UDC: 004.6004.42 Info M: str. 31-36

NEANDERTHAL: CLOJURE BIBLIOTEKA ZA BRZE OPERACIJE LINEARNE ALGEBRE
NEANDERTHAL: CLOJURE LIBRARY FOR FAST OPERATIONS OF LINEAR ALGEBRA

Dragan Đurić, Fakultet organizacionih nauka, Univerzitet u Beogradu

REZIME: Neanderthal je softverska biblioteka koja integriše brze platformski zavisne biblioteke za linearnu algebru sa Java platformom, namenjena korišćenju u Clojure programskom jeziku. Postojeće čiste Java biblioteke su spore i imaju proizvoljan apstraktni programski interfejs (API), dok biblioteke koje pozivaju platformski zavisne biblioteke imaju nepotrebne gubitke u komunikaciji, i takođe imaju proizvoljan API. Neanderthal ove probleme rešava tako što API modelira po uzoru na BLAS standard dok gubitke u komunikaciji minimizira kroz JNI-bazirani sloj za povezivanje pisan ručno u C programskom jeziku. Rezultujuća biblioteka ima jednostavniji API koji se slaže sa postojećom literaturom, dok je brzina dva i više puta veća u odnosu na postojeća najbrža rešenja.

KLJUČNE REČI: biblioteke za numeričku linearnu algebru, BLAS, Clojure, JNI

ABSTRACT: Neanderthal is a software library that integrates fast native linear algebra libraries with Java platform, targeted at Clojure programming language. The existing pure Java libraries are slow and have arbitrary abstract programming interfaces (APIs), while the libraries that call native libraries have unnecessary overhead in the communication layer, and also have arbitrary APIs. Neanderthal solves these problems by modeling the API on BLAS standard while it minimizes the communication overhead through a JNI-based hand-written layer written in the C programming language. The resulting library has a simpler API that matches the existing literature, while being at least twice faster than the fastest competing solutions.

KEY WORDS: Numerical linear algebra libraries, BLAS, Clojure, programming, JNI, Java

1. UVOD

Neanderthal je softverska biblioteka koja integriše brze platformski zavisne biblioteke za linearnu algebru sa Java platformom, namenjena korišćenju u Clojure programskom jeziku. Postojeće čiste Java biblioteke su spore i imaju proizvoljan apstraktni programski interfejs (API), dok biblioteke koje pozivaju platformski zavisne biblioteke imaju nepotrebne gubitke u komunikaciji, i takođe imaju proizvoljan API. Neanderthal ove probleme rešava tako što API modelira po uzoru na BLAS standard dok gubitke u komunikaciji minimizira kroz JNI-bazirani sloj za povezivanje pisan ručno u C programskom jeziku. Rezultujuća biblioteka ima jednostavniji API koji se slaže sa postojećom literaturom, dok je brzina dva i više puta veća u odnosu na postojeća najbrža rešenja. Nakon uvoda, osnove Clojure jezika i numeričke linearne algebre date su u sekcijama 2 i 3. Nakon toga, problem obezbeđivanja biblioteke za brzu linearnu algebru u Clojure jeziku na JVM platformi je opisan u sekciji 4, dok sekcija 5 opisuje dizajn rešenja. Sekcija 6 daje rezultate merenja i evaluaciju.

2. CLOJURE – PRAKTIČNI FUNKCIONALNI LISP NA JAVA PLATFORMI

Clojure [Hickey 2008] je jedan od novih jezika na Java platformi koji se kompajliraju u Java bajtkod. Kao dijalekt Lisp jezika, nasleđuje njegovu jednostavnost, veliku izražajnost i prilagodljivost, kao i jake teorijske temelje. Ovo je pragmatičan jezik, potekao iz industrije, koji izostavlja istorijske zamršenosti Lisp-a, a prigrljuje moderne mejnstrim platforme: prvenstveno Java virtuelnu mašinu, ali i sve prisutniji JavaScript, dok postoji i verzija za .NET. To mu omogućava neprimetnu integraciju sa ogromnim brojem dostupnih biblioteka na ovim platformama.

Clojure je funkcionalni jezik koji modeluje stvaran svet kao skup funkcija koje uzimaju određene vrednosti i vraćaju rezultujuće vrednosti. Čiste funkcije nemaju spoljnje efekte, ne menjaju nikakvu spoljnju memoriju, niti zavise od spoljnjih vrednosti, za razliku od danas preovlađujućeg imperativnog stila. U meri u kojoj je program pisan u funkcionalnom stilu, ne menja ništa što nije lokalno u svakoj funkciji, pa tako nema potrebe ni za kakvom sinhronizacijom.

Kao i u ostalim funkcionalnim jezicima, Clojure kod može da definiše funkcije koje kreiraju druge funkcije, funkcije koje primaju druge funkcije kao parametre i funkcije koje kombinuju druge funkcije sa skupom varijabilnih veza, nazvanim "closure". Clojure potpuno podržava i makroe. Makroi su u suštini funkcije koje generišu Lisp (Clojure) kod pre konačnog kompajliranja - programi koji pišu druge programe. Ove moćne funkcionalnosti koje su nezgrapno implementirane i teško primenjive u mejnstrim programskim jezicima elegantne su u Clojure-u i ostalim Lisp dijalektima zahvaljujući homoikoničnosti - Clojure kod je zapravo velika struktura podataka i nema razlike između koda i podataka sa kojima kod radi.

S obzirom da se Clojure kompajlira direktno u Java bajtkod i pruža pristup svim funkcionalnostima Java jezika i platforme, dobro napisani Clojure programi su brzi koliko i dobro napisani Java programi.

3. NUMERIČKA IMPLEMENTACIJA OSNOVNIH OPERACIJA LINEARNE ALGEBRE

U oblastima kao što je numeričko računanje, Java ne može iskoristiti sve potencijale mašine na kojoj se izvršava, prevashodno zato što joj nisu dostupne SIMD funkcionalnosti (npr. SSE). Ovo nije problem samo Jave, sve platforme koje na neki način apstrahuju mašinu na kojoj se izvršavaju, propuštaju

značajnu priliku za optimizaciju numeričkih operacija. Nekada su ti gubici zanemarivi u odnosu na ostale funckionalnosti aplikacije, ali veliko memorijsko zauzeće i složenost algoritama za rad sa matricama zahtevaju upotrebu primitivnih tipova podataka i sve moguće optimizacije dostupne na procesoru, čim dimenzije matrica narastu.

Operacije sa matricama, i linearna algebra generalno, su u osnovi mnogih grana nauke i tehnike, te se aplikacije za razne naučne primene na njih oslanjaju. Numeričke biblioteke se razvijaju još od samog začetka digitalnog računarstva, uz veliki broj naprednih optimizacija i prilagođavanja za veliki broj arhitektura. De fakto standard za operacije linearne algebre nad gustim podacima su Basic Linear Algebra Subprograms (BLAS) [Blackford et al, 2002] i LAPACK [Anderson et al, 1999] biblioteke, dok pokušaji da se standardizuju biblioteke za rad sa rasutim matricama nisu imali toliko uspeha.

BLAS programi su definisani na tri nivoa:

1. Nivo 1 definiše vektorske operacije koje se mogu implementirati jednom petljom O(n);
2. Nivo 2 definiše operacije sa matricama i vektorima, tipične složenosti O(n^2);
3. Nivo 3 definiše operacije sa matricama, tipične složenosti O(n^3), od kojih je najznačajnija proizvod dve matrice.

Na osnovama BLAS operacija izrađene su i mnoge biblioteke koje se bave sofisticiranijim problemima, poput rešavanja linearnih sistema, faktorizacija itd. Više o ovoj problematiči moze se pročitati u [Eijkhourt et al, 2015] i [Oliveira and Stewart 2006].

Za ovaj rad je značajno da postoje standardizovane biblioteke za numeričku linearnu algebru visokih performansi - potrebno je naći najbolji način da se one pozivaju iz Java platforme i Clojure programa na način koji je prirodan Clojure programerima.

4. ANALIZA PROBLEMA

Uprkos barem nekoj vrsti standardizacije biblioteka linearne algebre (BLAS, LAPACK), ne bi se moglo reći da na Java platformi postoji dominantno rešenje. Mana svih biblioteka je nekonzistentan i nepotpun API. Uz to, čiste Java biblioteke su uglavnom dosta sporije od platformski zavisnih rešenja, dok su biblioteke koje obmotavaju platformski zavisne biblioteke, poput jBLAS biblioteke [Braun 2015] veoma spore kada je rad sa manjim matricama u pitanju. Clojure rešenja u ovoj oblasti se zasnivaju uglavnom na pozivanju Java biblioteka, pa imaju iste karakteristike.

Najbitniji kriterijumi koje ćemo imati u vidu prilikom razmatrarnja rešenja su: brzina izvršavanja, uklapanje apstraktnog programskog interfejsa (API-ja) u Clojure, usklađenost sa standardima (prvenstveno BLAS) i kompatibilnost sa postojećom literaturom.

4.1 *Brzina*

Razmatrajmo množenje matrica. Ova operacija moze se obaviti sa tri ugnježdene petlje, i pri tome je njena kompleksnost O(n^3). Teorijski, postoje algoritmi sa manjom asimptotskom kompleksnošću, kao što je Strassen-ov (O(N^2.8074)) ili Coppersmith-Winograd (O(N^2.375477) ali oni imaju manju numeričku stabilnost i zahtevaju znatno složenije operacije sa memorijom, pa u praksi ne donose značajno ubrzanje [Skiena 2008].

Postojeće najbrže biblioteke rešenje nalaze u platformski zavisnim optimizacijama - optimalnom iskorišćenju keš memorije, registara procesora, kao i Single Instruction Multiple Data (SIMD) instrukcija. Njihov kod zbog toga postaje znatno složeniji od polazne tri petlje, te je vrlo teško, a zapravo i nije moguće u potpunosti, reimplementirati ga na Java platformi. Biblioteke programirane u čistoj Javi stoga znatno zaostaju u brzini za platformski zavisnim bibliotekama. Ova razlika zavisi od veličine i strukture matrica, ali generalno su barem za jedan red veličine sporije (kasnije ćemo to videti i pri merenjima).

Platformski zavisne biblioteke, rekli smo, koriste specifične hardverski zavisne optimizacije. Najbrže među njima su optimizovane za konkretne platforme, poput intelovog MKL-a [Wang et al, 2014]. Često su i komercijalne, i zatvorenog koda. Među rešenjima otvorenog koda najpopularnije su ATLAS [Whaley 2011] i OpenBLAS biblioteke, koje na osnovu merenja biraju optimalno rešenje za konkretnu platformu. Nešto su sporije od MKL-a ali vrlo malo, dok je njihova prednost dostupnost na većem broju platformi, sloboda i, naravno, cena. Za pozivanje ovih biblioteka iz Java platforme postoji više različitih tehnologija (JNI, JNA, automatsko generisanje omotača), svaka sa svojim prednostima i manama, i jedan od najbitnijih ciljeva ovog rada je izbor najbržeg od mogućih rešenja.

4.2 API

Pored same dostupnosti funkcija za rad sa matricama iz Clojure jezika, veoma bitan kriterijum je i da ponuđeno rešenje bude prirodno za korišćenje u Clojure okruženju. Sama priroda numeričkih operacija visokih performansi, koje zahtevaju korišćenje primitivnih struktura podataka i promenjlivost podataka, je dosta različita od uobičajenog Clojure stila koji se bazira na perzistentnim nepromenljivim strukturama podataka. Stoga je potrebno postići pravu meru kompromisa između C i/ili FORTRAN stila iz koga proizilazi BLAS i Clojure stila koji je na suprotnoj strani spektra. Java biblioteke, bez obzira da li su čiste ili obmotavaju platformski zavisne biblioteke, suviše pokušavaju da prate prakse tipične za objektno-orjentisano modelovanje generalnog softvera - stoga njihov API veoma varira. Zbog svoje neusklađenosti sa literaturom iz oblasti linearne algebre ne bi se moglo reći da je lak za učenje.

4.3 *Standard*

BLAS i LAPACK su standardi koji se razvijaju više decenija. Već dugo godina, njihov API se gotovo ne menja, što ukazuje na to da je postignut dobar balans između programskih zahteva i tehničkih mogućnosti implementacije, pa je softver veoma zreo. Pored toga, implementacije su dostupne za veoma

veliki broj hardverskih platformi. U računanju visokih performansi, ovo je veoma bitna činjenica, jer tehnički zahtevi često onemogućavaju programerski elegantija rešenja. Veliki broj biblioteka nudi naizgled prirodnije API-je, ali i sama raznovrsnost i nestabilnost njihovih rešenja, uz tipično višestruko niže performanse, pokazuje nužnost kompromisa.

4.4 *Literatura*

Uz tehnički prihvatljivo rešenje, potrebno je obezbediti i resurse za učenje. Stoga, rešenja koja se baziraju na postojećim, ili su barem modelovana na sličan način kao postojeća, imaju očiglednu prednost. Ovde je bitno napomenuti da velika većina biblioteka ima jako oskudnu dokumentaciju, uz gotovo kompletno odsustvo udžbeničke literature. BLAS i LAPACK imaju prednost u ovom smislu jer literatura vezana za naučni softver i softver visokih performansi uglavnom posvećuje prostor BLAS i LAPACK standardima ([Golub 2012] [Oliveira 2006]).

5. DIZAJN REŠENJA

Neanderthal biblioteka ne nastoji da zameni sve postojeće, niti smatram da je to moguće. Njeni ciljevi su fokusirani, prakticni, i spremni na kompromis. U skladu sa navedena četiri kriterijuma, Neanderthal nastoji:

1. da bude veoma brz, uporediv sa platformski zavisnim rešenjima;
2. da pruži API koji se uklapa u Clojure i funkcionalni stil programiranja;
3. da prati BLAS standard i iskoristi decenijsko iskustvo u razvoju numeričkih biblioteka za linearnu algebru;
4. da bude kompatibilan sa postojećom literaturom, koja je često orjentisana na BLAS

Uz to, biblioteka je besplatna i otvorenog je koda [Djuric 2015].

5.1 *Brzina*

Brzina platforski zavisnih biblioteka se smatra datom. U zavisnosti od toga da li ATLAS instaliramo kao prekompajlirani modul ili uz optimizovano kompajliranje iz sors koda, brzina će biti nešto manja ili nešto veća, ali svakako bliska dostižnom maksimumu na određenom hardveru, a za jedan ili više redova veličina veća nego odgovarajuće Java biblioteke.

Stoga, razlike u performansama biblioteka koje koriste ATLAS nastaju u procesu komunikacije između koda koji se izvršava u okviru JVM i spoljnjeg koda BLAS biblioteka. Osnovni parametri na koje ovde treba obratiti pažnju su:

1. Tehnologija komunikacije sa spoljnim bibliotekama: JNI, JNA, Gluegen, SWIG, BridgeJ itd.
2. Način transfera polaznih podataka iz JVM u spoljnju biblioteku i transfera rezultata natrag.

Osnovna tehnologija komunikacije Java virtuelne mašine sa spoljnim bibliotekama je Java Native Interface (JNI) [Gordon 1998]. Ovaj standard, trenutno u verziji 1.2, uključen je u Java SDK još devedesetih godina, i od tada je vrlo malo promenjen. JNI zahteva eksplicitno programiranje sloja posvećenog komunikaciji u programskom jeziku C. Iako nudi najbolje performanse, zahteva i najviše napora pri razvoju, i veoma je osetljiv na programerske greške, koje mogu dovesti i do pada virtuelne mašine. Java Native Access (JNA) [JNA 2015] se nalazi na drugom kraju spektra. JNA generiše pozive spoljnim bibliotekama u toku izvršavanja, platformski je nezavisna biblioteka, i veoma laka za koriscenje. Međutim, za jedan ili dva reda veličine je sporija od JNI. Tipično, pozivanje najjednostavnije platformski zavisne metode napisane u JNI je neznatno sporije od pozivanja čiste Java metode (nekoliko nanosekundi), dok pozivanje odgovarajuće JNA metode traje nekoliko stotina nanosekundi. Između ove dve krajnosti nalaze se ostale biblioteke, koje uglavnom pokušavaju da automatskim generisanjem JNI koda postignu bolje performanse, pritom pružajuci lakši način programiranja.

Pri razmatranju brzine izvršavanja, pored brzine samog poziva metode (npr. preko JNI), i brzine izvršavanja operacija u samoj spoljnoj biblioteci, na koju nemamo uticaja, potrebno je razmotriti i problem transfera podataka ka/iz spoljne biblioteke. Najjednostavniji način bi bio direktno referenciranje u C kodu ka adresi podataka (na primer niza float-ova) u samoj virtuelnoj mašini - ali to bi bilo katastrofalno rešenje. U ogranicenim uslovima testiranja verovatno bi i funkcionisalo, ali ako garbage collector (GC) izvrši premeštanje podataka u toku računanja, spoljna biblioteka bi vratila potpuno pogrešan rezultat, jer bi računala sa besmislenim podacima, ili bi srušila celu virtuelnu mašinu, zavisno od konkretne situacije. Ne smemo zaboraviti da Java reference nisu sirovi pokazivači, i da su podložne upravljanju od strane virtuelne mašine. Zbog ovoga, većina rešenja mora ili blokirati GC dok radi sa podacima, ili kopirati podatke pri svakom pozivu platformski zavisnih metoda, što znatno utiče na performanse i isplati se samo u slučaju metoda koje se i inače jako dugo izvršavaju. Postoji i rešenje upotrebe direktnih bafera kojima ne upravlja JVM, i ovo rešenje moguće je koristiti iz JNI, uz posebnu pažnju. Ostala rešenja ne pružaju ovakvu mogućnost optimizacije jer teže generičkim rešenjima, a direktan pristup zahteva direktnu odluku od strane programera u svakom posebnom slučaju.

Rešenje koje je primenjeno u Neanderthal biblioteci specifično je po tome što koristi ručno napisan JNI sloj za povezivanje u C programskom jeziku i što koristi direktne pokazivače, koji ne zahtevaju kopiranje podataka a pritom ne predstavljaju opasnost po stabilnost virtuelne mašine (ukoliko se pravilno koriste!). Sloj za povezivanje je kreiran u skladu sa JNI standardom, i prati specifikaciju platformski zavisnih metoda u Java delu biblioteke, preko kojih se poziva, prima podatke, i prosleđuje i spoljnoj BLAS biblioteci. Mora se kompajlirati za svaku konkretnu platformu (Linux, OSX, Windows itd.), a rezultujuća binarna datoteka se mora nalaziti u Java putanji biblioteka da bi se mogla pozvati. Ovo predstavlja teškoću, jer zahteva posebna podešavanja na svakom kompjuteru prilikom instalacije (jer nije čista Java biblioteka), ali je zahvaljujući

Maven [O'Brien et al, 2010] build sistemu i NAR pluginu za platformski zavisne biblioteke [NAR 2015] ovo prevaziđeno, pa je kreiranje i referenciranje biblioteke automatsko pomoću Maven ili Leiningen [Leiningen 2015] build sistema. Najviše uštede u vremenu komunikacije dobija se korišćenjem direktnih bafera. Zahvaljujući i činjenici da BLAS API koristi pakovane jednodimenzionalne nizove za predstavljanje i vektora i matrica, direktno je moguće preslikati direktne bafere iz Java dela programa u pokazivače na nizove u BLAS delu, pa je izbegnuto kopiranje kome moraju pribegavati druge biblioteke, koje koriste automatsko generisanje sloja za komunikaciju.

5.2 *API kao kompromis Clojure stila i BLAS standarda*

Principi po kojima je definisan Neanderthal API prate principe po kojima je definisan BLAS standard, i prilagođavaju ih Clojure stilu programiranja. Uzmimo za primer proceduru DGEMM, koja množi dve matrice. Prirodna reakcija novajlije je da ovakva procedura krši osnovna pravila imenovanja u programiranju. Zar nije prirodnije da se za ovu operaciju koristi neki operator, na primer *, ili, ako to nije tehnički izvodljivo, da se za ime procedure iskoristi neka varijacija glagola "multiply"?

Za ovo ipak postoji valjan razlog. Operacije sa matricama se najčešće koriste prilikom računanja u kome se zahtevaju visoke performanse. Bez obzira da li je u pitanju neki od jezika koji se interpretira, poput Python-a, jezika koji se izvršavaju na virtuelnoj mašini, kao što su Java ili C#, ili jezika bližih samoj mašini, poput C-a, kompleksnost operacija sa matricama čini da naivni algoritmi ne pružaju zadovoljavajuću brzinu. Vodeće biblioteke za rad sa matricama se oslanjaju na optimizacije koje zahtevaju programiranje prilagođeno svakoj procesorskoj arhitekturi posebno, pri tome vodeći računa o memorijskom zauzeću. Stoga se za operacije moraju koristiti primitivni tipovi podataka (float, double itd.), a strukture za smeštanje matrica i algoritmi za obradu se programiraju posebno za različite tipove matrica (generalne, simetrične, dijagonalne itd.). Pri tome, te strukture se zasnivaju na pakovanju podataka u jednodimenzionalne nizove, zbog boljeg iskorišćenja memorijskog prostora i paterna pristupa keš memoriji i registrima procesora. Mehanizmi za polimorfizam u programskim jezicima višeg nivoa ne podržavaju u potpunosti rad sa primitivnim podacima, te se sve ove operacije moraju eksplicitno identifikovati.

Konvencija imenovanja u BLAS standardu diferencira tri bitne informacije:

1. tip podataka (double, float, complex);
2. vrstu strukture u kojoj se nalaze podaci (vektor, razne vrste matrica);
3. vrstu operacije linearne algebre (proizvod vektora, proizvod matrica itd.)

Na primeru DGEMM operacije, ilustrovana je konvencija za imenovanje koja se koristi u BLAS standardu [BLAS 2015]:

D - Double, procedura radi sa primitivnim brojevima u pokretnom zarezu sa duplom preciznošću;

GE - GEneral, procedura je namenjena generalnim matricama kod kojih se svi elementi čuvaju u memoriji;

MM - Matrix Multiply, procedura obavlja operaciju množenja matrica.

Tipičan poziv BLAS metode je dat u primeru implementacije platformski zavisne metode CBLAS.dgemm:

```
   JNIEXPORT void JNICALL Java_uncomplicate_neanderthal_CBLAS_dgemm
   (JNIEnv *env, jclass clazz,
    jint Order, jint TransA, jint TransB,
    jint M, jint N, jint K,
    jdouble alpha,
    jobject A, jint lda,
    jobject B, jint ldb,
    jdouble beta,
    jobject C, jint ldc) {

     double *cA = (double *) (*env)->GetDirectBufferAddress(env, A);
     double *cB = (double *) (*env)->GetDirectBufferAddress(env, B);
     double *cC = (double *) (*env)->GetDirectBufferAddress(env, C);
     cblas_dgemm(Order, TransA, TransB, M, N, K, alpha, cA, lda, cB, ldb, beta, cC, ldc);
     };
```

cblas_dgemm metoda prima tri sirova niza primitivnih podataka (A, B i C) i više brojeva koji opisuju strukturu kojom se predstavljaju matrice (dimenzije, offset itd).

Dok se pri pozivu BLAS metoda moraju koristiti ovakve sirove metode niskog nivoa, Clojure deo biblioteke ima na raspolaganju polimorfizam i overloadovanje metoda, te je API jednostavniji. Množenje matrica se vrši metodom mm (matrix multiply), koja prihvata različite tipove matrica i prosleđuje ih odgovarajucim primitivnim metodama. D i GE deo semantike sadrže se u samom tipu matrice, koji ujedno enkapsulira i primitivni niz, i pruža veliki broj semantički bogatijih funkcija za rad sa matricama. Ovo možemo ilustrovati testovima koji prikazuju kreiranje i rad sa matricama (dge kreira generalne matrice sa duplom prezicnošću a mm i mm! množe matrice):

```
   (facts
    "BLAS 3 mm: Here is how you can multiply matrices. Note that
   this is matrix multiplication, NOT an element-by-element multiplication,
   which is a much simpler and less useful operation."
    (let [a (dge 2 3 (range 6))
          b (dge 3 1 (range 3))
          c (dge 2 1 [1 2])]

      (mm a b) => (dge 2 1 [10 13])
      (mm 1.5 a b) => (dge 2 1 [15 19.5])
      a => (dge 2 3 (range 6))
      b => (dge 3 1 (range 3))
      (mm! c a b) => (dge 2 1 [11 15])
      c => (dge 2 1 [11 15])
      (mm! c 1.5 a b 2.0) => (dge 2 1 [37 49.5])
      c => (dge 2 1 [37 49.5])))
```

Drugim rečima, ako već imamo matrice a i b, možemo ih pomnožiti veoma jednostavnim izrazom:

```
(mm a b)
```

Ovo je dosta jednostavnije i jasnije nego

```
cblas_dgemm(Order, TransA, TransB, M, N, K, alpha, cA, lda, cB, ldb, beta, cC, ldc);
```

Na web stranici http://neanderthal.uncomplicate.org dostupna su uputstva, a konkretni primeri mogu se videti u testovima dostupnim na https://github.com/uncomplicate/neanderthal/blob/master/test/uncomplicate/neanderthal/examples/guides/tutorial_test.clj

6 EVALUACIJA PERFORMANSI

Koliko Neanderthal ispunjava očekivanja?

6.1. Rezultati ukratko

Čak i za veoma male matrice (izuzev matrica manjih od 5x5), Neandertal je brži od Vectorz biblioteke. Pri računanju matrica svih veličina, Neandertal je brži od jBLAS biblioteke. Za velike matrice, Neandertal je konstantno dva puta brži od jBLAS-a. S obzirom da Neandertal kao podrazumevano podešavanje koristi jednoprocesorski ATLAS, ukoliko je potrebno može se jos ubrzati iskorištavanjem višeprocesorske verzije ATLAS-a.

6.2. *Šta je mereno*

Neanderthal koristi ATLAS uz gotovo nikakve gubitke - za sve sem za veoma male matrice, brz je koliko je brza kompajlirana verzija ATLAS-a koja se nalazi u instalaciji sistema.

Ovde ćemo se koncentrisati na merenje performansi množenja matrica, pošto je to najsloženija operacija (kompleksnost O(n^3) koja najbolje otkriva razlike u performansama testiranih pristupa. U kreiranju Neanderthal-a obraćena je posebna pažnja da rešenje bude što laganije u smislu korišćenih rešursa, koje potpuno izbegava kopiranje podataka u komunikaciji Java virtuelne mašine sa sistemskim bibliotekama, pa su čak i linearne operacije, vektor-vektor (BLAS 1) i matrica-vektor (BLAS 2) brze nego u čistoj Javi.

Neanderthal je upoređen sa dve tipične biblioteke za rad sa matricama:

1. Vectorz [Vectorz 2015] - Čista Java/Clojure biblioteka za rad sa matricama. Brza u radu sa malim matricama, ali sporija kod velikih matrica.
2. jBLAS - Biblioteka koja koristi statički kompajliranu platformski zavisnu ATLAS BLAS implementaciju. To je najpopularnija biblioteka za rad sa matricama na Java platformi. Laka je za instalaciju. Važi za brzu u BLAS 3 operacijama u radu sa velikim matricama, dok je sporija u radu sa malim matricama.

Ove dve biblioteke su dobri predstavnici stanja kada je reč o bibliotekama za linearnu algebru na Java platformi. Jedna je predstavnik čistih Java biblioteka, a druga predstavnik biblioteka koje koriste platforski zavisne biblioteke.

6.3. *Programski kod za merenje performansi*

Merenja su izvrsena korišćenjem Criterium biblioteke [Criterium 2015] za merenje performansi. Aplikacija za merenje dostupna je u okviru samog Neanerthal projekta na adresi https://github.com/uncomplicate/neanderthal/tree/master/examples/benchmarks

6.4. *Rezultati merenja*

Na jednom jezgru Intel Core i7 4790k procesora, uz 32 GB RAM memorije, Arch Linux sistemu, Neanderthal 0.1.1. bibliotekom koja poziva ATLAS 3.10.2, jBLAS 1.2.3 i Vectorz 0.44.0 bibliotekama izmereni su prosečni rezultati sa vrlo malom standardnom devijacijom (manje od 2.5%) dati u Tabeli 1.

Matrix Dimensions	Neanderthal	jBLAS	Vectorz	Neanderthal vs jBLAS	Neanderthal vs Vectorz
2x2	112.48270 ns	6.87139 µs	57.37400 ns	61.09	0.51
4x4	150.87644 ns	7.03403 µs	132.33028 ns	46.62	0.88
8x8	385.66491 ns	7.23608 µs	613.88615 ns	18.76	1.59
16x16	1.19812 µs	8.31470 µs	3.79212 µs	6.94	3.17
32x32	8.03734 µs	15.56198 µs	25.82390 µs	1.94	3.21
64x64	24.60521 µs	50.65159 µs	210.81286 µs	2.06	8.57
128x128	164.25813 µs	305.65059 µs	1.55666 ms	1.86	9.48
256x256	1.41115 ms	2.54841 ms	12.04184 ms	1.81	8.53
512x512	9.23014 ms	18.93424 ms	94.65790 ms	2.05	10.26
1024x1024	70.92693 ms	136.37988 ms	751.53860 ms	1.92	10.60
2048x2048	555.94468 ms	1.08081 sec	6.01701 sec	1.94	10.82
4096x4096	4.44797 sec	8.58928 sec	45.92363 sec	1.93	10.32
8192x8192	32.51815 sec	1.10277 min	6.14534 min	2.03	11.34

Tabela 1. - *Poređenje brzine operacije množenja matrica pomocu Neanderthal, Vectorz i jBLAS biblioteka.*

Naravno, merenje performansi jako je zavisno od konkretnog slučaja korišćenja. Ovi rezultati samo pokazuju da Neanderthal jako dobro iskorišćava brzinu ATLAS biblioteke, i od njenih porformansi potpuno zavisi. To može biti i prednost, jer se optimizacijom ATLAS instalacije dobijaju sve njene prednosti (nažalost, i mane). Detaljnija analiza performansi može se videti u [Whaley 2001], [Whaifj 2005] i [Goto 2008].

6.5. *Mane*

Pored očiglednih prednosti, ovakva biblioteka ima i nekoliko karakteristika koje mogu ali i ne moraju da budu bitne pri izboru rešenja u njenom korišcenju. Kao i sve biblioteke koje pozivaju spoljne, ne-Java biblioteke, ova biblioteka zahteva posebnu instalaciju ATLAS biblioteke na sistemu. Takođe, ona je još u razvoju i može trpeti stalne promene, za razliku od biblioteka koje se duže razvijaju i postigle su veću stabilnost. Takođe, za očekivati je da ima više bagova od biblioteka koje su duže prisutne i više se koriste.

7. ZAKLJUČAK

U radu je dat pregled problema brzih numeričkih softverskih biblioteka na Java platformi, konkretno namenjenih programskom jeziku Clojure, i predloženo rešenje u vidu biblioteke Neanderthal. Opisani su detalji problema i prikazano kako su rešeni kombinacijom Clojure, Java i C programskih jezika. Naravno, na kraju su prikazani rezultati evaluacije koji konkretno ukazuju na značajna poboljšanja performansi u odnosu na postojeća rešenja.

8. REFERENCE

[1] Hickey, Rich. "The clojure programming language." In Proceedings of the 2008 symposium on Dynamic languages, p. 1. ACM, 2008.

[2] Blackford, L. Susan, Antoine Petitet, Roldan Pozo, Karin Remington, R. Clint Whaley, James Demmel, Jack Dongarra et al. "An updated set of basic linear algebra subprograms (BLAS)." ACM Transactions on Mathematical Software 28, no. 2 (2002): 135-151.

[3] Anderson, Edward, Zhaojun Bai, Christian Bischof, Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz et al. LAPACK Users' guide. Vol. 9. Siam, 1999.

[4] Eijkhout, Victor. Introduction to High Performance Scientific Computing 2nd Ed., 2015.

[5] Oliveira, Suely, and David E. Stewart. Writing Scientific Software: A Guide to Good Style. Cambridge University Press, 2006.

[6] Braun, L. "Mikio. Jblas- fast linear algebra for java." https://www.scribd.com/doc/104872010/jblas-Fast-matrix-computations-for-Java (10.5.2015).

[7] Skiena Steven, S. "The algorithm design manual. 2nd ed" (2008).

[8] Wang, Endong, Qing Zhang, Bo Shen, Guangyong Zhang, Xiaowei Lu, Qing Wu, and Yajuan Wang. "Intel Math Kernel Library." In High-Performance Computing on the Intel® Xeon Phi™, pp. 167-188. Springer International Publishing, 2014.

[9] Golub, Gene H., and Charles F. Van Loan. Matrix computations. Vol. 3. JHU Press, 2012.

[10] Djuric, Dragan, "Neanderthal – Fast Matrix and Linear Algebra in Clojure", http://neanderthal.uncomplicate.org (10.5.2015.)

[11] Gordon, Rob, and Rob Gordon. Essential JNI: Java Native Interface. Vol. 11. Englewood Cliffs: Prentice Hall, 1998.

[12] O'Brien, T., J. van Zyl, B. Fox, J. Casey, J. Xu, and T. Locher. "Maven: By example. An introduction to Apache Maven." (2010).

[13] JNA, JNA Github project page, https://github.com/twall/jna, (10.5.2015.)

[14] NAR, NAR Github project page, https://github.com/maven-nar/nar-maven-plugin (10.5.2015.)

[15] Leiningen, "Leiningen for automating Clojure projects without setting your hair on fire", http://leiningen.org, (10.5.2015.)

[16] Criterium, Criterium Github project page, https://github.com/hugoduncan/criterium (10.5.2015.)

[17] Vectorz, Vectorz Github project page, https://github.com/mikera/vectorz (10.5.2015.)

[18] Whaley, R. Clint. "ATLAS (Automatically Tuned Linear Algebra Software)." In Encyclopedia of Parallel Computing, pp. 95-101. Springer US, 2011.

[19] BLAS, Naming Scheme, http://www.netlib.org/lapack/lug/node24.html (10.5.2015.)

[20] Goto, Kazushige, and Robert Van De Geijn. "High-performance implementation of the level-3 BLAS." ACM Transactions on Mathematical Software (TOMS) 35.1 (2008): 4.

[21] Whaley, R. Clint, Antoine Petitet, and Jack J. Dongarra. "Automated empirical optimizations of software and the ATLAS project." Parallel Computing 27.1 (2001): 3-35.

[22] Whaley, R. Clint, and Antoine Petitet. "Minimizing development and maintenance costs in supporting persistently optimized BLAS." Software: Practice and Experience 35.2 (2005): 101-121.

Dragan Đurić, Fakultet organizacionih nauka
Kontakt: dragandj@gmail.com
Oblasti interesovanja: programiranje, mašinsko učenje, inteligentni sistemi