Java™

ORACLE®

Kotlin + JVM = Awesome

Vaibhav Choudhary (@vaibhav_c)
JVM Team, Java Platforms Team
https://blogs.oracle.com/vaibhav

Java Your Next (Cloud)

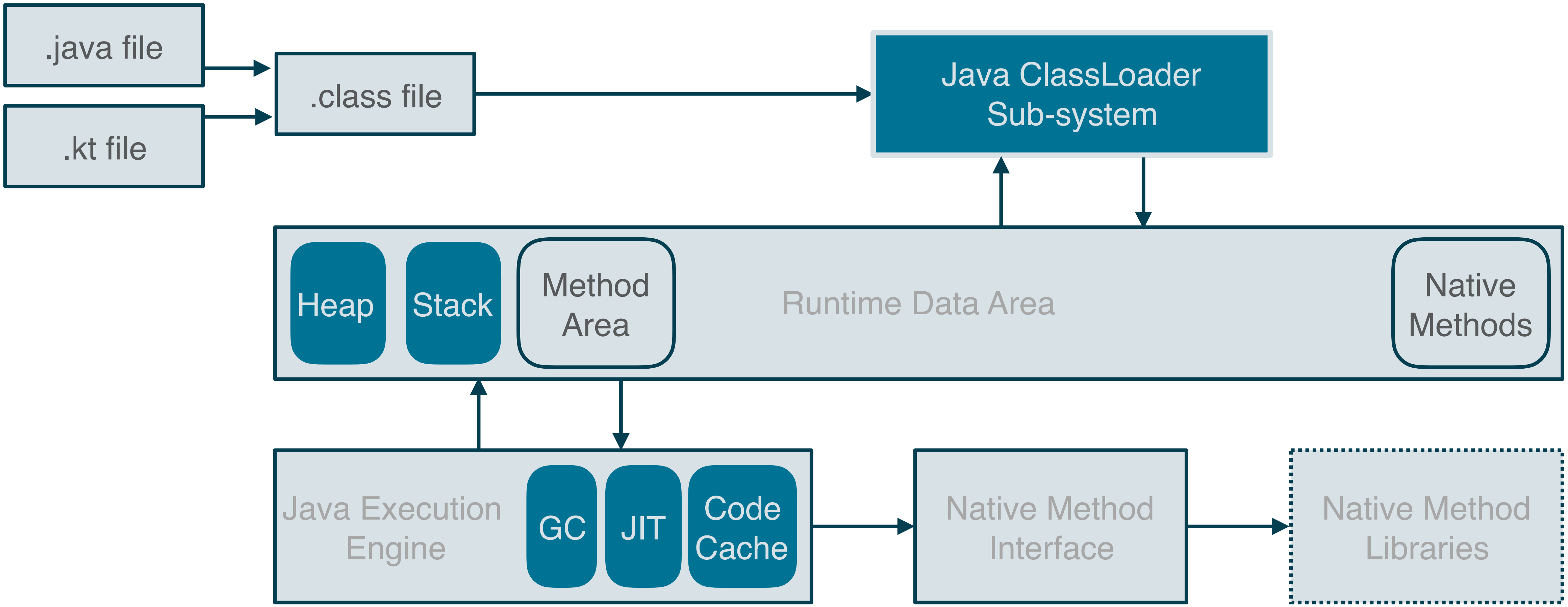
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Agenda of the day …

1. Java Virtual Machine - Architectural View
2. Java Runtime Compiler(s) and Optimizations
3. How Kotlin leverages runtime performance
4. Final Thoughts
5. Future

Java Virtual Machine - Architectural View



Beginning of Java Runtime Compilers (JIT)

Why we need Runtime Compilers ?

- Interpreter are slow in nature. How much ?
- We always have scope of optimization of runtime code.
- Can we be biased towards a path which is 99.9 percent times right ?
- With more run, we get profiled data and optimizations can be done as per the information.
- Just-In-Time (JIT) Compiler comes with 2 flavors to address all these :-
  - C1 - Less optimized, fast startup - perfect for UI (Client) app.
  - C2 - More optimized, slow startup - perfect for Server app.
- Graal - ??

In working, where I see JIT ?

- Whenever you use any of these flags :-

  - -client [C1]
  - -server [C2]
  - -XX:+TieredCompilation [Mix of C1, C2]
  - -Xint [Only interpreter]
  - -Xcomp [Only compiler]
  - -Xmixed [Mix of Interpreter and Compiler]
  - http://hg.openjdk.java.net/jdk8u/jdk8u/hotspot/file/2b2511bd3cc8/src/share/vm/runtime/advancedThresholdPolicy.hpp#l34

Optimizations_1 (Dead code removal + Expression Opt)

```
void someCalculation(int x1, int x2, int x3) {
int res1 = x1+x2;
int res2 = x1-x2;
int res3 = x1+x3;
(res1+res2)/2;
}
```

```
void someCalculation(int x1, int x2, int x3) {
x1;
}
```

Optimizations_2 (Monomorphic Dispatch)

```
public class Animal {
private String color;
public String getColor() {
return color; } }

Animal a = new Animal();
a.getColor();
```

If not overridden

```
Animal a = new Animal();
a.color;
```

Java ORACLE

Mono, Bi and Mega Morphic Calls

Can be two overridden calls only.
Can be three overridden calls only.
...
...
Can be in real a “true” Polymorphism.

```
java -XX:+PrintFlagsFinal | grep “Bimorphic"


     bool UseBimorphicInlining                              = true
{C2 product} {default}
     bool UseOnlyInlinedBimorphic                           = true
{C2 product} {default}
```

Optimizations_3_4 (Inlining and Caching)

- Inlining is one of the most powerful optimization. Method call replaced by the method body if function is pure.
- Inline is method is < 35 KB
- Inlining in Java is more powerful than languages like C++

- Caching is also a good optimization technique
  - p.x = (p1.x + p2.x) / 2 ; p.y = (p1.y + p2.y) / 2

Benchmarks

- Monomorphic: 2.816 +- 0.056 ns/op
- Bimorphic: 3.258 +- 0.195 ns/op
- Megamorphic: 4.896 +- 0.017 ns/op
- Inlinable Monomorphic: 1.555 +- 0.007 ns/op
- Inlinable Bimorphic: 1.555 +- 0.004 ns/op
- Inlinable Megamorphic: 4.278 +- 0.013 ns/op

Inline functions in Kotlin

- Inline functions are supported in Kotlin. Request the call-site to inline function.

```
fun main(args: Array<String>) {
    var a = 2
    println(someMethod(a, {println("Just some dummy function")}))
}
inline fun someMethod(a: Int, func: () -> Unit):Int {
    func()
    return 2*a
}
```

Is there any advantage with this ?

Pay-off - Explicit Inline

- Case I - explicit inlining

Benchmark	Mode	Cnt	Score		Error	Units
Monomorphic	avgt	15	3.925	±	0.151	ns/op
Bimorphic	avgt	15	4.276	±	0.187	ns/op
Megamorphic3	avgt	15	4.958	±	0.209	ns/op
Megamorphic4	avgt	15	5.229	±	0.235	ns/op

- Case I - w/o explicit inlining

Benchmark	Mode	Cnt	Score		Error	Units
Monomorphic	avgt	15	3.957	±	0.100	ns/op
Bimorphic	avgt	15	4.545	±	0.178	ns/op
Megamorphic3	avgt	15	6.189	±	0.190	ns/op
Megamorphic4	avgt	15	10.781	±	0.393	ns/op

* https://ionutbalosin.com/2019/03/kotlin-explicit-inlining-at-megamorphic-call-sites-pays-off-in-performance/

Optimizations_5 (Null Check Optimization)

- How Java handles Null Pointer ?

```
Point point = new Point();
x = point.x;
y = point.y;
```

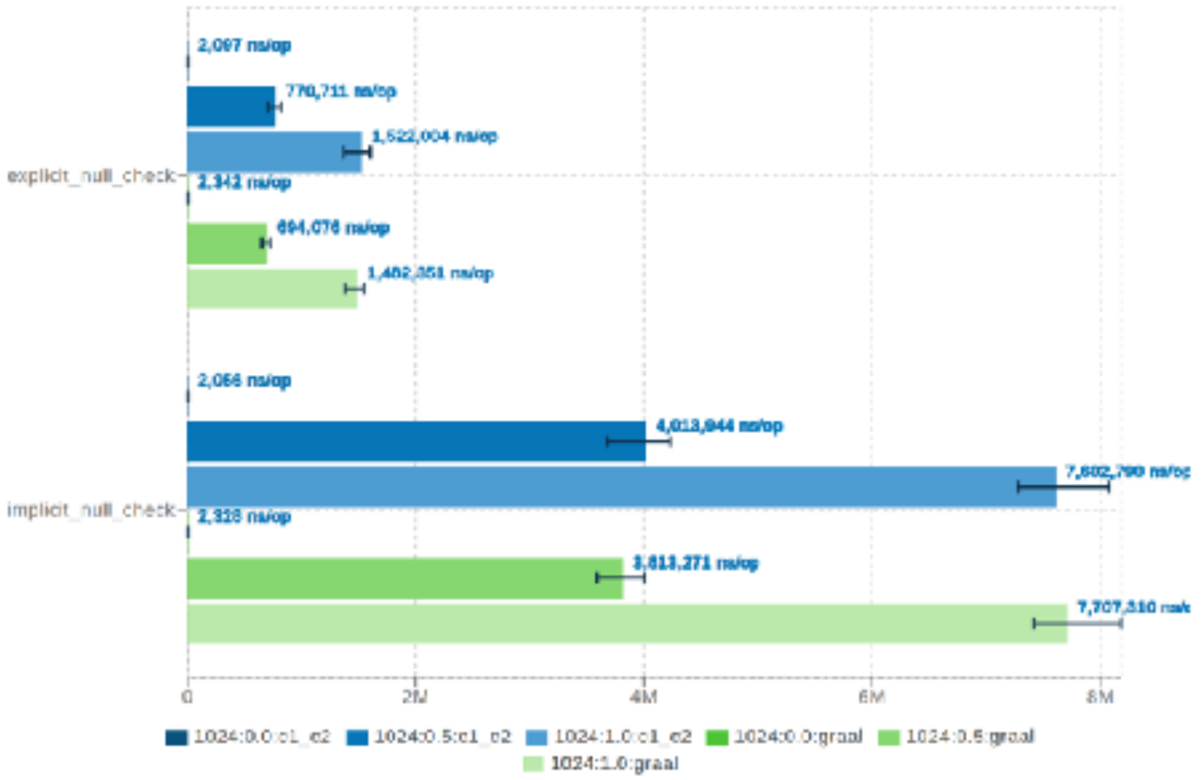
runtime conversion →

```
Point point = new Point();
if(point==null) throw NPE;
else {
x = point.x;
y = point.y;
}
```

- if profiled information says that, point is never going for null
  - remove the if-else condition
  - insert UnCommonTrap [path to de-optimization]
- http://hg.openjdk.java.net/jdk8u/jdk8u/hotspot/file/b8413a9cbb84/src/share/vm/runtime/deoptimization.cpp

Null Check Benchmarking

```
method() {
    try {
        //mode is {explicit, implicit}
        <mode>_null_check(object);
    } catch(NullPointerException e) {
        // swallow exception
    }
}

explicit_null_check(object) {
    if (object == null) {
        throw new NullPointerException("Oops!");
    }
    return object.field;
}

implicit_null_check(object) {
    return object.field; // might throw NPE
}
```



Tony Hoare words …

Apologies and retractions
Speaking at a software conference called QCon London in 2009, he apologised for inventing the null reference

I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.

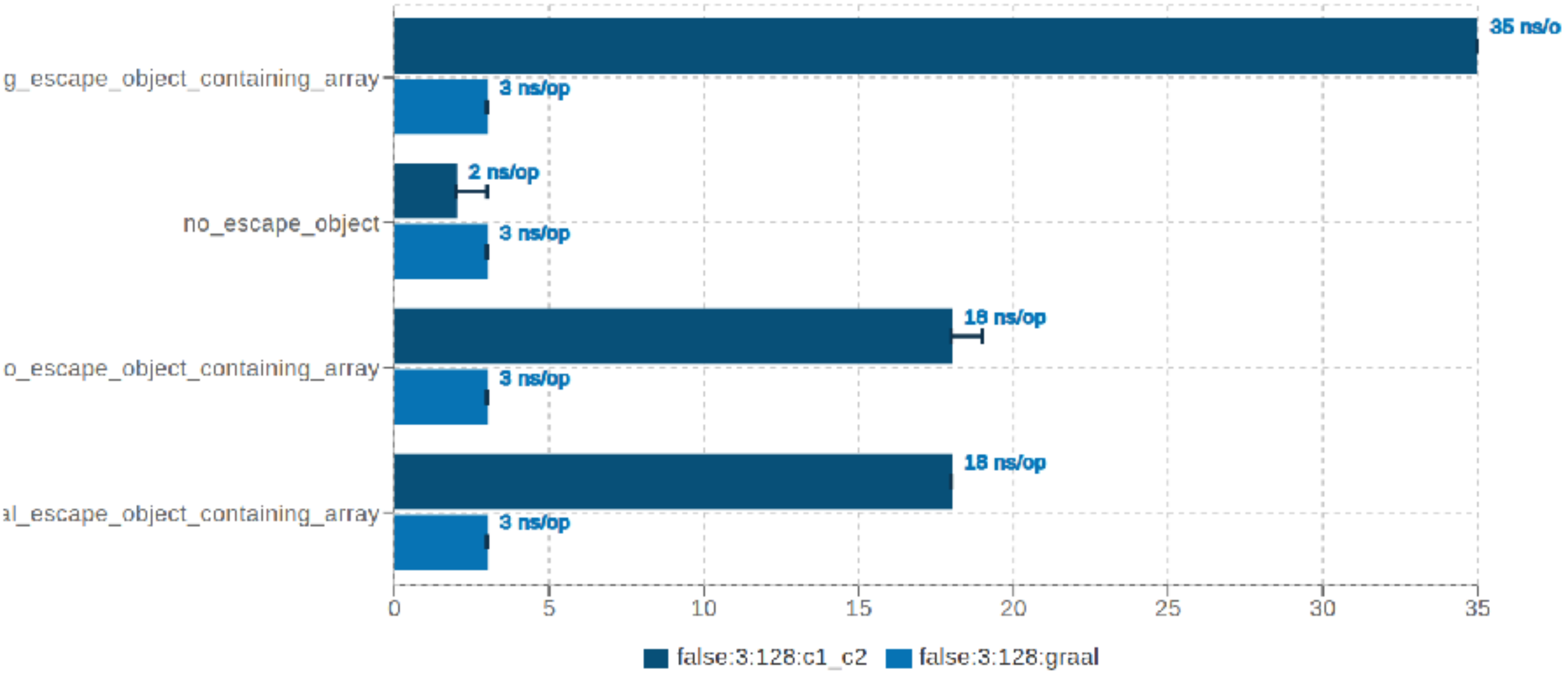
HOW KOTLIN HANDLES NULL

```
fun main(args : Array<String>) {
    var a: String = "abc"
    a = null // compilation error

    var b: String? = "abc"
    b = null // ok

    val l = a.length //ok

    val l = b.length // error: variable 'b' can be null
    val l = if (b != null) b.length else -1
}
```

Is it just a NullPointerException relief ?

Scalar Replacement

- Good Read - http://www.ssw.uni-linz.ac.at/Research/Papers/Stadler14/Stadler2014-CGO-PEA.pdf
- Compiler analyses the scope of a new object and decides whether it might be allocated or not on the heap. The method is called Escape Analysis (EA), which identifies if the newly created object is escaping or not into the heap
- Cases :-
  - NoEscape – the object cannot be visible outside the current method and thread.
  - ArgEscape – the object is passed as an argument to a method but cannot otherwise be visible outside the method or by other threads.
  - GlobalEscape – the object can escape the method or the thread. It means that an object with GlobalEscape state is visible outside method/thread.
- Flags : EliminateAllocationArraySizeLimit (default 64), EliminateAllocations (default true)



* https://ionutbalosin.com/2019/04/jvm-jit-compilers-benchmarks-report-19-04/

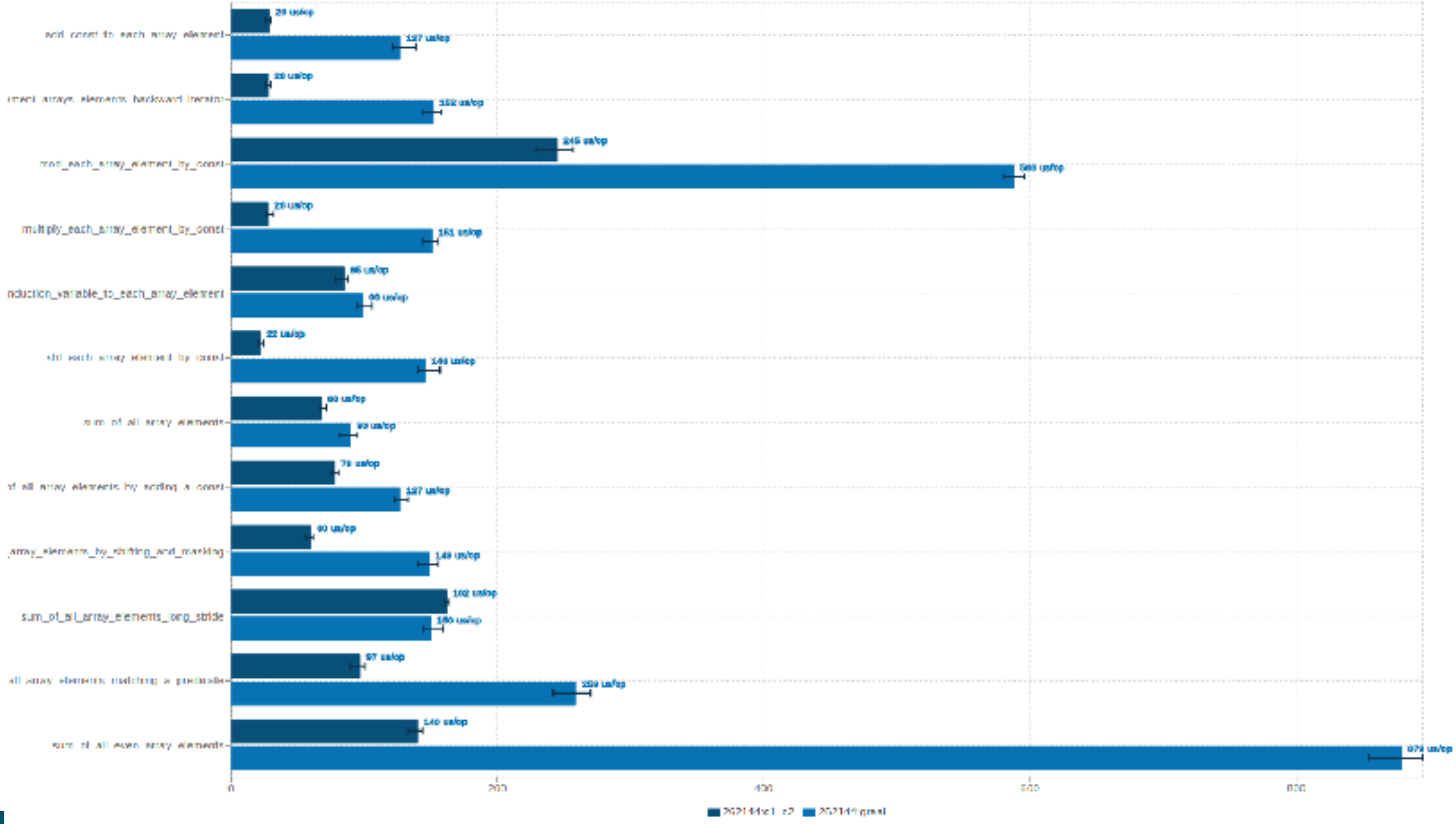
Vectorization Effect

- https://en.wikipedia.org/wiki/Automatic_vectorization
- A vectorizing compiler transforms loops into sequences of vector operations.
- Loop vectorization transforms procedural loops by assigning a processing unit to each pair of operands. Programs spend most of their time within such loops. Therefore vectorization can significantly accelerate them, especially over large data sets.

Cont …

```
int[] A;

// sum_of_all_array_elements
sum += A[i];

// sum_of_all_array_elements_by_adding_a_const
sum += A[i] + CONST;

// sum_of_all_even_array_elements
if ( (A[i] & 0x1) == 0 ) {
    sum += A[i];
}

// sum_of_all_array_elements_matching_a_predicate
if (P[i]) {
    sum += A[i];
}
```

```
// sum_of_all_array_elements_by_shifting_and_masking
sum += (A[i] >> SHIFT) & MASK;

// multiply_each_array_element_by_const
A[i] = A[i] * CONST;

// add_const_to_each_array_element
A[i] = A[i] + CONST;

// shl_each_array_element_by_const
A[i] = A[i] << CONST;

// mod_each_array_element_by_const
A[i] = A[i] % CONST;
// saves_induction_variable_to_each_array_element
A[i] = i;
```



multiply_each_array_element_by_const
nduction_variable_to_each_array_element
sum_of_all_array_elements_long_stride
127 us/op
245 us/op
151 us/op
22 us/op
97 us/op
140 us/op
0
200
400
600
800

Optimizations_6_7 (Thread and Loop Optimization)

- Eliminate lock if not reachable from other thread.
- Join sync blocks on the same object, if possible.
- Biased Locking

- Loop combining - if possible
- Replacing while from do-while
- Tiling Loops - Fitting into the cache exactly

Real life challenges and Future

- Higher start-up time - Not a good idea in low latency systems.
- Different techniques.

- Future projects (AOT, AppCDS …)

References

- Bangalore JUG Downloads : http://bangalorejug.org/Downloads
- Ionut Balosin Blog: https://ionutbalosin.com/
- Kotlin Language : https://kotlinlang.org