DOI 10.26886/2414-634X.3(47)2021.9

UDC: 621.39

GRAAL AS A MULTILINGUAL PLATFORM

Maxim Bartkov

https://orcid.org/0000-0003-3527-3642

e-mail: 1233566789b@gmail.com

RooX Solutions Java Team Lead, Ukraine, Khakov

The variety of software environments allows everyone to choose what suits the creation of a specific program, application, etc. GraalVM is software that provides a significant improvement in program performance and efficiency, ideal for creating IT products. It is designed for programs written in Java, JavaScript, LLVM-based languages such as C and C ++ and other dynamic languages. It eliminates isolation between programming languages and provides compatibility in a common runtime environment. It can work alone or in the context of OpenJDK, Node.js or Oracle Database. The paper notes that high-performance virtual machines (VMs) such as Java, HotSpot VM or V8 JavaScript VM correspond to features that were first implemented for SELF language: a multilevel optimization system with adaptive optimization and deoptimization. It is noted that Java has a system of many very high-quality libraries that are often not available in other systems, including native programs and other managed languages.

Keywords: high performance virtual machines, application performance and efficiency, JavaScript VM, GraalVM

Introduction. Most high-performance dynamic speech virtual machines use a duplication of language semantics in the interpreter, compiler, and runtime systems. This places an extra burden on the volume and runtime of the programs. Therefore, a universal interpreter in a single

compiler should be used. The interpreter performs special functions: it supplements the interpreted program with information about the type about profiling [1]. The created code is obtained automatically using partial evaluation when these functions are included. This allows partial evaluation to be performed in the context of dynamic languages: it reduces the size of the compiled code while compiling all parts of the operation that are relevant to a given program. When it fails to generate a file, the execution is passed back to the interpreter, the program goes through partial evaluation again at the interpreter, transforms the new file into compiled code again [6].

High-performance virtual machines (VMs) such as the Java, HotSpot VM or V8 JavaScript VM correspond to features that were first implemented for SELF language: a multi-level optimization system with adaptive optimization and deoptimization. The first level of execution, usually the interpreter, or fast compiler of the underlying compiler, provides a fast start. The second level of execution, the dynamic compiler generates optimized machine code for multi-execution code, which means that good performance is ensured. Deoptimization moves execution from the second level back to the first level, that is, it replaces some optimized code with some non-optimized code when the dynamic compiler runs longer. Multiple levels increase the cost of implementing and maintaining a virtual machine. Although the action of the interpreter or base compiler lower complexity optimizing compiler, their implementation is far from trivial, in addition, a new architecture is needed [3, 7]. But essentially, language semantics must be implemented several times in different styles: for the first level language operations are usually implemented as assembly code templates [2]. For the second level, language operations are implemented as an intermediate image of the compiler for a particular language. And for the runtime system, language operations are implemented in C or C ++.

Oracle Corporation has developed GraalVM based on HotSpot/OpenJDK implemented in Java [5]. Creating a Java virtual machine on Java itself is quite a difficult task. Creating a compiler, on the other hand, was possible to implement multilingual programs. The GraalVM compiler is a modern Java compiler.

The first version of GraalVM 19.0 appeared in May 2019, and the next GraalVM 20.0.0, GraalVM 20.1.0 and GraalVM 20.2.0 in 2020 [4].

Purpose and objectives of the article. The variety of software environments allows everyone to choose what suits the creation of a specific program, application, etc. However, there are projects that can be implemented using multiple programming languages. So, below in the article we will consider GraalVM as a multilingual environment for implementing projects.

Theoretical Background. GraalVM is a runtime environment that provides better performance and efficiency of applications. The compiler is designed for applications in Java, JavaScript, C and C ++, and other dynamic languages, and provides interoperability in one runtime environment.

The main purpose of GraalVM compiler is, first, to improve performance, reduce launching time of programs and their execution based on JVM technologies and directly ability to combine code from any programming language into one program.

Java Virtual Machine has the cost to run and the value of the program. GraalVM can create its own images for existing JVM programs. And the image generation process uses static analysis: find the code, perform full compilation ahead of time. The resulting file will contain the whole program for execution. The binary file can additionally include the compiler at a specific time to run other languages. For additional performance create your

own images using optimized managed profiles obtained by running previous programs.

Let's characterize, GraalVM compiler with Node.js programs, the benefits are the use of Java, R or Python libraries. JavaScript included in the standard Node.js, tuned to browser configurations and designed to work efficiently in small scripts. GraalVM allows you to work with significant volumes: sizes up to 32 GB with 32-bit compression, and large supports in 64-bit configurations. The GraalVM compiler effectively combines native code with JavaScript code. There is direct access to the native code, and the compiler allows you to build in any structures. This can be used in a scenario where effective data structures are managed and distributed in C, while other parts of the program are written in Node.js. Besides the standard GraalVM benefits, such as language interoperability (e.g., using Java or JavaScript from those programs), GraalVM can achieve high speedups for those languages. GraalVM is designed to be embedded and can run on databases. GraalVM is an open ecosystem where it is possible to connect your own programming language, tools, or platform. GraalVM is a high-performance JDK compiler. It is designed to accelerate the execution of programs written in Java and other JVM languages, while providing execution for JavaScript, Ruby, Python, and several other popular languages. GraalVM's multilingualism gives you the ability to combine multiple programming languages in a single application while eliminating any costs. GraalVM Community Edition consists of 3.6 million lines of code. So, the main GraalVM tools are: the GraalVM compiler is written in Java and supports both dynamic and static compilation; Truffle, a Truffle language implementation to create languages and tools for GraalVM; Substrate VM, a framework that lets you compile Java programs; and Sulong, to run LLVM bit code on GraalVM. Tools: GraalJS - implementation of GraalVM JavaScript (compatible with ECMAScript 2020) and Node.js

v12.18.0; FastR - implementation of GraalVM programming language 3.6.1; GraalPython - implementation of GraalVM programming language Python 3.8; TruffleRuby - implementation of GraalVM programming language Ruby 2.6. 6; SimpleLanguage, a simple demonstration language for GraalVM; VisualVM, a visual tool that integrates JDK command line tools; VS Code, a Visual Studio Code extension that supports polyglot application development with GraalVM.

A logical question arises, but how to work with GraalVM? Consequently, it is necessary to install GraalVM according to the installed operating system.

For example, GraalVM Community Edition based on OpenJDK 11:

```
java -version
openjdk version "11.0.11" 2021-04-20
OpenJDK Runtime Environment GraalVM CE 21.1.0 (build 11.0.11+5-jvmci-21.1-b03)
OpenJDK 64-Bit Server VM GraalVM CE 21.1.0 (build 11.0.11+5-jvmci-21.1-b03, mixe

js -version
GraalVM JavaScript (GraalVM CE Native 21.1.0)

lli --version
LLVM 10.0.0 (GraalVM CE Native 21.1.0)
```

By downloading and installing GraalVM, you can run applications based on Java, JavaScript and LLVM.

GraalVM also supports running Node.js programs. Node.js support is not installed by default, but it can be added surprisingly easily with GraalVM Updater:

```
gu install nodejs
node -v
v14.16.1
```

To set up support for the LLVM toolchain:

```
gu install llvm-toolchain
export LLVM_TOOLCHAIN=$(lli --print-toolchain-path)
```

With GraalVM you can run Python programs in a Python 3 runtime environment.

```
gu install python
```

Ruby is not available by default in GraalVM, but you can add it with the GraalVM upgrade tool:

```
gu install ruby
```

GraalVM provides a GNU-compatible environment for running R programs directly or in REPL mode.

```
gu install R
```

With GraalVM it is possible to run programs compiled in WebAssembly.

```
gu install wasm
```

GraalVM allows you to call one programming language to another and exchange data between them. GraalVM provides polyglot for compatibility.

Running js --jvm --polyglot example.js runs example.js in polyglot context.

It is worth noting the benefits of GraalVM. This way, the GraalVM JIT compiler is automatically used when running the java command included in GraalVM - no additional configuration is needed. The time command allows you to get the actual time elapsed to run and execute the whole program from start to finish, instead of having to configure a complicated Microtest. GraalVM is written in Java, not C ++ like most other JIT compilers for Java. This allows it to improve faster than existing compilers, thanks to powerful new optimizations such as partial output analysis, which is not available in standard JIT compilers for HotSpot. This makes Java applications much faster. An example is Twitter, which is one company that uses GraalVM in

production today. Twitter uses GraalVM to run Scala programs - GraalVM works at the JVM bytecode level, meaning it works for any JVM language. The strength of the Java platform is evident in long-running processes and peak loads, but short-running processes can suffer from longer startup times and large amounts of memory. GraalVM gives us a tool that solves this problem. GraalVM is a library-compiler and can be used in several ways. One is to compile ahead-of-time instead of just-in-time for normal compilers. The executable will run an order of magnitude faster and use an order of magnitude less memory than running the same program on the JVM. So GraalVM is a way to use existing Java applications, but with low memory usage and fast startup. It also frees up configuration issues, such as finding the right jar files at runtime, and allows you to have smaller Docker images. Also, Java, GraalVM including new implementations of JavaScript, Ruby, R and Python. They are written using a new framework called Truffle, which allows you to create language interpreters that are both simple and high-performance. When a language interpreter is written with Truffle, it will automatically use Graal to provide JIT compilation of the language. So, GraalVM is not only a JIT compiler and ahead of its own compiler for Java, but it can also be a JIT compiler for JavaScript, Ruby, R and Python. The languages in GraalVM aim to replace your existing languages. The execution engines of different languages in GraalVM work together - there is an API that allows you to run code from one language on a program written by another. This allows you to write multilingual programs written in multiple languages. For example, most of the program is written in one language, but it is possible to use a library written in another programming language with better data processing. With GraalVM you can run programs written in different languages and use modules from these languages together in one program. Another language that GraalVM supports is C. GraalVM can run C code as well as languages such as

JavaScript and Ruby. With GraalVM it is possible to run programs written in platform-dependent languages such as C and C ++, and to run C extensions to languages such as Python and Ruby, which cannot be implemented in JVM, unless programs written in JRuby are executed. If you program in Java, you use quality tools such as IDEs, debuggers, and profilers. Not all languages have such tools, but you can get them if you use the language in GraalVM. All GraalVM languages (except Java at the moment) are implemented using a common Truffle structure. This allows you to implement features such as debuggers once and make them available to all languages. The Truffle framework is a Nexus for languages and tools. If you program your language engine with Truffle, and you program your tools like this debugger with the Truffle API for tools, then every tool works with every language on Truffle, and you only need to write that tool once. GraalVM is a platform for creating high-quality tools for programming languages that don't have a good toolkit of their own. With Truffle and GraalVM, you can use other development tools, such as the tools in Chrome Debugger or VisualVM. All languages and tools can be used separately or together in the case of multilingual programs, these languages and tools can be added to a Java application. The new API in org.graalvm.polyglot package allows code in other languages to be loaded and run, and value from them. GraalVM is a single interface for embedding language code into a Java application. The Polyglot API allows you to take guest language objects and use them as Java interfaces and other complex interactions. GraalVM already includes one built-in library built from platform-dependent applications - it is a library that allows you to run code written in any GraalVM language from platform-dependent applications. GraalVM allows you to use any language in an embedded context in this way by linking to its Polyglot library. GraalVM - using a single library in an application to embed any GraalVM language. Java has a system of many

very high-quality libraries that are often not available in other systems, including native programs and other managed languages. If you use a Java library from your own application, you can embed the JVM, but it gets big and complicated. GraalVM allows you to take a Java library, either off-the-shelf or written in-house, and assemble it into a separate platform-dependent library for use with other languages. GraalVM gives you the ability to compile Java code into your own library, which you can then use in your own programs without using the full JVM. One application for the Polyglot library is the use in the Oracle database. To do this, the library was used to create the Oracle Database Multilingual Engine (MLE), which includes support for using GraalVM languages and modules with SQL. GraalVM makes it possible to run the languages supported by GraalVM inside the Oracle database. You can use logic from the front-end or back-end inside the database instead of always pulling data from the database to the application server. Truffle is a Java library that helps you write an abstract syntax tree (AST) interpreter. The AST interpreter is the easiest way to implement the language because it works directly with the parser's source data and does not involve interpreting bytes of code or compiling, but this approach is slow. Therefore, combining it with a technique called partial computation allows Truffle to use GraalVM to automatically JIT compile the language, based only on AST interpreter source data. Used by Truffle to create its own programming language, for a high-performance implementation of an existing programming language, or DSL. So, Truffle is a simple way to implement a programming language for GraalVM.

Conclusions. So, GraalVM provides a very wide range of new functionality - it is a platform on which you can create more powerful languages and tools and place them in more environments. The latter allows you to choose the language and modules you want, regardless of where the program runs or what languages it is written in. It is worth noting

that the benefits of GraalVM extend in the following aspects: fast execution of Java; start-up and memory usage time for Java is reduced; it is possible to combine JavaScript, Java, Ruby and R; execute programs written in platform-dependent languages; common tools for all programming languages are extended repeatedly, as are JVM applications; support for multiple programming languages in the database and creating programming languages for GraalVM is provided.

References:

1. Kozlov, V. (2021). JEP 410: Remove the Experimental AOT and JIT Compiler. https://openjdk.java.net/jeps/410
2. GraalVM. (n.d.). Retrieved July 19, 2021, from Graalvm.org website: https://www.graalvm.org/
3. All posts in a row / Habr. (n.d.). Retrieved July 19, 2021, from Habr.com website: https://habr.com/
4. Bonetta, Daniele. "GraalVM: metaprogramming inside a polyglot system (invited talk)." *Proceedings of the 3rd ACM SIGPLAN International Workshop on Meta-Programming Techniques and Reflection*. 2018.
5. Šelajev, O. (2020, March 4). A look at GraalVM 20.0: better Windows support, better Native Images, better tooling. Retrieved July 19, 2021, from graalvm website: https://medium.com/graalvm/a-look-at-graalvm-20-0-better-windows-support-better-native-images-better-tooling-4fabc1227a48
6. Šipek, M., Mihaljević, B., & Radovan, A. (2019, May). Exploring Aspects of Polyglot High-Performance Virtual Machine GraalVM. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)* (pp. 1671-1676). IEEE.
7. Stadler, L., Welc, A., Humer, C., & Jordan, M. (2016). Optimizing R language execution via aggressive speculation. *Proceedings of the 12th Symposium on Dynamic Languages*. New York, NY, USA: ACM.

Citation: Maxim Bartkov (2021). GRAAL AS A MULTILINGUAL PLATFORM. New York. TK Meganom LLC. Innovative Solutions in Modern Science. 3(47). doi: 10.26886/2414-634X.3(47)2021.9