Bc running on Truffle

Author : Sylvain Julmy

Supervisors : Boris Spasojević, Manuel Leuenberger

May 15, 2019

Project statement

The goal of this project is to implement **bc** as a Truffle languages to serve the purpose as a simple language to introduce Truffle and improving bc's performance.

bc

bc (**b**asic **c**alculator) is a language that support arbitrary precision number and a syntax close to the C programming language.

```
sum = ( 886612897528752 8 ^ 3) +\
      (-877840544286223 9 ^ 3) +\
      (-273611146880704 0 ^ 3)
print sum, "\n"
halt
```

Table of Contents

1. GraalVM and Truffle
2. Implementing bc with Truffle
3. Performance
4. Conclusion

GraalVM

GraalVM is a universal virtual machine for running applications written in various languages[1].
It aims to :

- match performance of JVM languages with native languages.
- allow freeform mixing of programming languages (polyglot applications).
- include a set of "polyglot programming tools".

[1] https://www.graalvm.org/

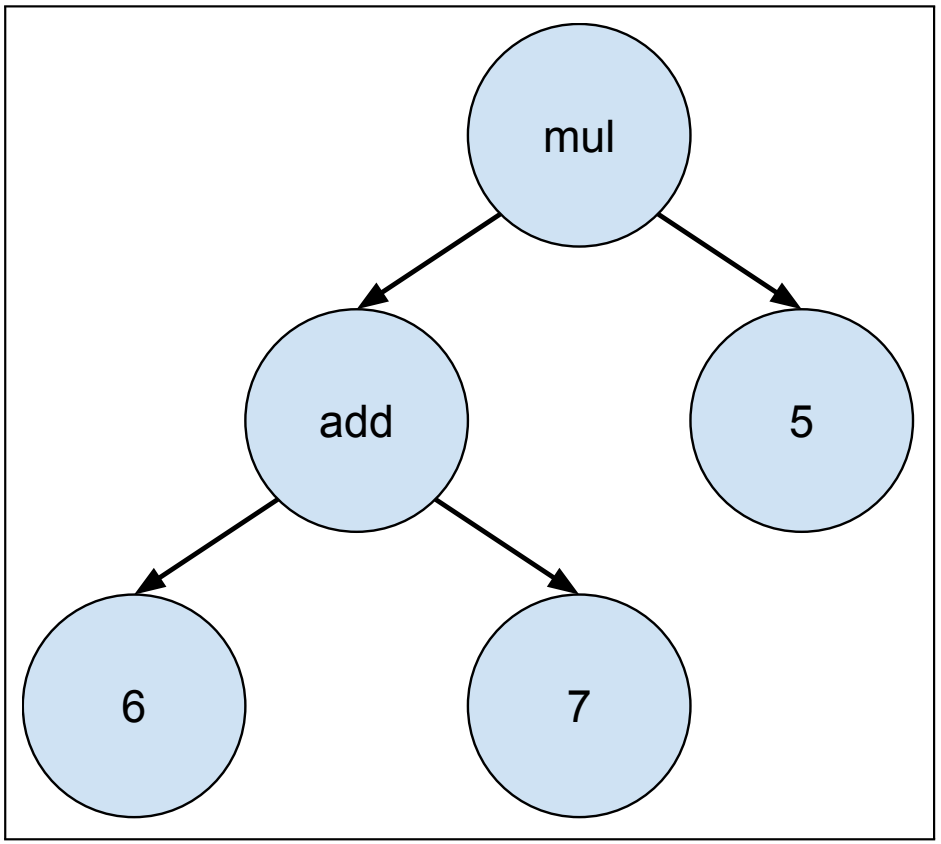
Truffle

Truffle is a framework for building programming languages as interpreters for self-modifying AST.

AST Interpreter

```
class MulNode extends Node {
  Node left;
  Node right;
  int execute() {
    return left.execute() *
           right.execute();
  }
}
```

Each node compute his own operation.



Simple AST

Table of Contents

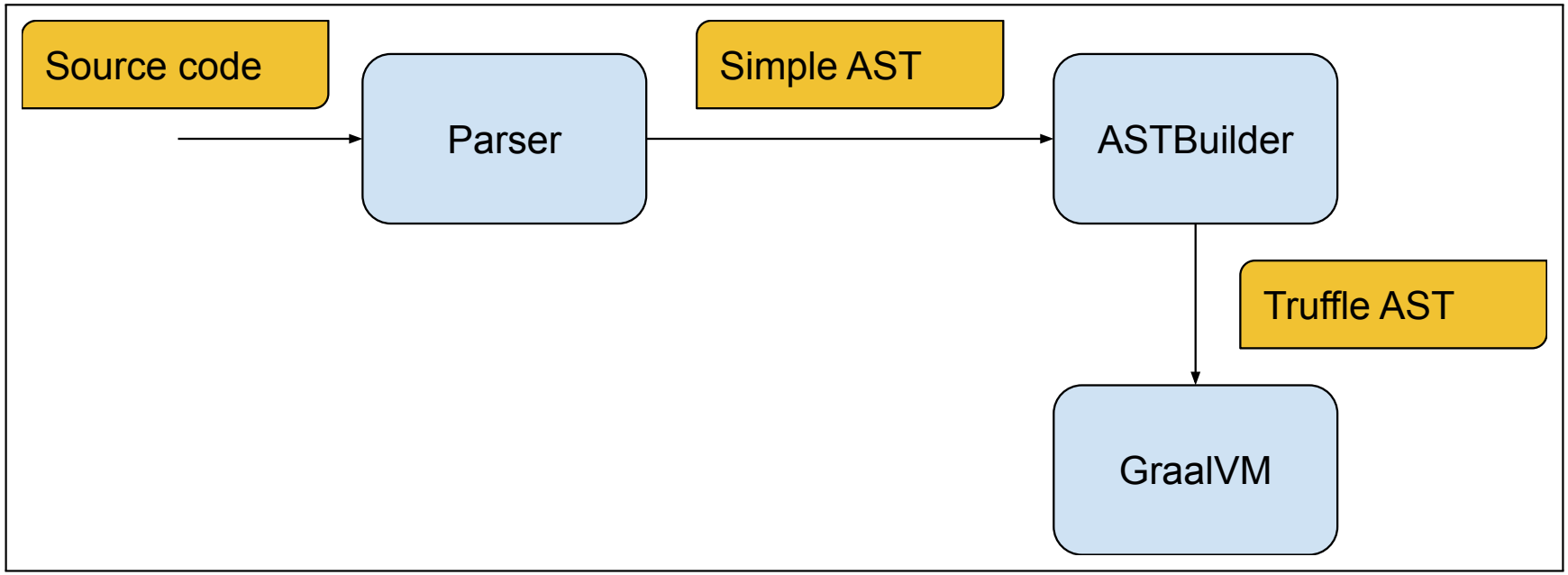
1 GraalVM and Truffle

2 Implementing bc with Truffle

3 Performance

4 Conclusion

Global architecture



bc-truffle architecture.

Parser

Parser is implemented in Scala using parser-combinator and produce an intermediate AST.
The AST is then visited to produce the Truffle AST.

Example : Add node

```
@NodeChildren({
        @NodeChild(value = "left", type = BcExpressionNode.class),
        @NodeChild(value = "right", type = BcExpressionNode.class)
})
public abstract class BcBinaryNode extends BcExpressionNode {}
```

Example : Add node

```
public abstract class BcAddNode extends BcBinaryNode {

  @Specialization(rewriteOn = ArithmeticException.class)
  protected long add(long left, long right) {
      return Math.addExact(left, right);
  }

  @Specialization
  protected BcBigNumber add(BcBigNumber left, BcBigNumber right) {
    return left.add(right);
  }

  @Specialization
  protected String doString(Object left, Object right) {
    return left.toString() + right.toString();
  }

  @Fallback
  protected Object typeError(Object left, Object right) {
    throw BcException.typeError(this, left, right);
  }
}
```

Table of Contents

1 GraalVM and Truffle

2 Implementing bc with Truffle

3 Performance

4 Conclusion

bc-truffle vs. Java vs. bc

Here is a simple program which multiply big number :

```
import java.math.BigDecimal;
class Bignumber {
  public static void main(String[] args) {
    BigDecimal y = new BigDecimal(1);
    for(int i=1; i<500_000;i++) {
      y = y.multiply(new BigDecimal(i));
    }
    System.out.println(y);
  }
}
```

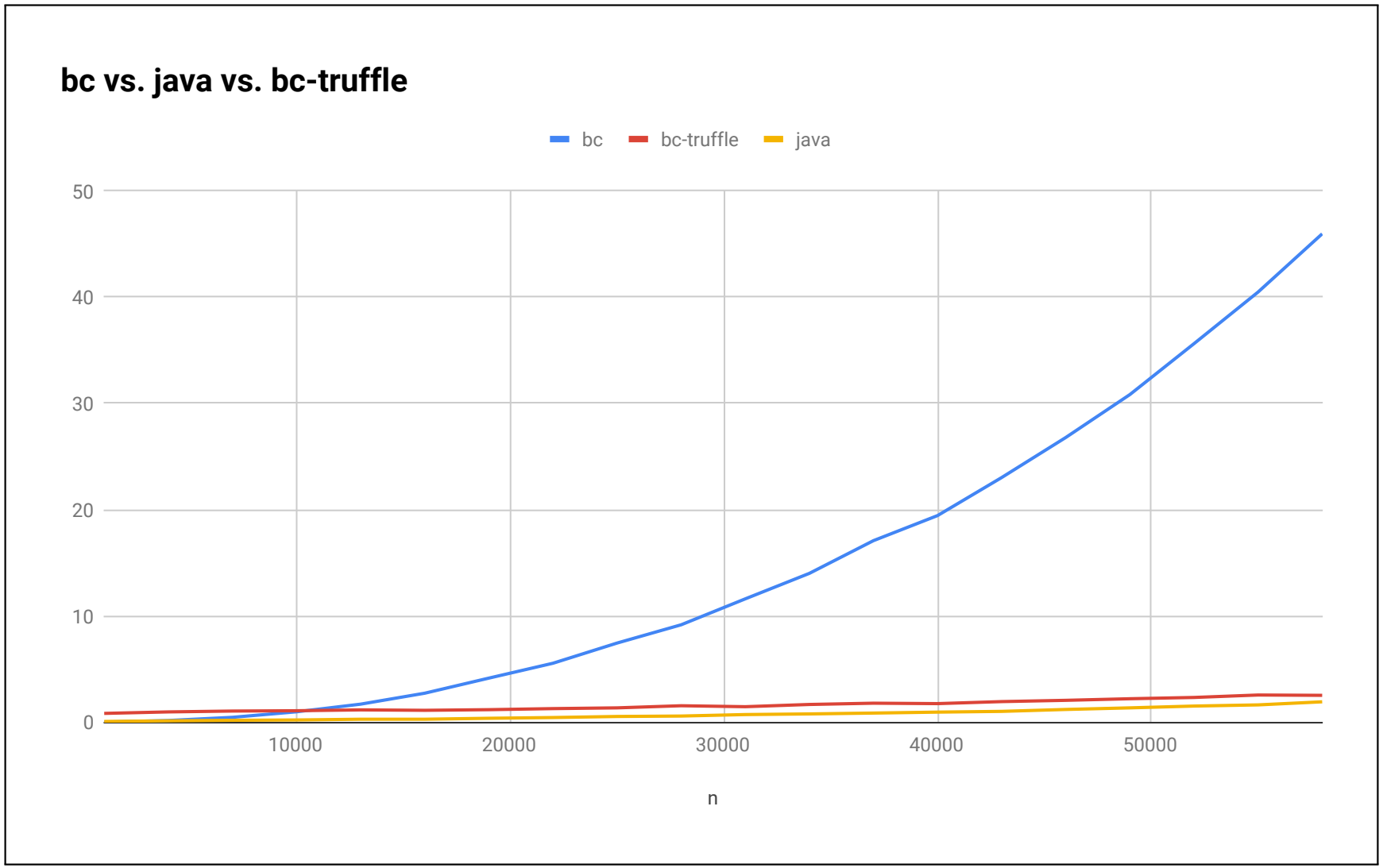
```
y = 1
for(i=1;i<500000;i++)
  y *= i
y
halt
```

Java time : 103s

bc-truffle : 105s

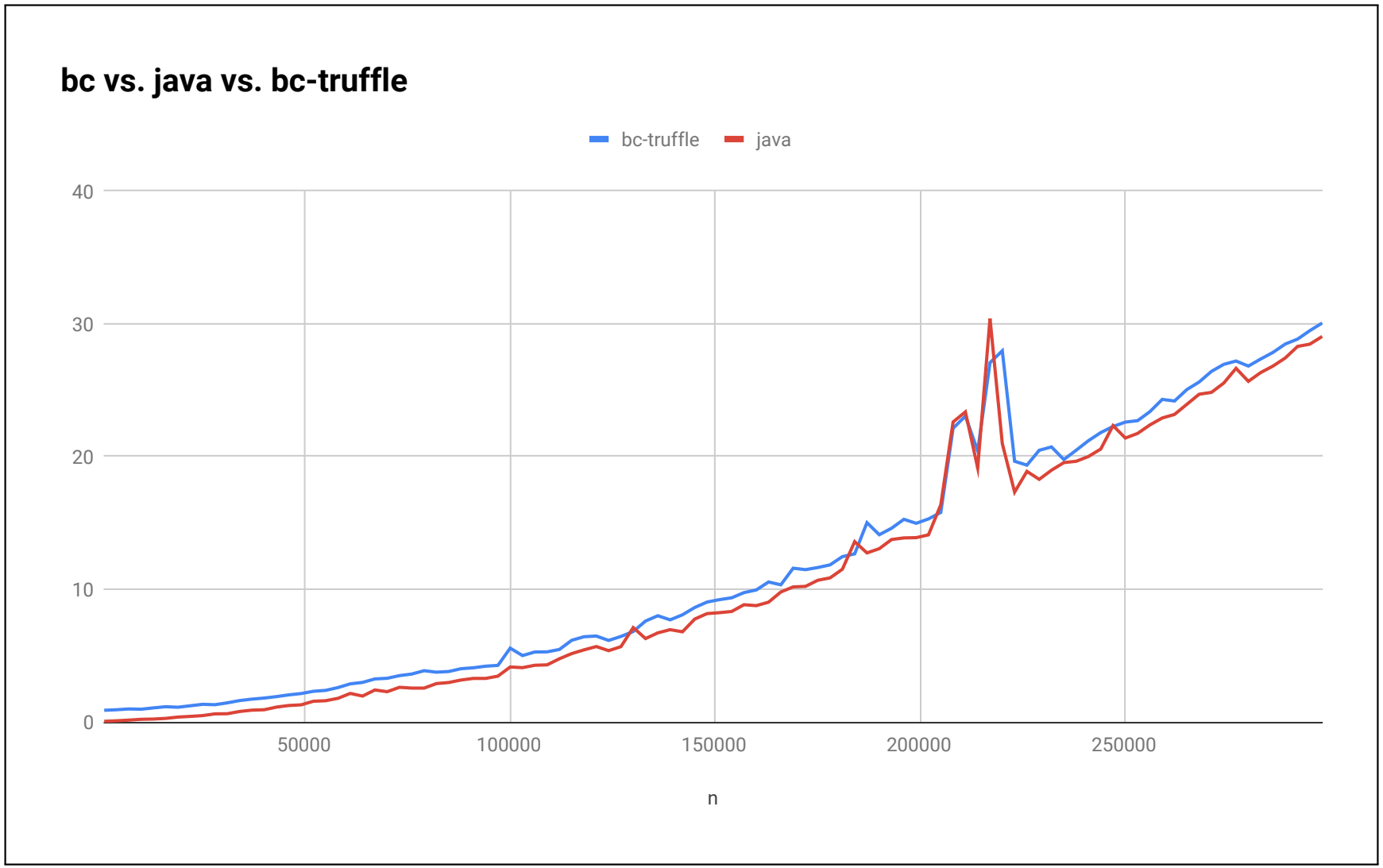
bc : 4213s

bc-truffle vs. Java vs. bc



bc-truffle vs. Java vs. bc

bc-truffle vs. Java



bc-truffle vs. Java vs. bc

Native image

The native image allows to ahead-of-time compile Java code to a standalone executable.

Table of Contents

1 GraalVM and Truffle

2 Implementing bc with Truffle

3 Performance

4 Conclusion

Using Truffle

- Simple language is a good introduction to understand some basic concept.
- Read some Truffle paper.
- Blog post about implementing a Lisp.
- Look at the others implementation.

Improvement

- Support all the bc's extensions (i.e. GNU bc).
- Add tool support.
- Add more opportunity for optimization.
- Support interoperability for polyglot applications.
- Find and fix bugs !

Links

Github repository :
`https://github.com/SnipyJulmy/bc-truffle`