Traženje eliptičkih krivulja velikoga ranga

Vinko Petričević
vpetrice@math.hr

PMF-MO Zagreb

16, ožujka 2022.

Funding

This work was supported by the QuantiXLie Centre of Excellence, a project co-financed by the Croatian Government and European Union through the European Regional Development Fund - the Competitiveness and Cohesion Operational Program (Grant KK.01.1.1.01.0004).

Za više informacija posjetite:
http://bela.phy.hr/quantixlie/hr/
https://strukturnifondovi.hr/

For more information:
http://bela.phy.hr/quantixlie/hr/
https://strukturnifondovi.hr/

Sadržaj ove prezentacije isključiva je odgovornost Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu te ne predstavlja nužno stajalište Europske unije.

The content of this presentation is exclusive responsibility of the Faculty of Science University of Zagreb and does not represent opinion of the European Union

EUROPSKA UNIJA
Zajedno do fondova EU

EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Operativni program KONKURENTNOST I KOHEZIJA

Projekt sufinancira Europska unija iz Europskog fonda za regionalni razvoj

Project co-financed by European Union through the European Regional Development Fund

Motivacija

O detaljima o eliptičkim krivuljama, torzijama, rangu, povijesnom razvoju i ostalom puno se može naći na stranicama prof. Dujelle:
`https://web.math.pmf.unizg.hr/~duje/tors/tors.html`

U novije vrijeme nekoliko novih primjera smo pronašli:

- U $\mathbb{Q}$ za $\mathbb{Z}/9\mathbb{Z}$
- U kvadratnim poljima: $\mathbb{Z}/3\mathbb{Z} \times \mathbb{Z}/3\mathbb{Z}$ i $\mathbb{Z}/4\mathbb{Z} \times \mathbb{Z}/4\mathbb{Z}$

U skripti s predavanja prof. Dujelle
`https://web.math.pmf.unizg.hr/~duje/diofelip/diofelip.pdf`
mogu se ću formule eliptičkih krivulja zadane određene torzijske grupe.

Problem je taj da za veliki broj eliptičkih krivulja treba ispitati koliko im je rang. U različitim matematičkim paketima postoje naredbe kojima možemo više ili manje dobro izračunati rang.

Od 2020. u programski paket PARI/GP uvedena je funkcija `ellrank` koja poprilično dobro i brzo računa rang.

Stabilne verzije se mogu naći na
https://pari.math.u-bordeaux.fr/download.html
Razvojne verzije su na
http://pari.math.u-bordeaux.fr/pub/pari/snapshots/. U linuxu skinemo najnoviju verziju:

```
wget http://pari.math.u-bordeaux.fr/pub/pari/snapshots/
    pari-2.14-1701-geeda577921.tar.gz

tar xf pari-2.14-1701-geeda577921.tar.gz
cd pari-2.14-1701-geeda577921/

./Configure --prefix=GPDIR --mt=pthread
make -j4 gp
make install-bin-sta

./Configure -prefix=GPDIR.dbg -g
make -j4 gp.dbg
make install -C Olinux-x86_64.dbg

GPDIR/bin/gp
```

Startamo ju sa

`/home/vpetrice/pariRank/pari-2.14-1701-geeda577921/gp`
ili
`./Olinux-x86_64/gp-sta`

Na stranici
`https://pari.math.u-bordeaux.fr/Events/PARI2022/talks/ellrank.pdf`
možemo vidjeti kako se radi s funkcijom `ellrank`.

Od 2020. pari ima podršku za višedretveno računanje. Prilikom instalacije trebamo navesti želimo li `--mt=pthread` ili `--mt=MPI`. Na stranici `https://pari.math.u-bordeaux.fr/pub/pari/manuals/2.13.3/parallel.pd` imamo uvodnik u rad u višedretvenom sustavu.

Pari, kao i ostale moderne matematičke programe možemo jednostavno koristiti i efektno napraviti prilično napredne stvari, ali takvi programi nisu baš previše brzi. Jedan od razloga je da se radi o interpreteru, za razliku od npr. C-a koji je prevodioc.
Dobra stvar je da Pari ima mogućnost pisanja dijela kôda u C-u.

Promotrimo sljedeći jednostavan program (koji računa zbroj prvih 2^{25} prirodnih brojeva):

```
ff(d)=my(s=0);{
  for(i=1,d,s=s+i);
  return(s);
}
print(ff(2^25)); quit
```

Na mom laptopu mu treba $\approx$ 4.8 sekudi. Na `euler.math.hr` oko 2.45 sekundi). Na mom laptopu u Mathematici mu treba skoro 20 sekundi, dok mu na euleru na magmi treba oko 2.4 sekunde (dok na npr. diophant2 2.8sec).
U C-u mu treba 0.02 sec!!! (obratimo pažnju da podatke učitavamo, jer će u protivnom kompajler to izračunati za vrijeme kompajliranja).

Rezultat je svugdje isti: 562 949 970 198 528.

Razvojne verzije pari-a su nešto sporije. Normalnoj (dinamički linkanoj) multithreaded treba 6.8 sec, singlethreaded 6.3 sec, dok je statički linkana u oba slučaja oko 3.6sec.

Uz pari dolazi i alat gp2c pomoću kojeg .gp program možemo prebaciti u C program, te ga eventualno kompajlirati.
U okviru njega imamo program `gp2c-run` pomoću kojega možemo pokrenuti .gp program. Program generira C kod, te ga automatski kompajlira, te starta gp sa učitanim tim novim naredbama. Sada se isti program izvrši za malo manje od 1.5 sec.

Ukoliko ne želimo *gubljenje* memorije, program pokrenemo sa opcijom -g, te će nam kreirati tkzv. repile kôd, koji oslobađa memoriju koju je koristio. (po defaultu ju ne oslobađa!)

Ukoliko pogledamo sadržaj foldera nakon pokretanja gp2c-run programa, on nam je krirao i neke dodatne datoteke (.run, .so).

Iz normalnog gp-a možemo prekopirati sadržaj datoteke .run, te koristiti novokompajlirane naredbe bez ponovnog kompajliranja.

Pa ipak, još je uvijek velika razlika u brzini izvođenja programa pozvanog direktno iz C-a i GP-a.

Promotrimo prvo još jedan primjer. Izračunajmo n!

Za izračunati $2^{17}!$, pariu treba oko 2.5sec (najbržoj verziji oko 1.9sec)

Identično napisan program u C-u se ponovo izvrši izuzetno brzo, ali daje rezultat 0.

To se dogodi jer C po defaultu računa s brojevima do 2^{64} (odnosno, ugrađenim tipovima bi mogao do 2^{128}). Međutim, točan rezultat je nešto veći od $10^{600\,000}$. Da bismo dobili točan rezultat za proizvoljno velik broj, treba nam neki library za rad s velikim brojevima.

Sa Linuxom standardno dolazi `gmp`, a može se instalirati i na Windowsima. Sada umjesto `long` (`__uint128_t`) koristimo `mpz_class`.

Kada ovaj program pokrenemo u C-u, treba mu oko 1.4 sec.

Vratimo se sada prvom primjeru. gp2c-run osim pokretanja .gp programa podržava i pokretanje .c programa, pa recimo budući da znamo da će nam rezultat biti *dovoljno* mali, možemo parijev tip podatka `GEN` tip zamijeniti sa `long`.

Novi program se izvrši za 14ms, što je isto kao u C-u.

Moguća je i obrnuta stvar, da iz C-a pozivamo pari naredbe, i da takav program kompajliramo i linkamo diretno iz terminala. Možemo pozivati sve pari naredbe, i one će se izvršavati jednako brzo kao u pariu (zavisi koja nam je trenutno verzija instalirana, u njoj će i raditi C-program).

Kada tražimo eliptičke krivulje velikoga ranga, zadane torzijske grupe, prvo trebamo pogledati kako izgledaju takve krivulje. Svi oblici se mogu naći u skripti prof. Dujelle na stranici 26
`https://web.math.pmf.unizg.hr/~duje/elkript/elkripto2.pdf`
Nakon što odaberemo familiju, da ne bismo pretraživali sve krivulje, dobro bi bilo prvo *prosijati* dobre kandidate (formule za sume se mogu naću u istoj skripti na stranici 47).

Nakon što smo napisali početnu verziju programa, ima tu dosta prostora za ubrzanje. A za sve torzijske grupe, programi su jako slični.

Promotrimo primjer grupe $\mathbb{Z}/9\mathbb{Z}$. Opći oblik eliptičke krivulje je $[A, B, C, 0, 0]$

$$A = u2^3 - u1^3 + u1^2 * u2,$$
$$B = -u1^2 * (-u2 + u1) * (u2^2 - u1 * u2 + u1^2) * u2,$$
$$C = -u1^2 * (-u2 + u1) * (u2^2 - u1 * u2 + u1^2) * u2^4,$$

gdje su $u1$ i $u2$ relativno prosti prirodni brojevi (tj. $u1/u2 \in \mathbb{Q}$).

Trenutni rekord je 4, a da ne bismo predugo čekali, ispisuju se i krivulje ranga 3.
Promotrimo kako bi izgledala jedna moguća optimizacija:
Program `z9pari1.gp` traži sve krivulje za $u1 + u2 \in [3..50]$. Izračun traje oko 8 sekundi (takve stvari smo računali obično do 5000, pa se razlike u vremenima bitno povećavaju).

Neke stvari se možda nepotrebno računaju (redukcije mod p), pa bi te dijelove mogli izbaciti. Tu dobijemo oko pola sekunde uštede. `z9pari2.gp`
Međutim neke stvari se neptrebno računaju više puta, pa te dijelove možemo unaprijed izračunati i zapamtiti (`z9pari3.gp`) Na uzorku do 50, ne vidi se razlika, čak i malo sporije radi.

Kao prvo, možemo iskoristiti više jezgri, te pokrenuti višedretveni oblik (`z9pari4.gp`).
Najdulji izračun se računa na kraju, pa bismo mogli promijeniti redoslijed računanja, te za kraj ostaviti najkraće (`z9pari5.gp`).

U drugim torzijskim grupama, osim *ap*-ova, mogli bismo pamtiti i kronekerov simbol, a ako imamo višeparametarsku familiju, bilo bi dobro unaprijed izračunati i *gcd*-ove.

Cijeli program možemo pomoću alata gp2c prebaciti u C, a možemo i samo dijelove prekopirati, pa iskoristiti pari iz C-a. Da bismo to napravili, trebamo preći u folder u kojem je on instaliran:

```
cd ~/pari/gp2c-0.0.11pl2
./gp2c -g ~/seminar/z9pari1.gp > z9pari1.c
```

Zavisi o našoj spretnosti, dobar dio stvari možemi optimizirati i u C++-u, a da bismo ga uspješno kompajlirali i linkali, treba napisati:

```
g++ -O3 -Wall -ffp-contract=off -fno-strict-aliasing -fPIC
-I/home/vpetrice/pariRank/pari-2.14-1609-g858311d9f3/GPDIR/include
z9.cpp -fPIC -lc -lm
-L/home/vpetrice/pariRank/pari-2.14-1609-g858311d9f3/GPDIR/lib
-lpari -lgmp -lgmpxx -o z9 -Wl,-rpath
/home/vpetrice/pariRank/pari-2.14-1609-g858311d9f3/GPDIR/lib
```

Na primjer za usporebu samo računanje suma do 1000, u normalnom pari-u (gp) treba 73 sekunde, u statički linkanom (`gp-sta`) oko 50 sekundi, dok u C-u treba oko 2 sekunde (i nešto manje od 3 sekunde za inicijalizaciju – koju možemo i ubrzati).

Može se i cijeli program napisati u C-u, ali možda bi jednostavnije bilo u ovome dijelu rezultate zapisati u datoteku, te prepraviti pari program koji samo čita te podatke te računa rang pripadne krivulje (zbog mogućih komplikacija s višedretvenošću u C-u i povezivanju paria s njim). Primjer `z9citaj.gp`

Zahvaljujem na Vašoj pažnji