Projekat GraalVM

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

Bojan Bardžić, Milica Gnjatović, Pavle Savić, Andrija Urošević
mi18300@alas.matf.bg.ac.rs, mi18018@alas.matf.bg.ac.rs,
mi17169@alas.matf.bg.ac.rs, mi18083@alas.matf.bg.ac.rs

12. decembar 2022.

Sažetak

Danas postoji veliki broj programskih jezika. Neki od njih su veoma aktuelni, neki su zastareli i neefikasni, neki se brzo razvijaju dok se neki više ne održavaju. Ako bismo imali jedan savršen jezik i dalje bi bio problem prevesti sve programe koji su u upotrebi u taj jedan jezik. GraalVM je projekat koji nam pruža mogućnost da izvršavamo aplikacije napisane u više programskih jezika, uključujući i jezike koji se slabo održavaju. Pored toga pokazaćemo kako GraalVM to radi efikasno i zašto ga veliki broj kompanija danas koristi.

Sadržaj

1 Uvod **2**
1.1 Istorijski razvoj 3

2 Šta obuhvata projekat? **3**
2.1 GraalVM kao zamena za JVM 3
2.2 Truffle — kompajler za kompajlere 4
2.3 Native Image 4
2.4 Espresso — Java on Truffle 5

3 Karakteristike **7**
3.1 Visoke performanse 7
3.2 Poliglot programiranje 8

4 Kompanije koje koriste GraalVM **9**

5 Zaključak **9**

Literatura **10**

Slika 1: Podržani jezici.

1 Uvod

GraalVM predstavlja JDK (*Java Development Kit*) visokih performansi [11]. GraalVM omogućava ubrzanje izvršavanja uz korišćenje manje resursa. Nudi prevođenje Java aplikacija na dva načina: AOT (*Ahead Of Time*) i JIT (*Just In Time*). Pored Jave, neki od jezika koje GraalVM podržava su (slika 1):

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

U ovom radu prolazimo kroz osnovne elemente i ciljeve ovog projekta, šta je to inovativno uvedeno i gde se sve koristi.

Upotrebom GraalVM-a omogućeno je efikasnije korišćenje više jezika na jednom projektu. U projektu je moguće kodom pisanim u jednom jeziku pozivati funkcije pisane u drugom jeziku. Dopušteno je deljenje struktura podataka između kodova pisanih u različitim jezicima. Zbog toga sakupljač otpadaka radi na celom projektu, bez obzira na to koliko različitih jezika je korišćeno. Ovim je omogućeno i jednostavnije debagovanje.

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 [10]. Još jedna od mogućnosti koje GraalVM nudi je implementiranje novih jezika i alata korišćenjem biblioteke Truffle [10].

GraalVM je implementiran u Javi. Čine ga Java Virtuelna Mašina — JVM i Java Development Kit — JDK. Ovaj alat omogućava brže izvršavanje koda, a da se pri tome koristi manje memorije.

GraalVM je dostupan na Linux, Windows i MacOS operativnim sistemima [11]. Dostupna su dva izdanja GraalVM-a:

Community edition je otvorenog koda i dostupno je na github-u. Korisnici mogu doprineti razvoju ove verzije [4].

Enterprise edition razvija i licencira kompanija Oracle [9].

GraalVM je podržan od strane razvojnih okruženja i protokola za debagovanje. Neka od tih okruženja su Eclipse, NetBeans, IntelliJ IDEA i Visual Studio Code. Ova okruženja su posebno dobra jer podržavaju 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 debagovanje JavaScript, R i Ruby kodova.

1.1 Istorijski razvoj

Projekat GraalVM je istraživački projekat koji razvija Oracle labs. Od 2012. preko šezdeset naučnih radova je izdato od strane razvojnog tima. Jedan od prvih radova koji iznosi ideju ovog projekta je *One VM to rule them all* [22].

Java Virtuelne mašine poput Oracle Java HotSpotVM i IBM Java VM postoje više od dve decenije. Međutim nije postojala virtuelna mašina koja bi omogućila efikasno izvršavanje kodova pisanih u različitim jezicima. Cilj ovog projekta je bio da se napravi objedinjena virtuelna mašina koja bi ovo omogućila.

Prva verzija GraalVM 19.0 je objavljena u maju 2019. Trenutno najnovija verzija je GraalVM 22.3.0, objavljena u oktobru 2022. [12].

2 Šta obuhvata projekat?

Kao što je prethodno navedeno, ovaj projekat uvodi neke nove koncepte. Sam projekat obuhvata više komponenti koje ga čine korisnim i inovativnim. U ovom odeljku se govori o tome kako GraalVM zamenjuje JVM, *Native Image* tehnologiji, mogućnostima koje nudi *Truffle* i trenutno veoma aktuelnoj *Espresso* komponenti.

2.1 GraalVM kao zamena za JVM

Jedna od prednosti GraalVM je što se može koristiti umesto Java virtuelne mašine, on može da pokreće Java, Scala i Kotlin programe, kao i sve ostale programe pisane u jezicima 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 se Ruby program izvršava i do 30 puta brže od originalne implementacije [2]. Detaljnija testiranja su ipak pokazala da se u proseku Ruby kod izvšava oko 30% brže na GraalVM, što je i dalje obećavajući rezultat [8].

Kada su testirani Java programi, rezultati su bili manje optimistični, performanse GraalVM-a su okvirno slične Oracle-ovom HotSpot kompajleru [10]. 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.

2.2 Truffle — kompajler za kompajlere

Truffle je biblioteka otvorenog koda za pravljenje alata i implementacija programskih jezika kao interpretera za samomodifikujuća apstraktna sintaksna stabla (eng. *abstract syntax tree*) [10]. On nam dozvoljava da pravimo interpretere bez većih problema. Osnovna prednost njegovog korišćenja je u tome što se interpretirani kod izvršava podjednako brzo kao i program dobijen prevođenjem. Truffle je korišćen za pisanje interpretera za jezik JavaScrpit u GraalVM-u. Pri izvršavanju koda potrebno mu je neko vreme da se "zagreje" (eng. *warm-up time*), ali kada dostigne optimalne performanse one su okvirno iste kao kod kompajlera V8 kompanije Google.

Optimizacija koda je zasnovana na ideji parcijalne evaluacije (eng. *partial evaluation*) [8]. Truffle uzima interpreter koji smo mu dali i naš program, zatim koristi JIT kompajler i promenom sintaksnog stabla vrši optimizaciju koda. Da bi se utvrdilo koje optimizacije je potrebno izvršiti program mora prvo biti pokrenut, zbog toga se javlja sporije vreme izvršavanja na početku.

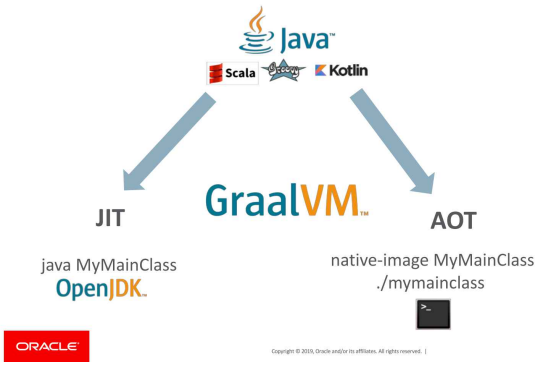
Takođe, pri pisanju interpretera koristeći Truffle možemo da naglasimo gde i kada želimo da kod bude optimizovan. U nekim slučajevima promena u izvornom kodu može da dovede do greške u programu i u tom slučaju je potrebno deoptimizovati kod. Sve ove optimizacije i izbacivanja nekorišćenog koda iz našeg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

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

2.3 Native Image

GraalVM pruža tehnologiju *Native Image* [10]. Ova tehnologija koristi AOT kompajler pomoću koga Java program prevodi u samostalnu izvršivu datoteku (eng. *standalone executable*) koja se, takođe, naziva *native image*. Ova izvršiva datoteka sadrži klase aplikacije, klase biblioteka, zavisne klase i statički linkovan kod iz *JDK*-a. Izvršiva datoteka se ne pokreće na *Java VM*, već uključuje neophodne delove *Java VM*-e. Za rezultat imamo smanjenje vremena pokretanja programa (eng. *startup time*) jer se učitavaju samo potrebne klase. Program je sigurniji, jer se sa manjom bazom koda mogućnost nastanka greške smanjuje. Program se lako isporučuje, jer se kapacitet kontejnera smanjuje.

Prevođenje Java programa sa bajt koda na mašinski kod je funkcionalnost koja nam dozvoljava tako nešto prvi put od kako postoji programski jezik Java. Jedan od slučajeva upotrebe ove tehnologije je izvšavanje mikroservisa (eng. *microservices*) na cloud-u [19]. Mikroservisi se sada kompajliraju u izvršivi fajl i time je eliminisana potreba da na serverskoj strani mora postojati Java virtuelna mašina. Ovo dovodi do smanjenja



Slika 3: Dva načina za prevođenje Java programa korišćenjem GraalVM-a.

zauzeća memorije (eng. *memory footprint*) na serveru. Kao što smo spomenuli, više nije potrebno da se za naše programe učitava mnoštvo Java klasa pri pokretanju, od kojih većina neće biti upotrebljena. Ovo dovodi do bržeg izvršavanja naših programa na serverskoj strani i sprečava sporo izvšavanje pri prvom pokretanju (eng. *cold start*).

Ovakav način prevođenja iako veoma primenjiv ipak ima i svoje nedostatke. Da bi izgradnja *native image*-a bila moguća neophodno je da sav kod bude poznat za vreme izgradnje slike (eng. *build time*) - uslov zatvorenog sveta (eng. *closed-world assumption*). Kao posledica ovog uslova ne mogu se koristiti određene dinamičke osobine Java-e. Ovaj problem u praksi je delimično rešen konfiguracionim fajlovima u kojima se navode očekivane klase u vremenu izvršavanja (eng. *runtime*). Druga tehnika jeste zamena dinamičkog izvornog Java koda statičkim korišćenjem anotacija [14].

2.4 Espresso — Java on Truffle

Sve donedavno GraalVM kompajler nudio je samo dva prethodno opisana standardna načina za prevođenje Java koda: *Ahead Of Time* i *Just In Time* (slika 3). Od verzije 21.0 uvedena je nova komponenta, nazvana *espresso*. *Espresso* predstavlja implementaciju JVM specifikacije, koja je napisana u Java-i pomoću Truffle radnog okvira. Ova komponenta nije podrazumevano deo GraalVM-a ali može se lako instalirati korišćenjem *GraalVM Updater tool*-a [13]. Na raspolaganju je i za GraalVM distribucije zasnovane na Java-i 8, kao i na Java-i 11 tako da se može koristiti kao zamena za JVM po izboru.

Java on Truffle režim izvršavanja pokreće Java kod preko bajtkod interpretera implementiranog korišćenjem Truffle-a (slika 4). Na ovaj način Java (i drugi JVM zasnovani jezici) pokreću se na isti način kao tradicionalni interpretirani i LLVM jezici koje GraalVM podržava. Ovo omogućava potpunu interoperabilnost sa njima — poliglot programiranje[1] [6, 13]. Iz tog razloga se mogu pozivati funkcije napisane u ovim jezicima u Java-i, kao i Java funkcije u drugim Truffle jezicima. Pri ovakvom izvršavanju podaci se nalaze u zajedničkom memorijskom prostoru.

[1]Poliglot programiranje je praksa pisanja koda u više jezika da bi se iskoristile dodatne mogućnosti svakog jezika. Više reči o tome će biti u delu 3.2.

Činjenica da je *Java on Truffle* implementirana u Java-i (eng. *self-hosting*) smatra se "Svetim gralom" u razvoju JVM-a. Da bi *Java on Truffle* mogla da pokrene Java kod neophodan joj je pristup JCL-u (Java Class Library), nativnim bibliotekama i metodama koje pruža JDK (Java Development Kit). *Java on Truffle* ponovno koristi Java Archive datoteke i nativne biblioteke iz GraalVM distribucije [6]. Kao posledica *self-hosting*-a *Java on Truffle* jeste metacirkularna VM[2]. Druga prednost ovoga je da je izvorni kod razumljiv Java programerima. Ovaj nivo transparentnosti, kroz zajednicu otvorenog koda, čini *Java on Truffle* projektom koji se ubrzano razvija i unapređuje.

Java on Truffle istovremeno je JVM i Java program, što znači da može biti pokrenuta unutar drugog Java programa. Ovo daje mogućnost razdvajanja aplikacije u komponente sa zajedničkom funkcionalnošću kako bi se proces programiranja učinio lakšim za upravljanje i podigla ponovna upotrebljivost koda. Na ovaj način može se ugraditi Java 8 kontekst u Java 11 aplikaciju i obrnuto, koristeći *GraalVM Polyglot API* [13]. Na primer ako su na raspolaganju obe distribucije (JDK 11 i JDK 8), *Java on Truffle* može biti pokrenuta kao Java 8 aplikacija, a potom može biti iskorišćena za pokretanje Java 11 bajtkoda i obrnuto [6]. Ako postoji biblioteka dostupna samo za Java-u 8, sada je moguće prebaciti se na noviji JDK i, uz manje programerske napore da se uspostavi interoperabilnost, koristiti je u Java 11 aplikaciji. Takođe ovime se povećava nivo izolovanosti *host* VM-a i Java programa koji se pokreće. Ovo povećava bezbednost pri izvršavanju manje pouzdanog ili nepoznatog koda [13].

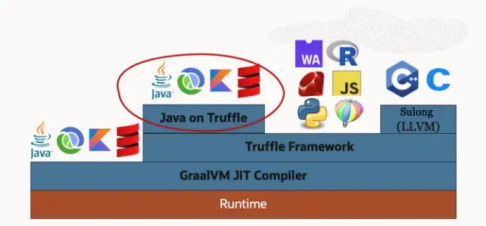
Java on Truffle ima sve osnovne komponente JVM, pa tako i JDWP (Java Debug Wire Protocol) koji služi za komunikaciju između debagera i ciljnog JVM-a. Nije potrebno ništa dodatno konfigurisati da bi se debagovale aplikacije pokrenute preko *Java on Truffle*. Moguće je korišćenje debagera integrisanog razvojnog okruženja [13]. Još jedna pogodnost koju nam *Java on Truffle* nudi jesu unapređene mogućnosti za redefinisanje klasa u fazi izvršavanja (eng. *hot swap*) u odnosu na HotSpot JVM tokom debagovanja. Neke od podržanih promena u verziji GraalVM 21.0:

- dodavanje i brisanje metoda i konstruktora klasa;
- dodavanje i brisanje metoda iz interfejsa;
- promena prava pristupa za metode i konstruktore;
- promena lambda izraza;
- dodavanje i brisanje anonimnih klasa.

Novije verzije donele su i mogućnost manipulacije poljima i hijerarhijama klasa. Nešto što treba očekivati u narednim verzijama jeste mogućnost promene enumeratorskih klasa [13].

Java on Truffle i dalje je eksperimentalna tehnologija. Maksimalne performanse su trenutno 2-3 puta sporije nego kod HotSpot-a. Vreme pokretanja je takođe nedovoljno optimizovano i još uvek nije na nivou brzine koju nudi standardno GraalVM JIT prevođenje [13]. Treba imati na umu da ovo nisu reprezentabilni podaci za ono što se očekuje od *Java on Truffle* u bliskoj budućnosti. Poboljšanje performansi jeste nešto na šta je razvojni tim trenutno najviše fokusiran. Pored toga radi se na podršci za Java agente[3], kao i na implementaciji boljih protokola interakcije sa drugim jezicima u okviru poliglotskih aplikacija [13].

[2]Metacirkularna VM može pokretati samu sebe nekoliko nivoa u dubinu, pri čemu svakim



Slika 4: *Java on Truffle* u GraalVM ekosistemu.

VS Code Extensions	Radno okruženje unutar VS Code-a
GraalVM Dashboard	Vizuelna reprezentacija delova projekta
Chrome Debugger	Debager za JS, Ruby, R i Python
VisualVM	Profajler, monitor, aktivne niti...
GraalVM Insight	Praćenje programa u izvršavanju
Ideal Graph Visualizer	Graf faza kompilacija

Tabela 1: Napredni alati unutar Projekta *GraalVM*.

3 Karakteristike

Projekat *GraalVM* karakterišu visoke performanse, *GraalVM AOT* kompajler u *native image* (opisan u sekciji 2.3), poliglot programiranje, i napredni alati [10] koji su prikazani u tabeli 1. U narednim podsekcijama će biti opisana svaka od ovih karakteristika.

3.1 Visoke performanse

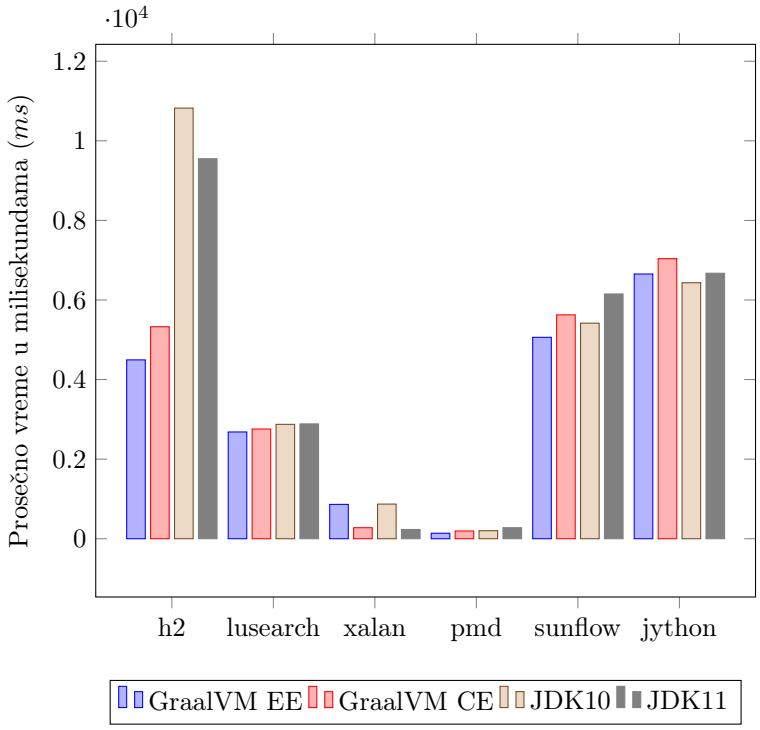
Jedna od karakteristika kojom *GraalVM* može da se pohvali, jesu veoma visoke performanse u odnosu na *OracleJDK*, i *OpenJDK*. Ovo tvđenje potkrepnjeno je brojnim benčmark (eng. *benchmark*) testiranjima.

Šipek i drugi [18] pokretanjem *DaCapo* benčmarka [1] pokazuju da *GraalVM Community Edition*, i posebno *GraalVM Enterprise Edition* daju bolje performanse u odnosu na *JDK10* i *JDK11* (slika 5). 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, 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 performansama, 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 [17]. Pokretanjem ovih testova uočeno je značajno poboljšanje *GraalVM JIT* kompajlera u odnosu na *JDK11* i *JDK17* [21].

silaskom postaje nešto sporija.

[3] Java agenti (eng. *Java agents*) su posebna vrsta klasa koje mogu zaustavljati i menjati bajtkod aplikacija koje se izvršavaju na JVM-u.



Slika 5: *DaCapo* benčmark na *GraalVM EE, GraalVM CE, JDK10, JDK11.*

Poboljšanje u performansama je vidljivo i na drugim benčmarkovima. Tako je uočeno da *Scala* ima bolje performanse kada se koristi *GraalVM* optimizator, takođe i na *DaCapo* benčmarkovima [20, 1]. *Cloud* servisi, takođe, dobijaju značajno unapređenje [19].

Treba napomenuti da postoje istraživanja koja pokazuju da *GraalVM* ne pokazuje značajna poboljšanja u odnosu na *JDK*. Fong i drugi [3] pokazuju da u nekim slučajevima *GraalVM* daje podjednake čak i lošije rezultate.

3.2 Poliglot programiranje

Jedna od ključnih karakteristika koju *GraalVM* pruža jeste poliglot programiranje [10]. Poliglot programiranje podrazumeva da programeri za rešavanje nekog 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 [16, 7]. 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 [5].

4 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 okvir 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 1.1 puta korišćenjem Community verzije i ubrzanje 1.42 puta korišćenjem Enterprise verzije GraalVM-a. [15]

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 mikroservisa Twitter-a je implementirano u jeziku Scala. Prelaskom na GraalVM je omogućeno 8-11% ušteda na procesoru. [15]

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 kompanija 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% [15].

Goldman Sachs

Goldman Sachs je investiciona banka. Ova kompanija ima svoj programski jezik Slang koji ima funkcije pogodne za njihove 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 Slang-u, a da je pri tom napisana minimalna količina novog koda [15].

Nvidia

Nvidia je jedan od najvećih proizvođača grafičkih kartica. Iako neki jezici jednostavno mogu da ubrzaju izvršavanje korišćenjem grafičke kartice, za neke druge to može biti izazovno. GraalVM i Truffle su omogućili kreiranje grCuda jezika koji je kao dodatni jezik GraalVM-a. Kako jezici u GraalVM-u efikasno komuniciraju međusobno tako je omogućena jednostavna komunikacija sa jezikom grafičke kartice i grafičkom karticom [15].

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 na cloud i mikroservisnu ahitekturu, a pri tom ne implementirati sve ispočetka. GraalVM je omogućio ovaj prelazak i ubrzanje, pri čemu je ostao značajan deo starog koda [15].

5 Zaključak

Projekat GraalVM donosi sa sobom mnogo novina, između ostalog imamo efikasan JDK pisan za Javu i ostale jezike zasnovane na JVM-u.

Pored toga ima i podršku za poliglot programiranje, kao i efikasno interpretiranje. Truffle nam nudi mogućnost pravljenja interpretera za potpuno nove jezike, kao i za stare koji su postali neefikasni i skupi za održavanje.

GraalVM se pokazao kao dobra alternativa JVM. Kroz benčmarkove smo utvrdili da GraalVM daje dobre performanse. Ovaj projekat je pokazao veliki potencijal u pogledu optimizacije i dopunjavanja kompajlera.

O kvalitetu ovog projekta nam govori činjenica da ovaj alat koriste mnoge velike kompanije, između ostalih, Oracle, Facebook, Twitter, i Nvidia. Posebno je dobro što imamo veliku kompaniju koja stoji iza projekta kao i zajednicu koja ga održava.

GraalVM je u ovom radu predstavljen kao nešto izuzetno korisno i efikasno. Dalje se postavlja pitanje 'Da li GraalVM ima konkurenciju?'. Trenutno nema projekata sličnih ovome. Prema tome, glavna konkurencija ostaju kompajleri i alati za pojedinačne jezike. Danas se kao glavna konkurencija navode Amazon Coretto, Red Hat, OpenJDK, Azul Platform Prime i Microsoft Build of OpenJDK.

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] Oracle Developers. GraalVM: Run Programs Faster Everywhere. online at: https://www.youtube.com/watch?v=iXCVaQzXi5w.

[3] Fredric Fong and Mustafa Raed. Performance comparison of graalvm, oracle jdk and openjdk for optimization of test suite execution time, 2021.

[4] github. GraalVM Community Edition github repository, 2020.

[5] 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.

[6] BJ Grooteman. *Java in Java with Truffle.* PhD thesis, Faculty of Science and Engineering, 2017.

[7] 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.

[8] Beyond Java. What About GraalVM? on-line at: https://www.beyondjava.net/what-about-graalvm.

[9] Oracle. GraalVM Enterprise Edition. on-line at: https://www.oracle.com/java/graalvm/.

[10] Oracle. GraalVM, 2018–2022. on-line at: https://www.graalvm.org/.

[11] Oracle. GraalVM Introduction, 2018–2022. on-line at: https://www.graalvm.org/latest/docs/introduction/.

[12] Oracle. GraalVM Releases, 2018–2022. on-line at: https://www.graalvm.org/release-notes/.

[13] Oracle. Java on Truffle, 2018–2022. on-line at: https://www.graalvm.org/latest/reference-manual/java-on-truffle/.

[14] Oracle. Native Image, 2018–2022. on-line at: https://www.graalvm.org/latest/reference-manual/native-image/.

[15] Oracle. GraalVM Use Cases, 2022. on-line at: https://www.graalvm.org/use-cases/.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] Renaissance Suite. Renaissance Suite, A modern benchmark suite for the JVMA, 2019. on-line at: https://renaissance.dev/.

[22] Thomas Würthinger, Christian Wimmer, Andreas Wöß, Lukas Stadler, Gilles Duboscq, Christian Humer, Gregor Richards, Doug Simon, and Mario Wolczko. One vm to rule them all. In *Proceedings of the 2013 ACM international symposium on New ideas, new paradigms, and reflections on programming & software*, pages 187–204, 2013.