

Projekat GraalVM

Seminarski rad u okviru kursa
Metodologija stručnog i naučnog rada
Matematički fakultet

Bojan Bardžić, Milica Gnjatović, Pavle Savić, Andrija Urošević
kontakt email prvog, drugog, trećeg, `mi18083@alas.matf.bg.ac.rs`

22. novembar 2022.

Sažetak

U ovom tekstu je ukratko prikazana osnovna forma seminarskog rada. Obratite pažnju da je pored ove .pdf datoteke, u prilogu i odgovarajuća .tex datoteka, kao i .bib datoteka korišćena za generisanje literature. Na prvoj strani seminarskog rada su naslov, apstrakt i sadržaj, i to sve mora da stane na prvu stranu! Kako bi Vaš seminarski zadovoljio standarde i očekivanja, koristite uputstva i materijale sa predavanja na temu pisanja seminarskih radova. Ovo je samo šablon koji se odnosi na fizički izgled seminarskog rada (šablon koji *morate* da koristite!) kao i par tehničkih pomoćnih uputstava. Pročitajte tekst pažljivo jer on sadrži i važne informacije vezane za zahteve obima i karakteristika seminarskog rada.

Sadržaj

1	Uvod	2
1.1	Istorijski razvoj	3
2	Kompanije koje koriste GraalVM	3
3	Šta obuhvata projekat?	4
3.1	GraalVM kao zamena za JVM	4
3.2	Truffle - kompajler za kompajlere	4
3.3	Izvršavanje mašinskog koda na kaludu	5
4	Karakteristike	5
4.1	Visoke performanse	5
4.2	Native Image	7
4.3	Poliglot programiranje	7
4.4	Napredni alati	7
5	Zaključak	7
	Literatura	7



Slika 1: Podržani jezici

1 Uvod

GraalVM je alat koji omogućava pisanje i izvršavanje koda u različitim jezicima. GraalVM podržava naredne jezike:

- Java
- JavaScript i Node.js
- Python
- Ruby
- R
- LLVM jezici poput C-a i C++-a
- WebAssembly

GraalVM je implementiran u javi.

Ovaj alat je koristan u radu sa mikroservisnom arhitekturom. Nekolicina okvira za rad sa Java mikroservisima je već prihvatila ovu platformu. Između ostalih to su Micronaut, Spring, Helidon i Quarkus.

Još jedna od mogućnosti koje GraalVM nudi je implemenitriranje novih jezika i alata korišćenjem biblioteke Truffle.

Korišćenjem ovog alata je omogućeno efikasnije korišćenje više jezika na jednom projektu. U projektu je moguće kodom jednog jezika pozivati funkcije pisane u drugom jeziku. Dopušteno je deljenje struktura podataka između kodova pisanih u različitim jezicima. Na ovaj način je omogućeno da sakupljač otpadaka radi na celom projektu bez obzira na to koliko različitih jezika je korišćeno. Ovim je omogućeno i jednostavnije debugovanje.

GraalVM čine Java Virtuelna Mašina - JVM i Java Development Kit - JDK. Ovaj alat je proizvod kompanije Oracle. Osnovni cilj projekta je omogućiti brže izvršavanje koda, a da se pri tome koristi manje memorije.

Ovaj alat je dostupan na Linux, Windows i macOS operativnim sistemima.

Dostupna su dva izdanja GraalVM-a:

Community edition je otvorenog koda i dostupno je u [github repozitorijumu](#). Korisnici mogu doprineti razvoju ove verzije kreiranjem githb issue-a i pravljenjem pull request-ova.

Enterprise edition razvija i licensira kompanija Oracle.

GraalVM podržavaju razvojna okruženja i protokoli za debugovanje. Neka od podržanih okruženja su Eclipse, NetBeans, IntelliJ IDEA i Visual Studio Code. Ova okruženja su posebno dobra jer podržaju sve jezike koje podržava i GraalVM. Ovaj alat obezbeđuje ugrađen Crome DevTools Protocol, Debug Adapter Protocol (DAP) i Language Server Protocol (LSP), čime je omogućeno debugovanje JavaScript, R i Ruby kodova.

1.1 Istorijski razvoj

Java Virtuelne mašine poput Oraklovog Java HotSpotVM i IBMov Java VM postoje već 20ak godina. Međutim nije postojala virtuelna mašina koja bi omogućila efikasno izvršavanje kodova pisani u različitim jezicima. Cilj ovog projekta je bio da se napavi objedinjena virtuelna mašina koja bi ovo omogućila.

GraalVM 19.0 je objavljen u Maju 2019e i to je prva verzija ovog alata. Trenutno najnovija verzija je GraalVM 22.1.0 objavljena u Aprilu 2022.

2 Kompanije koje koriste GraalVM

Facebook

Društvena mreža Facebook ima ogroman broj korisnika zbog čega je jako bitno da se kod dobro skalira i brzo izvršava. Na serverskoj strani koristi Javu i Spark framework za rad sa velikom količinom podataka. Prelazak sa Oracle JDK-a i Open JDK-a na GraalVM se sveo samo na promenu runtime okruženja, bez ikakvih promena u kodu. Izmereno je prosečno ubrzanje 1.1x korišćenjem Community verzije i ubrzanje 1.42x korišćenjem Enterprise verzije GraalVM-a.

Twitter

Kao i Facebook, Twitter je društvena mreža sa milionima korisnika. Sa porastom broja korisnika bile su neophodne promene koje bi omogućile brzo izvršavanje uz minimalne troškove. Većina mikroservisova tvitera je implementirano u Skali. Prelaskom na GraalVM je omogućeno 8-11% ušteda na procesoru.

Standard Chartered Bank

Ova kompanija je između ostalog koristila Javu, Python sa Spring-Boot okvirom kako bi obezbedila stabilnost i robusnost programa. Pre par godina kompanije je odlučila da svoju aplikaciju prebaci u cloud, što je donelo nove izazove. GraalVM je obezbedio jednostavan prelazak na cloud time što podržava tehnologije koje su programi već koristili. GraalVM je omogućio skalabilnost i ubrzanje od 7%.

Goldman Sachs

Goldman Sachs je investiciona banka. Ova kompanija ima svoj programski jezik Slang koji ima funkcije pogodne za njiove potrebe. Jezik je dosta glomazan i potrebno mu je unapređenje. Prevođenje celog koda u neki drugi jezik bi bilo previše zahtevno. GraalVM i Truffle su omogućili unapređenje i ubrzanje izvršavanja koda pisanog u Slangu, a da je pri tom napisana minimalna količina novog koda.

Nvidia

Nvidia je jedan od najvećih proizvođača grafičkih kartica. Iako neki jezici jednostavno mogu da ubrzaju izvršavanje korišćenjem grafičke karte, za neke druge to može biti izazovno. GraalVM i Truffle su omogućili kreiranje grCUDA jezika koji je kao dodatni jezik GraalVM. Kako jezici u GraalVM efikasno komuniciraju međusobno tako je omogućena jednostavna komunikacija sa jezikom grafičke karte i samom grafičkom.

Politie

Holandska policija je imala monolitnu aplikaciju koja je koristila između ostalog TypeScript, Angular, Scala, Axon, SpringBoot, Slick i R. Ova aplikacija je obrađivala velike količine podataka u realnom vremenu, ali to je bilo veoma sporo. Cilj je bio preći u cloud i na mikroservisnu arhitekturu, a pri tom ne implementirati sve iz početka. GraalVM je omogućio ovaj prelazak i ubrzanje, pri čemu je ostao značajan deo starog koda.

3 Šta obuhvata projekat?

Projekat GraalVM je istraživački projekat koji razvija Oracle labs. Od 2012. preko šezdeset naučnih radova je izdato od strane razvojnog tima. Sam projekat obuhvata više komponenti koje ga čine korisnim i inovativnim.

3.1 GraalVM kao zamena za JVM

Jedna od prednosti GraalVM je što se može koristiti umesto Java virtualne mašine, on može da pokreće Javu, Skalu, Kotlin i sve ostale jezike koje se prevode u Java bajt kod. Od 2019. može se izvršavati na Linux-u, a od verzije 20.1.0 ima podršku i za Windows.

Još jedna od prednosti koja se pripisuje GraalVM-u su odlične performanse, neke demonstracije pokazuju da Ruby program izvršava i do 30 puta brže od originalne implementacije. Detaljnija testiranja su ipak pokazala da se u proseku Ruby kod izvršava oko 30% brže na GraalVM, što je i dalje obećavajući rezultat.

Kada su testirani Java programi rezultati su bili manje optimistični, performanse GraalVM-a su okvirno slične Oracle-ovom HotSpot kompajleru. Ovo samo po sebi ne predstavlja loš rezultat, ali ne predstavlja ni neki revolucionarni napredak u odnosu na sadašnje tehnologije. U svakom slučaju, GraalVM će biti brži od klasične JVM. Iako ne predstavlja poboljšanje u odnosu na HotSpot VM, prednost GraalVM-a je u tome što je nov kompajler nad kojim nije vršena optimizacija preko dvadeset godina, kao što je to slučaj sa HotSpot-om. Još jedna prednost u odnosu na prethodne kompajlere je to što je pisan u Javi za razliku od prethodnih kompajlera koji su pisani u jezicima C i C++, to omogućava lakše proširenje i optimizaciju od strane Java programera.

3.2 Truffle - kompajler za kompajlere

Truffle je biblioteka otvorenog koda za pravljenje alata i implementacija programskih jezika kao interpretera za samomodifikujuća apstraktna

sintakсна stabla. On nam dozvoljava da pravimo interpretere bez većih problema, ali prednost njegovog korišćenja je u tome što se interpretirani kod izvršava podjednako brzo kao i kompajlirani kod. On je korišćen za pisanje interpretera za jezik JavaScript u GraalVM-u. Pri izvršavanju koda potrebno mu je neko vreme da se zagreje, ali kada dostigne optimalne performanse one su okvirno iste kao V8 kompajler kompanije Google.

Sama optimizacija koda je zasnovana ideji parcijalne evaluacije. Truffle uzima interpreter koji smo mu dali i naš program i koristi JIT kompajler i promene sintaksnog stabla da izvrši optimizacije u našem kodu. Da bi se utvrdilo koje optimizacije je potrebno izvršiti program mora prvo biti pokrenut i zbog toga se javlja sporije vreme izvršavanja na početku.

Takođe, pri pisanju interpretera koristeći Truffle možemo da naglasimo gde i kada želimo da kod bude optimizovan. U nekim slučajevima promena u izvornom kodu može da dovede do greške u programu i u tom slučaju je potrebno deoptimizovati kod. Sve ove optimizacije i izbacivanja nekorišćenog koda iz našeg programa dovode do velike brzine izvršavanja.

Standardni podržani jezici za Truffle su JavaScript sa okruženjem Node.js ugrađenim u njega, kao i Ruby, R i Python. Pored toga, Truffle ima u sebe ugrađen Sulong koji može da izvršava LLVM bitkod i tako izvršava programe pisane na jezicima kao što su C, C++, C#, D, Lua i FORTRAN. Ovim omogućavamo da GraalVM izvršava veliki broj različitih programskih jezika kao i da se funkcije iz jednog jezika pozivaju u drugom jeziku omogućavajući veću fleksibilnost pri pisanju programa.

3.3 Izvršavanje mašinskog koda na kaludu

Jedna od glavnih funkcionalnosti koje uvodi ovaj projekat jeste kompajliranje Java programa sa bajt koda na mašinski kod. Ovo je novina koja nam dozvoljava tako nešto prvi put od kako postoji programski jezik Java. Jedan od slučajeva upotrebe za ovo je izvršavanje mikroservisa na kladu.

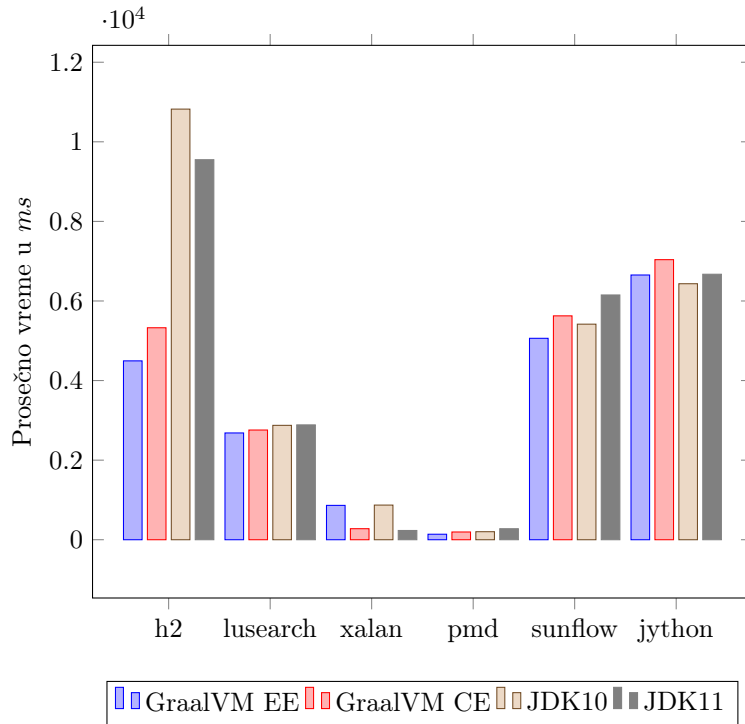
Mikroservisi se sada kompajliraju u izvršni fajl i time je eliminisana potreba da na serverskoj strani mora postojati Java viruelna mašina, što dovodi do smanjenja zauzeća memorije na serveru. Još jedna prednost ovog pristupa je u tome što više nije potrebno da se za naše programe učitava mnoštvo Java klasa pri pokretanju, od kojih većina neće biti upotrebljeno. Ovo dovodi do bržeg izvršavanja naših programa na serverskoj strani i sprečava sporo izvršavanje pri prvom pokretanju.

4 Karakteristike

Projekat *GraalVM* karakterišu visoke performanse, poliglot programiranje, *GraalVM AOT (ahead-of-time)* kompilacija u *Native Image*, napredni alati[5]. U narednim pod sekcijama će biti opisani svaki od ovih karakteristika.

4.1 Visoke performanse

Jedna od karakteristika kojom *GraalVM* može da se pohvali, jestu veoma visoke performanse u odnosu na *OracleJDK*, i *OpenJDK*.



Slika 2: *DaCapo* benčmark na *GraalVM EE*, *GraalVM CE*, *JDK10*, *JDK11*

Šipek i drugi [8] pokretanjem *DaCapo* benčmarka [1] pokazuju da *GraalVM CE*, i posebno *GraalVM EE* daju bolje performanse u odnosu na *JDK10* i *JDK11* (slika 2). Na testu h2 *GraalVM CE* daje rezultate i za 19.8% bolje od *JDK11*. Na drugim testovima vidimo malo poboljšanje *GraalVM* kompajlera u odnosu na *JDK*, ali i to da *JDK* u malom broju testova daje bolje rezultate. Pored toga, treba uočavamo da *GraalVM EE* u odnosu na *GraalVM CE* nad h2 testovima daje rezultate bolje i za 47.06%, što je veoma velika razlika. Što se stabilnosti tiče, tj. veličine standardne devijacije u performansam, najbolje se pokazuje *JDK11*, a najgore *GraalVM EE*.

Pored *DaCapo* benčmarka razvija se i novi *Renaissance* benčmark. *Renaissance* benčmark obećava prednosti nad konkurencijom, tako što koristi moderna, realna, konkurentna, i objektno orijentisana preopterećenja kako bi prikazao bolju sliku performansi kompajlera [7]. Pokretanjem ovih testova uočeno je značajno poboljšanje *GraalVM JIT* kompajlera u odnosu na *JDK11* i *JDK17* [11].

Poboljšanje u performansama je vidljivo i na drugim benčmarkovima. Tako je uočeno da *Scala* ima bolje performanse kada se koristi *GraalVM* optimizator, takođe na *DaCapo* benčmarkovima [10, 1]. Cloud servisi, takođe, dobijaju značajno unapređenje [9].

Treba napomenuti da postoje istraživanja koja pokazuju da *GraalVM* ne pokazuje značajna poboljšanja u odnosu na *JDK*. Fong i drugi [2] pokazuju da u nekim slučajevima *GraalVM* daje podjednake čak i lošije rezultate.

VS Code Extensions	Radno okruženje unutar VS Code-a
GraalVM Dashboard	Vizualna reprezentacija delova
Chrome Debugger	Debager za JS, Ruby, R i Python
VisualVM	Profajler, monitor, aktivni tredovi...
GraalVM Insight	Praćenje programa u izvršavanju
Ideal Graph Visualizer	Faze kompilacije

Tabela 1: Napredni alati unutar Projekta *GraalVM*

4.2 Native Image

GraalVM pruža tehnologiju *Native Image* [5]. *Native Image* predstavlja izvršivi binarni fajl koji je dobijen *GraalVM AOT* kompajlerom. Ovaj fajl sadrži klase aplikacije, klase biblioteka, zavisne klase i statički linkovan kod iz *JDK*-a. Dobijeni izvršivi fajl se ne pokreće na *Java VM*, već on uključuje potrebne dodatke iz *Java VM*-e. Za rezultat imamo to da se dobijeni program pokreće brže, jer učitava samo potrebne klase. Program je sigurniji, jer se sa manjom bazom koda mogućnost nastanka бага smanjuje. Program se lako dostavlja, jer se kapacitet kontejnera smanjuje.

4.3 Poliglot programiranje

Jedna od ključnih karakteristika koju *GraalVM* pruža jeste poliglot programiranje [5]. Poliglot programiranje podrazumeva da programeri za rešavanje neko problema koriste odgovarajući programski jezik. Tako rešene probleme onda koriste u projektu. Mehanizmi integracije više programskih jezika stvara dodatne kompleksnosti. Postoje mnogi radovi koji tvrde da se efikasnost programera, samim tim i programa smanjuje uvođenjem više jezičke strukture [6, 4]. Sa ciljem rešavanja ovog problema *GraalVM* pruža *TruffleVM* koji obećava lako i jednostavno prenošenje podataka, i korišćenje konstrukcija nekog programskog jezika unutar nekog drugog programskog jezika [3].

4.4 Napredni alati

U tabeli 1 su prikazani dostupni alati i njihove funkcionalnosti unutar projekta *GraalVM* [5].

5 Zaključak

Literatura

- [1] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann. The DaCapo Benchmarks: Java benchmarking development and analysis (extended version). Technical Report TR-CS-06-01, ANU, 2006. <http://www.dacapobench.org>.
- [2] Fredric Fong and Mustafa Raed. Performance comparison of graalvm, oracle jdk and openjdk for optimization of test suite execution time, 2021.

- [3] Matthias Grimmer, Chris Seaton, Roland Schatz, Thomas Würthinger, and Hanspeter Mössenböck. High-performance cross-language interoperability in a multi-language runtime. *SIGPLAN Not.*, 51(2):78–90, oct 2015.
- [4] Rebecca L Hao and Elena L Glassman. Approaching polyglot programming: what can we learn from bilingualism studies? In *10th Workshop on Evaluation and Usability of Programming Languages and Tools (PLATEAU 2019)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2020.
- [5] Oracle. GraalVM, 2018–2022. on-line at: <https://www.graalvm.org/>.
- [6] Cole S Peterson. Investigating the effect of polyglot programming on developers. In *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 1–2. IEEE, 2021.
- [7] Aleksandar Prokopec, Andrea Rosà, David Leopoldseder, Gilles Duboscq, Petr Tůma, Martin Studener, Lubomír Bulej, Yudi Zheng, Alex Villazón, Doug Simon, Thomas Würthinger, and Walter Binder. Renaissance: Benchmarking suite for parallel applications on the jvm. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2019*, page 31–47, New York, NY, USA, 2019. Association for Computing Machinery.
- [8] Matija Šipek, B Mihaljević, and Aleksander Radovan. Exploring aspects of polyglot high-performance virtual machine graalvm. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1671–1676. IEEE, 2019.
- [9] Matija Sipek, D Muharemagic, Branko Mihaljevic, and Aleksander Radovan. Enhancing performance of cloud-based software applications with graalvm and quarkus. *arXiv preprint arXiv:2201.11851*, 2021.
- [10] Lukas Stadler, Gilles Duboscq, Hanspeter Mössenböck, Thomas Würthinger, and Doug Simon. An experimental study of the influence of dynamic compiler optimizations on scala performance. In *Proceedings of the 4th Workshop on Scala*, pages 1–8, 2013.
- [11] Renaissance Suite. Renaissance Suite, A modern benchmark suite for the JVMA, 2019. on-line at: <https://renaissance.dev/>.