UDC: 026:004.738.52 INFO M: str. 11-18

STATIČKI TIPIZIRANA MATRICA NA JEZIKU C++
STATICALLY TYPED MATRIX IN C++ LANGUAGE

Predrag S. Rakić, Lazar Stričević, Zorica Suvajdžin Rakić
Fakultet tehničkih nauka Novi Sad

REZIME: U savremenim C++ bibliotekama matrice su implementirane kao da su samo tip elementa i broj dimenzija zaista bitne karakteristike, koje čine statički tip matrice. Veličina dimenzija je u ovakvoj implementaciji tretirana kao promenljiva karakteristika, koja se određuje dinamički u vreme izvršavanja. Na ovaj način implementirana matrica predstavlja delimično statički, a delimično dinamički tip. Na jeziku C++ to je dozvoljena i među korisnicima opšte prihvaćena implementacija. I pored toga, ovakav hibridni pristup smatramo neusklađenim sa duhom statički tipiziranog jezika, kakav je C++. Takođe smatramo da je posledica ove neusklađenosti manje ili više primetna neusaglašenost interfejsa i semantike matričnih operacija.

U ovom radu predstavljamo implementaciju u kojoj se tip elemenata, broj dimenzija i veličina dimenzija tretiraju ravnopravno i zajedno čine statički tip matrice. Upotrebljivost predloženog pristupa je demonstrirana implementacijom potpuno statički tipiziranih matrica u C++ biblioteci pod imenom Typed Matrix Library (TML). Predstavljanje matrica na ovaj način omogućava verifikaciju korektnosti matričnih operacija u vreme prevođenja i ne unosi usporenje (overhead) u izvršavanje programa. Interfejs ovako implementiranog tipa matrice je sličan interfejsu opšte prihvaćene, hibridne implementacije. Smatramo da upotreba ovde predstavljenog tipa matrice ne zahteva dodatne informacije (bar ne neke kojih programer već nije svestan), a time ni dodatni napor od programera.

KLJUČNE REČI: tip matrice, statičko tipiziranje, C++, templejt meta-programiranje

ABSTRACT: Matrices in contemporary C++ libraries are implemented as if element type and number of dimensions are the only relevant characteristics, included in statically typed part of matrix type. Size of each dimension is treated as variable property and determined dynamically, at runtime. Implemented like this, matrix is typed partially statically and partially dynamically. Types like this are legal in C++ language and generally accepted in C++ community. However, we perceive this hybrid implementation as inconsistent with the spirit of the language. Consequence of this inconsistency is more or less noticeable discrepancy in the matrix operation's interface and semantic.

Matrix implementation in which element type, number of dimensions and size of each dimension are all treated equally as elements of matrix static type is presented in this paper. Proposed implementation is demonstrated in the proof-of-concept C++ template library called Typed Matrix Library (TML). Representing matrices this way enables compile-time correctness verification in matrix operations, incurring no run-time overhead. Our implementation interface is similar to generally accepted, hybrid implementation, requiring no additional information (that programmer is not already aware of) and thus no extra developer effort.

KEY WORDS: matrix type, static typing, C++, template meta-programming

1. UVOD

U istraživačkom radu i inženjerstvu se često pojavljuju složeni matematički modeli i računski zahtevni algoritmi i programi. Oblast računarstva koja se bavi ovom klasom programa naziva se "naučno računarstvo" (*scientific computing*) [14]. Programi iz ove oblasti su zahtevni po dve dimenzije. Sa jedne strane visoko su kompleksni, pa je proces modelovanja i razvoja složen i često vodi ka mukotrpnom i dugotrajnom testiranju i debagiranju. Sa druge strane ova vrsta programa zahteva značajne računarske resurse (procesorsku snagu i memorijski prostor) za izvršavanje.

Od te dve dimenzije ova druga, vezana za algoritme, računarske mehanizme, brzinu izvršavanja i zauzeće memorije se tradicionalno posmatra kao domen dominantne kompleksnosti u ovoj oblasti [12]. Fortranske biblioteke za rad sa matricama se dominantno bave algebarskim transformacijama i njihovom optimizacijom. U nekim C++ bibliotekama pored efikasnosti izvršavanja, autori pažnju obraćaju na interfejs i kao jedan od ciljeva implementacije navode lakoću upotrebe [19].

U numeričkim C++ bibliotekama kao što su Blitz++ [2], Armadillo [19], uBLAS [8] ili MTL [6], matrice različitih dimenzija nisu matrice različitih tipova. Dimenzije ovih matrica postaju poznate tek u vreme izvršavanja, što znači da se tek tada može proveriti da li operandi matričnih operacija zadovoljavaju invarijante (npr. da li su matrice koje se sabiraju istih dimenzija). Biblioteka Blitz++ pruža donekle mogućnost statičke tipizacije. Ona dozvoljava postojanje više-dimenzionih matrica i posmatra matrice različitog broja dimenzija kao matrice različitih tipova. Iako može biti korisna, ova klasifikacija je suviše gruba da bi omogućila pouzdanu proveru slaganja dimenzija.

Ovakvo hibridno, delimično statičko, a delimično dinamičko tipiziranje matrica, je verovatno iz istorijskih razloga, opšte prihvaćen način programiranja numeričkih algoritama čak i na C++ jeziku. U ovom radu kada govorimo o hibridnim tipovima, mislimo na način (to jest, vreme) određivanja i provere ispravnosti tipa - statička ili dinamička tipizacija (*static vs. dynamic typing*) [9]. U slučaju hibridnog tipa radi se o kombinaciji ova dva načina.

Implementaciju tipa matrice u savremenim C++ bibliotekama smatramo hibridnom zato što je deo bitnih karakteristika tipa implementiran statički, a deo dinamički. Osobine koje nisu poznate u vreme prevođenja su zadate dinamički. I prove-

ra ispravnosti tipa je hibridna: statički se proverava ono što se može proveriti u vreme prevođenja, a dinamički sve ostalo što je potrebno proveriti (a statički nije moguće, na primer zato što u vreme prevođenja nije poznato).

Jedan od osnovnih razloga za upotrebu hibridnih tipova je ograničena izražajnost statičke specifikacije, tj. nemogućnost izražavanja složenih apstrakcija [13]. Na jeziku C++ to nije problem, bar ne u slučaju matrica. Mehanizam tipova ugrađen u C++ je dovoljno razvijen da može da opiše sve što je potrebno za predstavljanje modela matrice.

Hibridni tipovi predstavljaju interesantno rešenje i kada deo karakteristika tipa nije poznat u vreme prevođenja, što najčešće jeste problem tokom programiranja program koji rade sa matricama. U vreme pisanja programa, uglavnom nije poznata veličina sistema koji se rešava. Veličina sistema, tj. dimenzije matrica zavise od skupa ulaznih podataka. Ulazni podaci su promenljivi i najčešće poznati tek u vreme izvršavanja. Potreba da se tip matrice implementira tako da se dimenzije matrice mogu zadati u vreme izvršavanja je verovatno pravi/osnovni razlog za hibridnu implementaciju tipa matrice.

Prednosti kombinacije statičke i dinamičke provere ispravnosti tipova su davno prepoznate [10] i teorijski razrađene. Razvijen je funkcionalni programski jezik SAGE u kojem su hibridni mehanizmi ugrađeni u sam jezik [15]. Ovaj pristup je očigledno teorijski analiziran, praktično proveren, primenjiv i opšte prihvaćen. I pored toga, smatramo da je na jeziku C++, koji je sam statički tipiziran, bolje implementirati matrice kao potpuno statički tipizirane nego kao hibridne tipove.

Napredovanjem mehanizama templejta i tehnika meta-programiranja ugrađenih u C++11 standard, unapređene su mogućnosti za statičku tipizaciju matrica, što pruža višestruke prednosti tokom razvoja programa. Smatramo da je u statički tipiziranom jeziku, kakav je C++, prirodno koristiti statički tipizirane promenljive gde god je to moguće.

Pored toga što je prirodan, ovakav pristup omogućava kompajleru da u ranim fazama razvoja koda, tj. tokom svakog prevođenja, izvrši proveru slaganja tipova (*type checking*). Rano otkrivanje grešaka u programu je poželjno, jer su greške sve skuplje što se kasnije otkriju. Cena grešaka u programu raste eksponencijalno u funkciji faze razvoja programa u kojoj je pronađena [20]. Pored toga, prepuštanje kompajleru da obavi proveru ispravnosti što većeg dela semantike programa je dragoceno jer na taj način programer ima garanciju da su zadata ograničenja zadovoljena svuda u programu. Nasuprot ovog pristupa, za hibridne matrice kao metod provere ispravnosti semantike (pored inspekcije koda), ostaje testiranje, što je naporan, skup i nepouzdan proces.

2. SAVREMENE BIBLIOTEKE ZA RAD SA MATRICAMA

U ovoj glavi je dat kratak pregled nekoliko značajnih fortranskih i C++ biblioteka koje efikasno implementiraju linearne transformacije vektora i dvo-dimenzionih i više-dimenzionih matrica.

Basic Linear Algebra Subprograms (BLAS)

Kao što ime kaže, BLAS [1, 16] je grupa podprograma koji obavljaju osnovne transformacije iz linearne algebre (elementarne vektorske i matrične operacije). Biblioteka je inicijalno objavljena 1979. godine na jeziku Fortran. Interfejs ove biblioteke danas predstavlja de-fakto standard, tako da za većinu modernih HPC (*high performance computing*) platformi postoji optimizovana BLAS biblioteka i omotači (*wrapper*) za druge programske jezike.

Biblioteka je podeljena u 3 nivoa. Na prvom nivou su definisane skalarno-vektorske i vektorsko-vektorske operacije, na drugom matrično-vektorske, a na trećem matrično-matrične operacije.

Linear Algebra PACKage (LAPACK)

LAPACK [11] je fortranska biblioteka za numeričku linearnu algebru. Ova biblioteka predstavlja proširenje BLAS biblioteke, koju intenzivno koristi. LAPACK pruža operacije za rešavanje sistema linearnih jednačina, pronalaženje svojstvenih vrednosti, faktorizaciju matrica i slično.

Blitz++

Blitz++ [2] je biblioteka razvijena za vektorsku matematiku. Vektori (tj. matrice) predstavljeni ovom bibliotekom mogu da imaju proizvoljan broj dimenzija. Svi preklopljeni C++ operatori izvršavanju operacije nad korespodentnim (*element-wise*) elementima vektora (matrica). Iako nema ugrađen operator za množenje 2-dimenzionih matrica, Blitz++ ima koncepte kao što su kriškanje (*slicing*), pod-nizovi (*sub-arrays*) i generičke indekse (*index placeholders*), koji pružaju široke mogućnosti izražavanja vektorskih izraza.

Blitz++ koristi razne tehnike meta-programiranja za ubrzavanje računanja matematičkih izraza. Između ostalih i templejte izraze (*expression templates*) što joj omogućava da optimizuje izračunavanje vektorskih (matričnih) izraza bez upotrebe BLAS (ili sličnih) biblioteka.

Pored brzine izvršavanja, u ovoj biblioteci je pažnja posvećena i stilu pisanja vektorskih (matematičkih) izraza. Autori biblioteke su se potrudili da obezbede prirodnu matematičku sintaksu i da na taj način pojednostave njenu upotrebu. Ovakvim pristupom su približili biblioteku korisnicima koji nisu eksperti u programiranju (najčešće inženjeri i naučnici), a imaju potrebu za višedimenzionim matričnim računom.

uBLAS

uBLAS [8] je biblioteka koja pruža operacije funkcionalno ekvivalentne onim koje su definisane u BLAS biblioteci. Dizajn i implementacija objedinjuju intuitivnu matematičku notaciju (upotrebom preklapanja operatora) i efikasno generisanje koda (korišćenjem templejta izraza). uBLAS podržava različite tipove vektora i matrica (guste, retke, dijagonalne, simetrične, ...), kao i različite „poglede“ na matrice kori-

šćenjem raspona (*ranges*), kriški (*slices*), adaptera (*adapter classes*) i indirektnih nizova (*indirect arrays*).Vezu između kontejnera, „pogleda“ i izraza čine iteratori nalik onim definisanim u standardnoj C++ biblioteci (*Standard Template Library - STL*).

Armadillo

Armadillo [19] je biblioteka otvorenog koda (*open-source*) dominantno razvijena u NICTA [7] australijskom istraživačkom centru. To je C++ biblioteka prevashodno namenjena za linearnu algebru, za razliku od Blitz++ biblioteke koja sve operacije nad matricama tretira kao skalarne operacije nad njihovim elementima. Armadillo biblioteka je koncentrisana na matrične operacije i u svojoj sintaksi i semantici bliska je programskom jeziku Matlab [5] (i GNU Octave [3] - na nivou izvornog koda, Matlab-u kompatibilnom ekvivalentnu iz grupe projekata otvorenog koda).

U implementaciji Armadilo biblioteke intenzivno se koristi templejt meta-programiranje. Prethodno pomenute uBLAS i Blitz++ biblioteke takođe koriste templejt izraze, tj. prepuštaju kompajleru da nezavisno optimizuje svaki izraz na nivou elementarnih operacija nad elementima matrica. Za razliku od njih, Armadilo mehanizam templejta koristi (samo) da obezbedi odloženu evaluaciju matričnih izraza, a za same algebarske transformacije koristi druge (opšte poznate, izuzetno efikasne i intenzivno testirane) biblioteke kao što su LAPACK [11], BLAS [1] i ATLAS [21].

Matrix Template Library (MTL)

Osnovni cilj autora MTL [6] biblioteke je da korisnicima obezbedi visoke performanse tokom izvršavanja koda i istovremeno visoku produktivnost tokom razvoja programa. Biblioteka pruža lak za korišćenje i intuitivan interfejs. Za optimizaciju izvršavanja koristi istovremeno i templejte izraza i druge biblioteke kao što je BLAS.

Iz ovog kratkog pregleda numeričkih biblioteka se mogu uočiti neki trendovi:

S jedne strane imamo težnju za brzim izvršavanjem matematičkih (matričnih) operacija. Zarad postizanja tog cilja definisana je BLAS biblioteka (i implementirane mnoge platformski-opimizovane verzije), razvijene LINPACK [4] i LAPACK biblioteke (koje proširuju skup optimizovanih operacija) i pokrenut ATLAS projekat (koji obezbeđuje generisanje automatski optimizovane BLAS biblioteke). Sve ove biblioteke su inicijalno razvijene pre dve decenije na jeziku Fortran (uz eventualne C omotače), sa ciljevima dominantno usmerenim ka efikasnosti izvršavanja. Pri tome, ovim projektima efikasnost upotrebe biblioteka (jednostavnost interfejsa i lakoća upotrebe) nije bio jedan od primarnih ciljeva.

Drugi trend koji se može uočiti je povećanje nivoa apstrakcije matričnih biblioteka. Predstavnici ove grupe biblioteka na jeziku C++ su na primer: MTL, Blitz++ i Armadillo. Jedan od dominantnih ciljeva ovih biblioteka je pojednostavljenje interfejsa i na taj način pojednostavljivanje procesa modelovanja složenih matematičkih algoritama, bez (značajnih) gubitaka brzine izvršavanja. U bibliotekama se koriste različiti mehanizmi za obezbeđivanje notacije slične matematičkoj. Zahvaljujući tom, drugom trendu, množenje dve matrice i dodela rezultata trećoj se na jeziku C++ može pisati kao:

```
c = a * b;
```

dok se korišćenjem BLAS biblioteke na Fortranu taj isti iskaz piše:

```
call dgemm('N', 'N', 4, 4, 5, 1.0, a,
4, b, 5, 0.0, c, 4)
```

Smatramo da je povećavanje nivoa apstrakcije evidentan i prirodan trend u bibliotekama za rad sa matricama. Uvođenje još jednog nivoa apstrakcije u implementaciji tipa matrice je, po nama, logičan sledeći korak kojim se dalje može olakšati upotreba matričnih biblioteka.

3. STATIČKI TIPIZIRANE MATRICE

Matematika postavlja razna ograničenja u pogledu dimenzija matrica na operacije linearnih transformacija. Na primer: „da bi dve matrice mogle da se saberu (da bi sabiranje bilo definisano) neophodno je da budu istih dimenzija“ ili „da bi dve matrice mogle da se pomnože (da bi množenje bilo definisano) neophodno je da su odgovarajuće (unutrašnje) dimenzije množenih matrica jednake“.

Sve značajnije C++ biblioteke za rad sa matricama su (koliko je nama poznato) implementirane kao hibridne: tip elementa i broj dimenzija se zadaju statički, a veličine svake od dimenzija se zadaju dinamički. Razlog za ovakvo stanje je postojanje funkcionalnosti koje hibridna implementacija obezbeđuje (o kojima govorimo kasnije), a koje nije moguće ili nije lako (u zavisnosti od jezika koji se koristi) realizovati u potpuno statičkoj implementaciji matrice.

Sa druge strane, ovako realizovane, sve matrice istog tipa elemenata sa istim brojem dimenzija su međusobno istog tipa, tj. sve one kompajleru izgledaju isto (bez obzira na veličine dimenzija). Savremene biblioteke obično tretiraju kao različite tipove matrice sa različitim brojem dimenzija, ali to je i dalje suviše gruba (*coarse grained*) podela da bi bila upotrebljiva za proveru ograničenja u pogledu dimenzija. Ovakva implementacija matrice, koja ne uključuje veličinu dimenzija, sprečava kompajler da obavi statičku verifikaciju koda u pogledu dimenzione ispravnosti matrica u linearnim transformacijama, tj. da proveri da li su ispunjena ograničenja koja matematika nameće.

U ovom radu je predstavljena implementacija matrice u kojoj se tip elemenata, broj dimenzija i veličina dimenzija matrice tretiraju kao jednako bitne karakteristike. Predloženi pristup je implementiran C++ biblioteci pod imenom **Typed Matrix Library** (TML).

TML je templejt biblioteka razvijena sa ciljem da demonstrira upotrebljivost koncepta (*proof-of-concept*) statički

tipiziranih matrica. Ova biblioteka se sastoji od templejta klasa (*class templates*) koji omogućavaju provere semantičke ispravnosti matričnih operacija još u vreme prevođenja programa (*compile-time checking*). U TML-u je implementiran samo minimum neophodnih funkcionalnosti za statičku tipizaciju. Ostale mogućnosti (već decenijama) prisutne u drugim bibliotekama su preuzete (*reused*) iz njih. Matrične operacije u TML-u su realizovane kao omotač oko neke druge biblioteke koje sadrže ne potpuno statički tipizirane matrice. Ovaj pristup je dobar iz više razloga. Pored toga što je efikasan za implementaciju, on omogućava lako poređenje brzine izvršavanja između TML biblioteke i biblioteke koju TML obmotava. Na taj način se lako može izmeriti usporenje (*overhead*) koje TML unosi. Trenutno postoji samo implementacija realizovana nad Armadillo bibliotekom, ali se lako mogu napraviti i implementacije nad drugim bibliotekama (na primer nad MTL).

Ideja za razvoj ovakve biblioteke je proizašla iz sledeće zapažanja: numerički zahtevni programi obično u svom radu koriste mnogo matrica različitih dimenzija. Da bi kompajler mogao da proveri da li je moguće sabrati ili pomnožiti dve matrice, neophodno je još u vreme prevođenja dimenziono odrediti matrice. Konkretne veličine matrica (obično) ne mogu biti zadate pre izvršavanja programa, jer su im dimenzije promenljive i zavise od konkretnog ulaznog skupa (*input set*). Srećom to nije neophodno. Svaka matrica u programu ima (ili bi trebalo da ima) algoritmom definisanu namenu. U skladu sa namenom određene su i njene dimenzije u odnosu na ostale matrice u programu (algoritmu). Ovi odnosi su nepromenjivi, bez obzira na konkretne vrednosti samih dimenzija matrica.

Tokom razvoja programa je moguće identifikovati nezavisne dimenzija matrica i izraziti zavisne preko nezavisnih. Na primer, ako posmatramo dimenzije tri nezavisne, dvodimenzionalne matrice: A, B i C vidimo da one, u opštem slučaju, mogu biti izražene korišćenjem šest nezavisnih skalarnih veličina: x_a, y_a, x_b, y_b, x_c i y_c.

$$C = A * B \quad (1)$$

Ako ove matrice nisu nezavisne, već povezane jednačinom (1) tada, da bi jednačina (1) bila definisana, linearna algebra zahteva da ove matrice budu i dimenziono zavisne. Unutrašnje dimenzije množenih matrica (broj kolona matrice A i broj vrsta matrice B) moraju da budu jednake i broj vrsta matrica C mora da bude jednak broju vrsta matrice A, a broj kolona matrice C mora da bude jednak broju kolona matrice B, tj. uvek moraju da budu ispunjene jednakosti iz jednačine (2).

$$\begin{aligned} y_a &= x_b \\ x_c &= x_a \\ y_c &= y_b \end{aligned} \quad (2)$$

Svaka matrična jednačina uvodi ograničenja (slična ovim) u pogledu veličina dimenzija matrica koje se u njoj pojavljuju. Smatramo da programe treba pisati tako da kompajler može da proveri da li su ova ograničenja zadovoljena.

Tipovi dimenzija matrica

Dimenzija matrica je moguće (i treba) izraziti tokom pisanja programa, na takav način da ih kompajler razume i da na osnovu njih može da obavi statičku verifikaciju ispravnosti koda. Pošto kompajleri razlikuju samo tipove, a ne instance, prvo što pada na pamet je da dimenzije treba predstaviti kao tipove.

Sa druge strane, konkretne vrednosti dimenzija matrica obično nisu poznate pre izvršavanja programa i zato dimenzije matrica treba predstaviti tako da u vreme izvršavanja mogu da im se pridruže vrednosti. Na jeziku C++ se (*run-time*) vrednosti obično dodeljuju instancama tipova (objektima), iz čega sledi da bi dimenzije matrice bilo logično čuvati i kao vrednosti instanci.

Ova dva kontradiktorna zahteva:

- da kompajler međusobno razlikuje dimenzije matrica i
- da konkretna vrednost dimenzije matrice može da bude zadata u vreme izvršavanja;

su dovela do kompleksne implementacije dimenzije matrice, kakva je predstavljena u daljem tekstu.

Pošto kompajler ne može da razlikuje vrednosti, preostalo nam je da dimenzije matrica predstavimo kao tipove (klase) i da u te klase ugradimo mehanizam koji omogućava pridruživanje konkretnih vrednosti samoj klasi i to u vreme izvršavanja.

U tu svrhu smo razvili nekoliko templejta klasa (*class templates*) koje se koriste za predstavljanje dimenzija matrica. Čitavu kategoriju templejt klasa (*template classes*) generisanih od ovih templejta, nazivamo **tipovi dimenzija matrica**.

C++ jezik omogućava dodelu konkretnih vrednosti korisnički definisanim klasama u vreme izvršavanja (iako to nije uobičajen/preporučen način njihove upotrebe). U sledećem primeru je prikazan pojednostavljen primer definicije ovakve klase.

```
struct dimension_type {
     static size_t& size() {
          static size_t _size_ = 0;
          return _size_;
     }
};
```

Statička funkcija članica ‘size()’ iz primera 1. sadrži statičku lokalnu promenljivu ‘_size_’ i vraća referencu na nju. Statička funkcija ‘size()’ može+ da se izvrši nad klasom, a vraća referencu na promenljivu kojoj može da se dodeli vrednost. Statička lokalna promenljiva ‘_size_’ traje tokom celog programa, a ne mora da se inicijalizuje globalno kao statički atribut.

U vreme izvršavanja, vrednost može da se dodeli klasi (iako nijedna instanca klase ne postoji) izrazom:

```
dimension_type::size() = 5;
```

Instanciranje TML matrice

Pri definisanju tipa TML matrice potrebno je zadati 3 templejt parametra: tip elemenata i tipove dimenzija matrice. Pre samog definisanja tipa matrice neophodno je definisati tipove dimenzija:

```
class dimX : public dimension_type<dimX> {};
```

ili upotrebom preprocesorske direktive:

```
dimension(dimY);
```

Posle toga je moguće definisati tip TML matrice, na primer:

```
typedef tml::Matrix<double, dimX, dimY>
Atype;
```

Pre instanciranja same matrice neophodno je zadati konkretne vrednosti dimenzija:

```
tml::setSize<dimX>(3);
tml::setSize<dimY>(5);
```

Posle čega je moguće instancirati objekat matrice, na primer:

```
Atype A;
```

4. INTERFEJS

„Interfejs bi trebao da bude intuitivan i predvidiv. U procesu razvoja interfejsa uvek treba odabrati dizajn koji će najmanje iznenaditi korisnika.“ Ovaj pristup je poznat kao pravilo ili princip najmanjeg iznenađenja [18].

C++ je statički tipiziran (*statically typed*) jezik. Smatramo da su karakteristike biblioteke sačinjene od potpuno statički tipiziranih elemenata prirodne u C++ okruženju. Takođe smatramo da su, za korisnika, karakteristike ovakve biblioteke manje iznenađujuće od karakteristika ekvivalentne biblioteke sačinjene od ne potpuno statički tipiziranih (hibridnih) elemenata.

Inicijalizacija

U savremenim C++ bibliotekama za rad sa matricama, podrazumevani (*default*) konstruktor konstruiše objekat matrice, ali ne zauzima memoriju za smeštanje elemenata matrice. Ovo je jedino moguće i samim tim logično ponašanje podrazumevanog konstruktora. Pošto dimenzije matrice nisu deo tipa matrice, bez dodatnih informacija konstruktor ne može da uradi ništa drugo. Posledica je da objekti kreirani podrazumevanim konstruktorom nisu potpuno spremni za upotrebu i zato biblioteke poput Armadillo dozvoljavaju donekle ručno upravljanje memorijom. Na primer:

```
arma::Mat<double> A, B(4, 4);
```

U prethodnom iskazu za elemente matrice A nije zauzeta memorija, dok za elemente matrice B jeste. Korisniku programeru je prepušteno da odluči da li da odmah zauzme prostor za elemente matrica ili da zauzimanje odloži. Rezultat ovakve fleksibilnosti je situacija u kojoj korisnik mora da vodi računa o trenutnom stanju objekata i da izvršava operacije nad njima u skladu sa njihovim trenutnim stanjem. Za matrice A i B definisane u prethodnom primeru imamo:

```
A(0, 0);    // Greska
B(0, 0);    // OK
```

Pristup elementima matrice A nije definisana operacija jer matrica A nema elemente (odnosno nema ni memorijski prostor u koje bi ti elementi bili smešteni), pa im nije moguće ni pristupiti.

Za razliku od matrice A, pristup elementima matrice B je definisan i dozvoljen.

Nasuprot ovde prikazanog ponašanja Armadillo biblioteke, matrica iz TML biblioteke konstruisana podrazumevanim konstruktorom je potpuno spremna za upotrebu. Dimenzije jesu deo tipa matrice u TML biblioteci tako da podrazumevani konstruktor ima informacije o veličini prostora koji treba da alocira. U stvari, TML matrica ne može da postoji bez definisanih dimenzija i bloka memorije odgovarajuće veličine, za smeštanje elemenata. Ovo je značajna semantička razlika (i razlika u interfejsu) u TML biblioteci u odnosu na klasične biblioteke.

Iako TML matrica ne može da postoji bez definisanih dimenzija i memorijskog bloka za elemente, korisnik može da koristi pravila C++ jezika o dužini trajanja lokalnih promenljivih (*scoping rules*) da upravlja trenutkom uništenja matrice (i time oslobađanja memorije). Značajna razlika u odnosu na biblioteke koje sadrže ne potpuno statički tipizirane matrice je u tome što je za TML matrice to optimizaciona tehnika koja ne može da utiče na ispravnost programa.

Dodela vrednosti

Iskaz dodele skalara matrici je prikazan u sledećem primeru:

```
A = 0.;
```

Ovaj iskaz je, u bibliotekama koje sadrže ne potpuno statički tipizirane matrice (npr. Armadillo), semantički dvosmislen. Matematičar, fizičar ili inženjer (a oni su najčešći korisnici ovakvih biblioteka) bi verovatno prvo pomislili da se na ovaj način vrednost svih elementima matrice postavlja na nulu. S druge strane iskusan korisnik biblioteke zna da je u pitanju redefinisanje dimenzija matrice na 1x1 (i dodela vrednost tom jedinom elementu).

U našoj implementaciji matrice gde su dimenzije matrice određene statički i ne mogu se menjati, prethodni primer je nedvosmislen. Dodela skalara matrici dovodi do postavljanja vrednosti svih elemenata matrice (bez obzira na njene trenutne dimenzije) na zadatu, što je u programima mnogo češće potrebno od inicijalizacije matrice dimenzija 1x1.

Isto tako, dodela vektora vrste (kolone) matrici, ne menja matricu u vektor, već postavlja vrednosti svih vrsta (kolona) u matrici na vrednosti iz vektora sa desne strane operatora dodele.

Kopirajući konstruktor (*copy constructor*)

Posmatraćemo na primer Blitz++ biblioteku. Kopirajuća operacija dodele u ovoj biblioteci ima kopirajuću semantiku. Ako su A i B matrice, iskaz:

```
A = B;
```

kopira sadržaj elemente matrice B u matricu A. Po završetku operacije dodele, obe matrice imaju iste elemente u različitim blokovima memorije. Nasuprot ovome, kopirajući konstruktor je dizajniran kao nekopirajuća operacija. Iskaz:

```
blitz::Array<double, 2> A(B);
```

stvara matricu A koja sadrži iste elemente kao matrica B. Posle ovog iskaza matrice A i B referenciraju isti memorijski prostor.

Dizajn prethodne dve operacije je semantički kontradiktoran sa usvojenom C++ praksom da kopirajući operator dodele (*copy assignment operator*) i kopirajući konstruktor treba da se ponašaju na isti način.

Kompajleru je dozvoljeno da iskaze koji kombinuju podrazumevani konstruktor i operator dodele, kao u:

```
blitz::Array<double, 2> A = B;
```

optimizuje i da umesto ove dve operacije pozove kopirajući konstruktor. Ova optimizacija je ispravna samo ako su kopirajući konstruktor i kopirajući operator dodele funkcionalno identični i zato to C++ zahteva.

TML trenutno ne podržava referenciranje. Sve operacije implementiraju kopirajuću semantiku (*copy semantic*). Ova vrsta optimizacije će biti implementirana kasnije, kada budu podržane hijerarhijske matrice.

5. PRIMER UPOTREBE

Deo koda koji množi matrice po jednačini (1) bi korišćenjem Armadillo biblioteke mogao da izgleda kao u sledećem primeru:

```
// define dimension variables
size_t dimX;
size_t dimY;
size_t dimZ;

// set dimensions
dimX = VALUE
// ...

// define matrices
arma::mat A(dimX,dimY);
arma::mat B(dimY,dimZ);
// set A and B values

// ...

// multiply && asign
arma::mat C = A * B;
```

Kompajler na osnovu ovakvog koda ne može ništa da zna o dimenzijama matrica, jer one u vreme prevođenja nisu poznate. Da li je množenje matrica A i B definisano može da se proveri samo testiranjem.

Program iste funkcionalnosti, u kojem se koristi TML biblioteka, bi mogao da izgleda kao u ovom primeru:

```
// declare dimension names
dimension(dimX);
dimension(dimY);
dimension(dimZ);

// set dimensions
dimX::setSize(VALUE);
// ...

// define matrices
tml::Matrix<double, dimX, dimY> A;
tml::Matrix<double, dimY, dimZ> B;

// set A and B values
// ...
// multiply && asign
tml::Matrix<double, dimX, dimZ> C = A
* B;
```

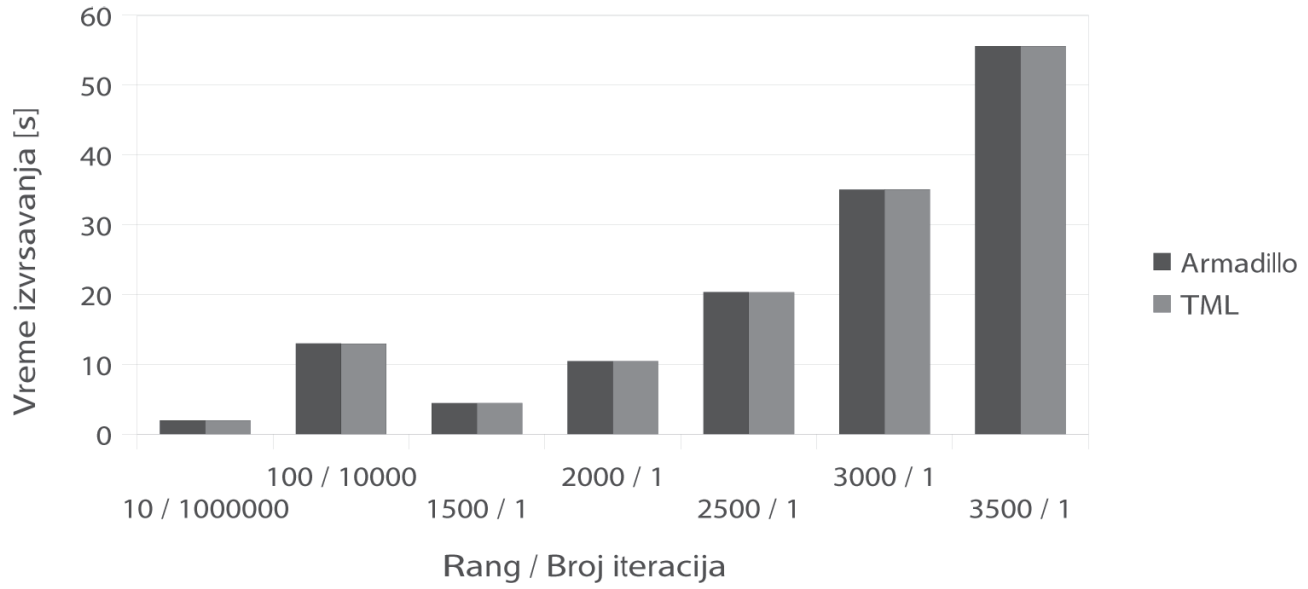
Prvo se uvode tipovi dimenzija - pošto kompajler radi sa tipovima (a ne promenljivama). Tipovi dimenzija su u stvari imena klasa, deklarisanih preprocesorskom direktivom 'dimension()'.

Zatim je potrebno svakoj dimenziji dodeliti konkretnu vrednost[1]. Ovo se događa u vreme izvršavanja. Vrednost se mora dodeliti dimenziji pre instanciranja matrice koje koriste tu dimenziju.

Zatim se definišu matrice. Kao prvi templejt argument se navodi tip elemenata matrice, a zatim se navode još dva argumenta koji predstavljaju dimenzije matrice. Dimenzije se zadaju korišćenjem imena uvedenih u prvom koraku.

Veličina sistema (dimenzije matrica) i dalje nije poznata u vreme prevođenja, ali nad ovako definisanim matricama A, B i C kompajler može da garantuje da će dimenzije, kolike god da budu, uvek zadovoljavati invarijante iz formule (2).

[1] Funkcija 'setSize()' je bazirana na mehanizmima prikazanim u funkciji 'size()' klase 'dimension_type' predstavljenoj u glavi 3.



Slika 1: *Uporedni prikaz vremena izvršavanja programa koji koriste Armadillo i TML biblioteku.*

6. EFIKASNOST IMPLEMENTACIJE

U svrhu ispitivanja eventualnog usporenja izvršavanja programa koji koriste TML biblioteku (u odnosu na one koji direktno koriste Armadillo biblioteku), obavljena je serija merenja vremena izvršavanja množenja dve kvadratne matrice različitih veličina, programima koji koriste ove dve biblioteke. Svako merenje je izvršeno po 10 puta. Na osnovu ovih rezultata je izračunata srednja vrednost trajanja izvršavanja programa.

Na slici 1. su prikazana (na ordinati) srednja vremena izvršavanja množenja dve kvadratne matrice. Na apscisi su za svako merenje prikazani rang množenih matrica i broj uzastopnih množenja koja su zajedno merena, da bi se dobilo precizno merljivo (ne suviše malo) vreme.

Za svaku veličinu množenih matrica prikazana su po dva stubića. Levi predstavlja vreme izvršavanja programa u kojem se direktno koristi Armadillo biblioteka, a desni vreme izvršavanja programa u kojem se koristi TML. Sa slike 1. se vidi da izvršavanja i Armadillo i TML programa (za istu veličinu množenih matrica) traju jednako, bez obzira na veličinu samih matrica.

Na osnovu rezultata sa slike 1. zaključujemo da je usporenje koje TML biblioteka unosi u izvršavanje (ako i postoji) zanemarljivo. Ovo su očekivani rezultati jer je TML templejt biblioteka, dizajnirana i razvijena tako da u toku prevođenja kompajler može da obavi semantičke provere, a da generisani kod ne radi ništa više od koda generisanog direktnim korišćenjem Armadillo biblioteke. Detaljnija analiza rezultata merenja brzine izvršavanja se može naći u [17].

7. ZAKLJUČAK

Analizirali smo nekoliko savremenih C++ biblioteka za rad sa matricama. Ukazali smo na nedostatke u njihovom dizajnu, to jest na neusaglašenost interfejsa i semantike na pojedinim mestima. Smatramo da su uočeni nedostaci posledica hibridne (delimično statičke, a delimično dinamičke) implementacije tipa matrice. Takođe, smatramo da je u statički tipiziranom jeziku kakav je C++ korisno, što je moguće češće, upotrebljavati statičku tipizaciju. Na taj način se kompajler može iskoristiti za rano i pouzdano otkrivanje grupe semantičkih grešaka vezanih za dimenzionalnost matrica u operacijama linearnih transformacija.

Pored ovog naveli smio i razloge zbog kojih jednostavna statička implementacija matrica najčešće nije prihvatljiva: deo karakteristika tipa (konkretno veličine dimenzija) matrice obično nije poznat u vreme prevođenja.

Identifikovali smo ove dve grupe kontradiktornih zahteva: sa jedne strane potrebna je statička tipizacija implementacije matrice, a sa druge strane je potrebno ostaviti mogućnost da se veličine dimenzija razreše u vreme prevođenja. Zatim smo ponudili rešenje u vidu C++ biblioteke za rad sa matricama pod nazivom TML. Ova biblioteka uvodi sloj indirekcije u implementaciju dimenzija matrice. Veličine dimenzija matrice su predstavljene posebnom klasom tipova nazvanom „tipovi dimenzija“, nad kojom kompajler može da radi. Ova klasa tipova omogućava razrešavanje veličina dimenzija u vreme izvršavanja.

Smatramo da smo u nekoliko jednostavnih primera pokazali da TML biblioteka:

- ima jasnu i jednostavnu semantiku,
- od programera se ne zahteva raspolaganje suštinski novim znanjima i
- unosi praktično zanemarljivo usporenje (*overhead*) izvršavanja programe koji je koriste.

Smatramo da ovde predložen način implementiranja matrica pruža nove mogućnosti u modelovanju hijerarhijski organizovanih matrica. Savremene biblioteke, po našem mišljenju, nemaju jednostavan interfejs za rad sa matricama čiji su elementi druge matrice. Takođe, ne pružaju zadovoljavajuću

fleksibilnost u pri raspoređivanju elemenata ovakvih matrica u memoriju računara. Primena ovde izloženog načina implementacije matrica (upotrebom tipova dimenzija) će biti jedan od pravaca daljih istraživanja.

LITERATURA

[1] BLAS reference implementation, http://www.netlib.org/blas/, nov. 2011.
[2] Blitz++ library, http://sf.net/projects/blitz/, jul 2012.
[3] GNU Octave, http://www.gnu.org/software/octave/, jun 2012.
[4] LINPACK reference implementation, http://http://www.netlib.org/linpack/, nov. 2012.
[5] MathWorks MATLAB, http://www.mathworks.com/products/matlab/, jun 2012.
[6] MTL library, http://www.osl.iu.edu/research/mtl/, feb. 2012.
[7] NICTA, http://nicta.com.au/, dec. 2011.
[8] uBLAS library; http://www.boost.org/doc/libs/1_49_0/libs/numeric/ublas/, feb. 2012.
[9] Wikipedia: Type system, http://en.wikipedia.org/wiki/type_system#type_checking; jul 2012.
[10] M. Abadi, L. Cardelli, B. Pierce, and G. Plotkin. Dynamic typing in a statically-typed language. In Proceedings of the ACM Symposium on Principles of Programming Languages, pages 213#227, 1989.
[11] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.
[12] J. J. Barton and L. R. Nackman. *Scientific and Engineering C++: An Introduction with Advanced Techniques and Examples*. Addison-Wesley Professional; 1 edition, 1994.
[13] C. Flanagan. Hybrid type checking. SIGPLAN Not., 41(1):245#256, Jan. 2006.
[14] M. T. Heath. *SCIENTIFIC COMPUTING: An Introductory Survey*. McGraw-Hill, 1997.
[15] K. Knowles, A. Tomb, J. Gronski, S. N. Freund, and C. Flanagan. Sage: Unified hybrid checking for first-class types, general refinement types, and dynamic (extended report). http://sage.soe.ucsc.edu/sage-tr.pdf, 2006.
[16] C. L. Lawson, R. J. Hanson, D. Kincaid, and F. T. Krogh. Basic linear algebra subprograms for fortran usage. *ACM Transactions on Mathematical Software*, 5:308-323, 1979.
[17] P. S. Rakić, L. Stričević, and Z. Suvajdžin Rakić. Provera odnosa dimenzija matrica u vreme prevodjenja. In YU INFO (18; Kopaonik), pages 629-634. Društvo za informacione sisteme i računarske mreže, 2012.
[18] E. S. Raymond. *The Art of UNIX Programming*. Addison-Wesley Professional, 2003. ISBN 978-0131429017.
[19] C. Sanderson. Armadillo: An open source C++ linear algebra library for fast prototyping and computationally intensive experiments. Technical report, NICTA, 2010.
[20] J. Westland. The cost of errors in software development: evidence from industry. *Journal of Systems and Software*, 62(1):1-9, 2002.
[21] R. C. Whaley, A. Petitet and J. J. Dongarra. Automated Empirical Optimization of Software and the ATLAS Project, *Parallel Computing* 27, 1--2 (2001), pp. 3--35.

Predrag S. Rakić,
Fakultet tehničkih nauka Novi Sad
pec@uns.ac.rs
Oblasti interesovanja: C++, konkurentno programiranje, templejt meta-programiranje

Lazar Stričević,
Fakultet tehničkih nauka Novi Sad
lucky@uns.ac.rs
Oblasti interesovanja: arhitektura računara, računarske mreže, distribuirani računarski sistemi

Zorica Suvajdžin Rakić
Fakultet tehničkih nauka Novi Sad
tweety@uns.ac.rs
Oblasti interesovanja: kompajleri, programski jezici

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, apstrakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M