4.1 Analyse

Première chose que l'on remarque est que pour 50000 éléments la méthode de concaténation de string affiche un score de 0 ops/s. On peut se douter que cela n'est pas normal. Cela pourrait être du au fait que le nombre réel est tellement faible qu'une fois arrondi il devient 0.

Ensuite on voit que de manière générale fusionner 50 morceaux de chaine est plus rapide que en fusionner 50000.

Si l'on compare les méthodes testées:

- Sur des petites fusions (50 éléments) le StringBuilder domine largement le StringBuffer lui même largement supérieur à la concaténation simple.
- Sur des grandes fusions (50000 éléments) StringBuffer et StringBuilder sont equivalents pendant que la concaténation est en PLS.

Les tests ont été refaits avec une capacité initiale de 500 pour les StringBuffer et StringBuilder. On remarque que sur des petites itérations, la performance du StringBuffer et StringBuilder est dégradée. Cela s'explique par le fait qu'une allocation de 500 est inutile pour 50 éléments.

En revanche pour des fusions de 50000 éléments les performances sont:

- Légèrement améliorées pour le StringBuilder
- Très largement améliorées pour le StringBuffer

De son coté la concaténation simple pleure toujours.

Appendix A

MyBenchmark

Listing A.1 – MyBenchmark.java

```
/*
 * Copyright (c) 2014, Oracle America, Inc.
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions are met:
 *
 *  * Redistributions of source code must retain the above copyright notice,
 *    this list of conditions and the following disclaimer.
 *
 *  * Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 *
 *  * Neither the name of Oracle nor the names of its contributors may be used
 *    to endorse or promote products derived from this software without
 *    specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS"
 * AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR CONTRIBUTORS BE
 * LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR
 * CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF
 * SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS
 * INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
 * CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF
 * THE POSSIBILITY OF SUCH DAMAGE.
 */

package fr.polytechtours.javaperformance.tp2;

import org.openjdk.jmh.annotations.*;
import org.openjdk.jmh.infra.Blackhole;

import java.util.Collections;
import java.util.concurrent.TimeUnit;

public class MyBenchmark {



    @State(Scope.Thread)
    public static class MyState {
        @Param ({"5", "10"})
        public int i;
    }

    @Benchmark
```

```
51      @Warmup (iterations = 2, time = 8, batchSize = 3)
52      @Measurement (iterations = 2, time = 8, batchSize = 3)
53      public void testFact(MyState state, Blackhole bh) {
54          bh.consume(fact(state.i));
55      }
56
57      @Benchmark
58      @Warmup (iterations = 2, time = 8, batchSize = 3)
59      @Measurement (iterations = 2, time = 8, batchSize = 3)
60      public void testFactRec(MyState state, Blackhole bh) {
61          bh.consume(factRec(state.i));
62      }
63
64      public static int fact(int j){
65          int res = 1;
66          for (int i = 1; i <= j; i++) {
67              res *= i;
68          }
69          return res;
70      }
71
72      public static int factRec(int j){
73          if(j <= 1)
74              return 1;
75          return j * factRec(j-1);
76      }
77
78  }
```

Appendix B

MyBenchmark

Listing B.1 – MyBenchmark2.java

```java
package fr.polytechtours.javaperformance.tp2;

import org.openjdk.jmh.annotations.*;
import org.openjdk.jmh.infra.Blackhole;

public class MyBenchmark2 {

    @State(Scope.Thread)
    public static class MyState {
        @Param ({"50", "50_000"})
        public int i;
    }

    @Benchmark
    @Warmup (iterations = 2, time = 8, batchSize = 3)
    @Measurement (iterations = 2, time = 8, batchSize = 3)
    public void testConcat(MyBenchmark.MyState state, Blackhole bh) {
        bh.consume(stringConcat(state.i));
    }

    @Benchmark
    @Warmup (iterations = 2, time = 8, batchSize = 3)
    @Measurement (iterations = 2, time = 8, batchSize = 3)
    public void testBuilder(MyBenchmark.MyState state, Blackhole bh) {
        bh.consume(stringBuilder(state.i));
    }

    @Benchmark
    @Warmup (iterations = 2, time = 8, batchSize = 3)
    @Measurement (iterations = 2, time = 8, batchSize = 3)
    public void testBuffer(MyBenchmark.MyState state, Blackhole bh) {
        bh.consume(stringBuffer(state.i));
    }



    public String stringConcat(int max) {
        String string = "";

        for (int i = 0; i < max; i++)
            string = string + i;

        return string;
    }

    public String stringBuilder(int max) {
        StringBuilder string = new StringBuilder(100);

        for (int i = 0; i < max; i++)
            string.append(i);
```

```
51
52          return string.toString();
53      }
54
55      public String stringBuffer(int max) {
56          StringBuffer string = new StringBuffer();
57
58          for (int i = 0; i < max; i++)
59              string.append(i);
60
61          return string.toString();
62      }
63
64  }
```

Listing B.2 – out2.log

```
1  /usr/lib/jvm/java-9-oracle/bin/java -Dfile.encoding=UTF-8 -classpath
       /home/administrateur/Bureau/java-performance/TP2/jmh/target/classes:/home/administrateur/.m2/repository/org/openj
       org.openjdk.jmh.Main fr.polytechtours.javaperformance.tp2.MyBenchmark2.*
2  WARNING: An illegal reflective access operation has occurred
3  WARNING: Illegal reflective access by org.openjdk.jmh.util.Utils
       (file:/home/administrateur/.m2/repository/org/openjdk/jmh/jmh-core/1.21/jmh-core-1.21.jar) to field
       java.io.PrintStream.charOut
4  WARNING: Please consider reporting this to the maintainers of org.openjdk.jmh.util.Utils
5  WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
6  WARNING: All illegal access operations will be denied in a future release
7  # JMH version: 1.21
8  # VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
9  # VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
10 # VM options: -Dfile.encoding=UTF-8
11 # Warmup: 2 iterations, 1 s each, 3 calls per op
12 # Measurement: 2 iterations, 1 s each, 3 calls per op
13 # Timeout: 10 min per iteration
14 # Threads: 1 thread, will synchronize iterations
15 # Benchmark mode: Throughput, ops/time
16 # Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer
17 # Parameters: (i = 50)
18
19 # Run progress: 0.00% complete, ETA 00:02:00
20 # Fork: 1 of 5
21 # Warmup Iteration 1: 454602.693 ops/s
22 # Warmup Iteration 2: 598122.027 ops/s
23 Iteration  1: 603770.187 ops/s
24 Iteration  2: 631909.187 ops/s
25
26 # Run progress: 3.33% complete, ETA 00:02:19
27 # Fork: 2 of 5
28 # Warmup Iteration 1: 121061.894 ops/s
29 # Warmup Iteration 2: 128396.304 ops/s
30 Iteration  1: 129493.763 ops/s
31 Iteration  2: 195869.715 ops/s
32
33 # Run progress: 6.67% complete, ETA 00:02:11
34 # Fork: 3 of 5
35 # Warmup Iteration 1: 120718.627 ops/s
36 # Warmup Iteration 2: 128071.331 ops/s
37 Iteration  1: 129115.744 ops/s
38 Iteration  2: 198825.557 ops/s
39
40 # Run progress: 10.00% complete, ETA 00:02:05
41 # Fork: 4 of 5
42 # Warmup Iteration 1: 120862.152 ops/s
43 # Warmup Iteration 2: 129484.160 ops/s
44 Iteration  1: 131156.775 ops/s
45 Iteration  2: 192861.403 ops/s
46
47 # Run progress: 13.33% complete, ETA 00:02:00
```

```
# Fork: 5 of 5
# Warmup Iteration 1: 118380.560 ops/s
# Warmup Iteration 2: 127874.840 ops/s
Iteration  1: 129438.129 ops/s
Iteration  2: 196461.529 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer":
  253890.199 +-(99.9%) 293994.970 ops/s [Average]
  (min, avg, max) = (129115.744, 253890.199, 631909.187), stdev = 194459.469
  CI (99.9%): [~= 0, 547885.169] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer
# Parameters: (i = 50000)

# Run progress: 16.67% complete, ETA 00:01:55
# Fork: 1 of 5
# Warmup Iteration 1: 192.576 ops/s
# Warmup Iteration 2: 211.876 ops/s
Iteration  1: 217.638 ops/s
Iteration  2: 221.509 ops/s

# Run progress: 20.00% complete, ETA 00:01:51
# Fork: 2 of 5
# Warmup Iteration 1: 189.546 ops/s
# Warmup Iteration 2: 212.817 ops/s
Iteration  1: 218.188 ops/s
Iteration  2: 219.848 ops/s

# Run progress: 23.33% complete, ETA 00:01:46
# Fork: 3 of 5
# Warmup Iteration 1: 185.343 ops/s
# Warmup Iteration 2: 217.435 ops/s
Iteration  1: 213.623 ops/s
Iteration  2: 218.633 ops/s

# Run progress: 26.67% complete, ETA 00:01:41
# Fork: 4 of 5
# Warmup Iteration 1: 186.599 ops/s
# Warmup Iteration 2: 214.574 ops/s
Iteration  1: 219.555 ops/s
Iteration  2: 226.569 ops/s

# Run progress: 30.00% complete, ETA 00:01:37
# Fork: 5 of 5
# Warmup Iteration 1: 186.632 ops/s
# Warmup Iteration 2: 211.505 ops/s
Iteration  1: 217.689 ops/s
Iteration  2: 223.534 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer":
  219.679 +-(99.9%) 5.377 ops/s [Average]
  (min, avg, max) = (213.623, 219.679, 226.569), stdev = 3.557
  CI (99.9%): [214.301, 225.056] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
```

```
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder
# Parameters: (i = 50)

# Run progress: 33.33% complete, ETA 00:01:32
# Fork: 1 of 5
# Warmup Iteration 1: 456087.730 ops/s
# Warmup Iteration 2: 564047.839 ops/s
Iteration  1: 585250.271 ops/s
Iteration  2: 594795.498 ops/s

# Run progress: 36.67% complete, ETA 00:01:27
# Fork: 2 of 5
# Warmup Iteration 1: 469129.526 ops/s
# Warmup Iteration 2: 571545.304 ops/s
Iteration  1: 575985.212 ops/s
Iteration  2: 577642.178 ops/s

# Run progress: 40.00% complete, ETA 00:01:23
# Fork: 3 of 5
# Warmup Iteration 1: 473455.424 ops/s
# Warmup Iteration 2: 562980.085 ops/s
Iteration  1: 588724.059 ops/s
Iteration  2: 594426.456 ops/s

# Run progress: 43.33% complete, ETA 00:01:18
# Fork: 4 of 5
# Warmup Iteration 1: 468560.663 ops/s
# Warmup Iteration 2: 570173.399 ops/s
Iteration  1: 590509.874 ops/s
Iteration  2: 601103.438 ops/s

# Run progress: 46.67% complete, ETA 00:01:13
# Fork: 5 of 5
# Warmup Iteration 1: 466260.404 ops/s
# Warmup Iteration 2: 567276.435 ops/s
Iteration  1: 587735.990 ops/s
Iteration  2: 594173.684 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder":
  589034.666 +-(99.9%) 11847.506 ops/s [Average]
  (min, avg, max) = (575985.212, 589034.666, 601103.438), stdev = 7836.392
  CI (99.9%): [577187.160, 600882.172] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder
# Parameters: (i = 50000)

# Run progress: 50.00% complete, ETA 00:01:09
# Fork: 1 of 5
# Warmup Iteration 1: 187.287 ops/s
# Warmup Iteration 2: 218.884 ops/s
Iteration  1: 221.553 ops/s
```

```
Iteration  2: 227.495 ops/s

# Run progress: 53.33% complete, ETA 00:01:04
# Fork: 2 of 5
# Warmup Iteration 1: 196.395 ops/s
# Warmup Iteration 2: 227.352 ops/s
Iteration  1: 222.548 ops/s
Iteration  2: 230.681 ops/s

# Run progress: 56.67% complete, ETA 00:00:59
# Fork: 3 of 5
# Warmup Iteration 1: 209.639 ops/s
# Warmup Iteration 2: 240.521 ops/s
Iteration  1: 242.583 ops/s
Iteration  2: 237.776 ops/s

# Run progress: 60.00% complete, ETA 00:00:55
# Fork: 4 of 5
# Warmup Iteration 1: 207.647 ops/s
# Warmup Iteration 2: 234.289 ops/s
Iteration  1: 236.680 ops/s
Iteration  2: 241.378 ops/s

# Run progress: 63.33% complete, ETA 00:00:50
# Fork: 5 of 5
# Warmup Iteration 1: 193.578 ops/s
# Warmup Iteration 2: 212.556 ops/s
Iteration  1: 221.714 ops/s
Iteration  2: 223.888 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder":
  230.630 +-(99.9%) 12.635 ops/s [Average]
  (min, avg, max) = (221.553, 230.630, 242.583), stdev = 8.357
  CI (99.9%): [217.995, 243.264] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat
# Parameters: (i = 50)

# Run progress: 66.67% complete, ETA 00:00:46
# Fork: 1 of 5
# Warmup Iteration 1: 140990.784 ops/s
# Warmup Iteration 2: 185676.895 ops/s
Iteration  1: 185106.032 ops/s
Iteration  2: 186963.077 ops/s

# Run progress: 70.00% complete, ETA 00:00:41
# Fork: 2 of 5
# Warmup Iteration 1: 138280.016 ops/s
# Warmup Iteration 2: 200963.293 ops/s
Iteration  1: 190270.207 ops/s
Iteration  2: 190615.425 ops/s

# Run progress: 73.33% complete, ETA 00:00:36
# Fork: 3 of 5
# Warmup Iteration 1: 140461.741 ops/s
# Warmup Iteration 2: 197374.199 ops/s
Iteration  1: 188812.037 ops/s
Iteration  2: 190808.788 ops/s
```

```
# Run progress: 76.67% complete, ETA 00:00:32
# Fork: 4 of 5
# Warmup Iteration 1: 140257.435 ops/s
# Warmup Iteration 2: 193882.697 ops/s
Iteration  1: 187249.931 ops/s
Iteration  2: 183960.761 ops/s

# Run progress: 80.00% complete, ETA 00:00:27
# Fork: 5 of 5
# Warmup Iteration 1: 134898.070 ops/s
# Warmup Iteration 2: 186897.960 ops/s
Iteration  1: 184198.596 ops/s
Iteration  2: 190199.397 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat":
  187818.425 +-(99.9%) 4088.841 ops/s [Average]
  (min, avg, max) = (183960.761, 187818.425, 190808.788), stdev = 2704.515
  CI (99.9%): [183729.584, 191907.266] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat
# Parameters: (i = 50000)

# Run progress: 83.33% complete, ETA 00:00:23
# Fork: 1 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 86.67% complete, ETA 00:00:20
# Fork: 2 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 90.00% complete, ETA 00:00:16
# Fork: 3 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 93.33% complete, ETA 00:00:12
# Fork: 4 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 96.67% complete, ETA 00:00:06
# Fork: 5 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s
```

```
Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat":
  ~= 0 ops/s


# Run complete. Total time: 00:03:25

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

Benchmark                  (i)   Mode  Cnt      Score       Error  Units
MyBenchmark2.testBuffer     50  thrpt   10  253890.199 +- 293994.970 ops/s
MyBenchmark2.testBuffer  50000  thrpt   10     219.679 +-     5.377  ops/s
MyBenchmark2.testBuilder    50  thrpt   10  589034.666 +- 11847.506 ops/s
MyBenchmark2.testBuilder 50000  thrpt   10     230.630 +-    12.635  ops/s
MyBenchmark2.testConcat     50  thrpt   10  187818.425 +-  4088.841 ops/s
MyBenchmark2.testConcat  50000  thrpt   10         ~= 0              ops/s

Process finished with exit code 0
```

Listing B.3 – out2.log avec capacité initiale

```
/usr/lib/jvm/java-9-oracle/bin/java -Dfile.encoding=UTF-8 -classpath
    /home/administrateur/Bureau/java-performance/TP2/jmh/target/classes:/home/administrateur/.m2/repository/org/openj
    org.openjdk.jmh.Main fr.polytechtours.javaperformance.tp2.MyBenchmark2.*
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.openjdk.jmh.util.Utils
    (file:/home/administrateur/.m2/repository/org/openjdk/jmh/jmh-core/1.21/jmh-core-1.21.jar) to field
    java.io.PrintStream.charOut
WARNING: Please consider reporting this to the maintainers of org.openjdk.jmh.util.Utils
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer
# Parameters: (i = 50)

# Run progress: 0.00% complete, ETA 00:02:00
# Fork: 1 of 5
# Warmup Iteration 1: 61645.200 ops/s
# Warmup Iteration 2: 83584.566 ops/s
Iteration  1: 89105.141 ops/s
Iteration  2: 126027.081 ops/s

# Run progress: 3.33% complete, ETA 00:02:23
# Fork: 2 of 5
# Warmup Iteration 1: 148570.975 ops/s
# Warmup Iteration 2: 174353.456 ops/s
Iteration  1: 185422.046 ops/s
Iteration  2: 185512.970 ops/s

# Run progress: 6.67% complete, ETA 00:02:13
# Fork: 3 of 5
# Warmup Iteration 1: 142747.920 ops/s
# Warmup Iteration 2: 175933.814 ops/s
Iteration  1: 188628.389 ops/s
Iteration  2: 188726.998 ops/s
```

```
# Run progress: 10.00% complete, ETA 00:02:07
# Fork: 4 of 5
# Warmup Iteration 1: 140770.418 ops/s
# Warmup Iteration 2: 176918.283 ops/s
Iteration  1: 187189.945 ops/s
Iteration  2: 185607.517 ops/s

# Run progress: 13.33% complete, ETA 00:02:02
# Fork: 5 of 5
# Warmup Iteration 1: 146205.819 ops/s
# Warmup Iteration 2: 176641.124 ops/s
Iteration  1: 184576.999 ops/s
Iteration  2: 188780.648 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer":
  170957.773 +-(99.9%) 52248.281 ops/s [Average]
  (min, avg, max) = (89105.141, 170957.773, 188780.648), stdev = 34559.003
  CI (99.9%): [118709.493, 223206.054] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer
# Parameters: (i = 50000)

# Run progress: 16.67% complete, ETA 00:01:57
# Fork: 1 of 5
# Warmup Iteration 1: 75.862 ops/s
# Warmup Iteration 2: 91.878 ops/s
Iteration  1: 88.724 ops/s
Iteration  2: 113.749 ops/s

# Run progress: 20.00% complete, ETA 00:01:52
# Fork: 2 of 5
# Warmup Iteration 1: 75.536 ops/s
# Warmup Iteration 2: 86.621 ops/s
Iteration  1: 86.707 ops/s
Iteration  2: 113.935 ops/s

# Run progress: 23.33% complete, ETA 00:01:47
# Fork: 3 of 5
# Warmup Iteration 1: 76.870 ops/s
# Warmup Iteration 2: 90.705 ops/s
Iteration  1: 88.746 ops/s
Iteration  2: 116.773 ops/s

# Run progress: 26.67% complete, ETA 00:01:42
# Fork: 4 of 5
# Warmup Iteration 1: 78.757 ops/s
# Warmup Iteration 2: 84.889 ops/s
Iteration  1: 87.700 ops/s
Iteration  2: 112.813 ops/s

# Run progress: 30.00% complete, ETA 00:01:37
# Fork: 5 of 5
# Warmup Iteration 1: 76.830 ops/s
# Warmup Iteration 2: 88.689 ops/s
Iteration  1: 88.838 ops/s
Iteration  2: 113.741 ops/s


```

```
Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuffer":
  101.173 +-(99.9%) 20.841 ops/s [Average]
  (min, avg, max) = (86.707, 101.173, 116.773), stdev = 13.785
  CI (99.9%): [80.332, 122.013] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder
# Parameters: (i = 50)

# Run progress: 33.33% complete, ETA 00:01:33
# Fork: 1 of 5
# Warmup Iteration 1: 173775.025 ops/s
# Warmup Iteration 2: 211729.943 ops/s
Iteration  1: 215284.764 ops/s
Iteration  2: 218631.739 ops/s

# Run progress: 36.67% complete, ETA 00:01:28
# Fork: 2 of 5
# Warmup Iteration 1: 157557.533 ops/s
# Warmup Iteration 2: 183660.207 ops/s
Iteration  1: 189750.611 ops/s
Iteration  2: 196137.194 ops/s

# Run progress: 40.00% complete, ETA 00:01:23
# Fork: 3 of 5
# Warmup Iteration 1: 155657.877 ops/s
# Warmup Iteration 2: 185109.075 ops/s
Iteration  1: 195943.684 ops/s
Iteration  2: 194432.085 ops/s

# Run progress: 43.33% complete, ETA 00:01:19
# Fork: 4 of 5
# Warmup Iteration 1: 153848.693 ops/s
# Warmup Iteration 2: 182624.150 ops/s
Iteration  1: 190604.641 ops/s
Iteration  2: 194276.855 ops/s

# Run progress: 46.67% complete, ETA 00:01:14
# Fork: 5 of 5
# Warmup Iteration 1: 157671.679 ops/s
# Warmup Iteration 2: 183402.281 ops/s
Iteration  1: 194370.664 ops/s
Iteration  2: 195532.950 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder":
  198496.519 +-(99.9%) 15103.942 ops/s [Average]
  (min, avg, max) = (189750.611, 198496.519, 218631.739), stdev = 9990.323
  CI (99.9%): [183392.577, 213600.461] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
```

```
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder
# Parameters: (i = 50000)

# Run progress: 50.00% complete, ETA 00:01:09
# Fork: 1 of 5
# Warmup Iteration 1: 157.649 ops/s
# Warmup Iteration 2: 191.718 ops/s
Iteration  1: 200.731 ops/s
Iteration  2: 203.776 ops/s

# Run progress: 53.33% complete, ETA 00:01:05
# Fork: 2 of 5
# Warmup Iteration 1: 154.657 ops/s
# Warmup Iteration 2: 194.712 ops/s
Iteration  1: 201.891 ops/s
Iteration  2: 205.854 ops/s

# Run progress: 56.67% complete, ETA 00:01:00
# Fork: 3 of 5
# Warmup Iteration 1: 154.730 ops/s
# Warmup Iteration 2: 184.683 ops/s
Iteration  1: 198.592 ops/s
Iteration  2: 204.655 ops/s

# Run progress: 60.00% complete, ETA 00:00:55
# Fork: 4 of 5
# Warmup Iteration 1: 155.477 ops/s
# Warmup Iteration 2: 186.696 ops/s
Iteration  1: 197.661 ops/s
Iteration  2: 198.729 ops/s

# Run progress: 63.33% complete, ETA 00:00:51
# Fork: 5 of 5
# Warmup Iteration 1: 153.934 ops/s
# Warmup Iteration 2: 190.428 ops/s
Iteration  1: 204.812 ops/s
Iteration  2: 204.809 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testBuilder":
  202.151 +-(99.9%) 4.602 ops/s [Average]
  (min, avg, max) = (197.661, 202.151, 205.854), stdev = 3.044
  CI (99.9%): [197.549, 206.753] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat
# Parameters: (i = 50)

# Run progress: 66.67% complete, ETA 00:00:46
# Fork: 1 of 5
# Warmup Iteration 1: 53031.156 ops/s
# Warmup Iteration 2: 72061.441 ops/s
Iteration  1: 74622.057 ops/s
Iteration  2: 75097.545 ops/s

# Run progress: 70.00% complete, ETA 00:00:41
# Fork: 2 of 5
# Warmup Iteration 1: 54590.917 ops/s
# Warmup Iteration 2: 71989.212 ops/s
Iteration  1: 73464.031 ops/s
```

```
Iteration  2: 74692.911 ops/s

# Run progress: 73.33% complete, ETA 00:00:37
# Fork: 3 of 5
# Warmup Iteration 1: 52401.019 ops/s
# Warmup Iteration 2: 68533.864 ops/s
Iteration  1: 73458.601 ops/s
Iteration  2: 73319.080 ops/s

# Run progress: 76.67% complete, ETA 00:00:32
# Fork: 4 of 5
# Warmup Iteration 1: 53058.474 ops/s
# Warmup Iteration 2: 69837.646 ops/s
Iteration  1: 72937.215 ops/s
Iteration  2: 75301.096 ops/s

# Run progress: 80.00% complete, ETA 00:00:27
# Fork: 5 of 5
# Warmup Iteration 1: 49132.657 ops/s
# Warmup Iteration 2: 70599.258 ops/s
Iteration  1: 73262.082 ops/s
Iteration  2: 72122.081 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat":
  73827.670 +-(99.9%) 1570.538 ops/s [Average]
  (min, avg, max) = (72122.081, 73827.670, 75301.096), stdev = 1038.813
  CI (99.9%): [72257.132, 75398.208] (assumes normal distribution)


# JMH version: 1.21
# VM version: JDK 9, Java HotSpot(TM) 64-Bit Server VM, 9+181
# VM invoker: /usr/lib/jvm/java-9-oracle/bin/java
# VM options: -Dfile.encoding=UTF-8
# Warmup: 2 iterations, 1 s each, 3 calls per op
# Measurement: 2 iterations, 1 s each, 3 calls per op
# Timeout: 10 min per iteration
# Threads: 1 thread, will synchronize iterations
# Benchmark mode: Throughput, ops/time
# Benchmark: fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat
# Parameters: (i = 50000)

# Run progress: 83.33% complete, ETA 00:00:23
# Fork: 1 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 86.67% complete, ETA 00:00:18
# Fork: 2 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 90.00% complete, ETA 00:00:14
# Fork: 3 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s

# Run progress: 93.33% complete, ETA 00:00:09
# Fork: 4 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s
```

```
# Run progress: 96.67% complete, ETA 00:00:04
# Fork: 5 of 5
# Warmup Iteration 1: ~= 0 ops/s
# Warmup Iteration 2: ~= 0 ops/s
Iteration  1: ~= 0 ops/s
Iteration  2: ~= 0 ops/s


Result "fr.polytechtours.javaperformance.tp2.MyBenchmark2.testConcat":
  ~= 0 ops/s


# Run complete. Total time: 00:02:28

REMEMBER: The numbers below are just data. To gain reusable insights, you need to follow up on
why the numbers are the way they are. Use profilers (see -prof, -lprof), design factorial
experiments, perform baseline and negative tests that provide experimental control, make sure
the benchmarking environment is safe on JVM/OS/HW level, ask for reviews from the domain experts.
Do not assume the numbers tell you what you want them to tell.

Benchmark                (i)   Mode  Cnt      Score      Error  Units
MyBenchmark2.testBuffer   50  thrpt   10  170957.773 +- 52248.281  ops/s
MyBenchmark2.testBuffer 50000 thrpt   10     101.173 +-   20.841  ops/s
MyBenchmark2.testBuilder  50  thrpt   10  198496.519 +- 15103.942  ops/s
MyBenchmark2.testBuilder 50000 thrpt  10     202.151 +-    4.602  ops/s
MyBenchmark2.testConcat   50  thrpt   10   73827.670 +- 1570.538  ops/s
MyBenchmark2.testConcat 50000 thrpt   10         ~= 0             ops/s

Process finished with exit code 0
```