Složenost algoritama
6. predavanje

Saša Singer

singer@math.hr

web.math.hr/~singer

PMF – Matematički odjel, Zagreb

Sadržaj nekoliko sljedećih predavanja

Model složenosti i eksperimentalna provjera modela na

- tri jednostavna (iterativna) algoritma sa složenostima različitih redova veličina:
  - Parcijalne sume reda za $\ln 2$ — složenost $\Theta(n)$;
  - Zbrajanje matrica reda n — složenost $\Theta(n^2)$;
  - Množenje matrica reda n — složenost $\Theta(n^3)$.

Uz analizu rezultata eksperimenta, napravit ćemo još i

- ubrazanje izloženih algoritama (kad ide):
  - Brzo i točno računanje sume za $\ln 2$;
  - Hijerarhijska građa memorije i pristup podacima;
  - Cache memorija i uloga;
  - Blok–algoritmi za iskorištavanje cachea;

Model složenosti

U svakom od tri algoritma koje ćemo napraviti

- elementarne operacije su osnovne aritmetičke operacije na realnim brojevima (tzv. "floating–point" operacije, IEEE tip double).

Ključna pretpostavka za model složenosti je:

- trajanje tih operacija ne ovisi o operandima.

Dakle, složenost ovih algoritama mjerimo

- brojem floating–point operacija.

Napomena. Gornja pretpostavka ne znači da je trajanje floating–point operacija konstantno.

- Upravo to ćemo testirati!

Model složenosti (nastavak)

Prvi korak — brojanje.

Jednostavnim prebrojavanjem ovih operacija dolazimo do

- ukupnog broja floating–point operacija,
- u funkciji odabranog parametra n, koji mjeri veličinu problema.

Oznaka:

$$F(n) = \text{neka funkcija od } n.$$

Slovo F asocira na "Flops", što je kratica za "floating–point operations" (ali ne "per second").

Dakle, samo broj operacija, a ne brzina izvođenja.

Model složenosti (nastavak)

Drugi korak — brzina.

Kad bi brzina c izvođenja floating–point operacija bila konstantna,

- recimo, $c = 3$ Gigaflopsa, tj. $3 \cdot 10^9$ floating–point operacija u sekundi,

onda bi vremenska složenost $T(n)$ bila

$$T(n) = \frac{F(n)}{c},$$

iz $F(n) = c \cdot T(n)$ (broj operacija je brzina puta vrijeme).

A to lako možemo eksperimentalno provjeriti.

Model složenosti (nastavak)

Treći korak — mjerenje vremena i brzine.

Ako programskom "štopericom"

- izmjerimo vremensku složenost $T(n)$,
- i to za razne vrijednosti od n,

onda brzinu izvođenja operacija c možemo odrediti kao funkciju od n, iz modela

$$c(n) = \frac{F(n)}{T(n)}.$$

Iz tablica ili grafova vrijednosti $c(n)$ lako je provjeriti da li je to konstantno ili ne.

Što testiramo?

Naš eksperiment se svodi na

- mjerenje brzine izvršavanja algoritma (odn. osnovnih operacija) u funkciji od n.

Pa da vidimo!

Napomena. Za male vrijednosti od n

- vrlo je teško dovoljno točno mjeriti vremena $T(n)$

(rezolucija timera je, uglavnom, fiksna).

Dakle, čitav eksperiment treba relativno pažljivo isplanirati, da bismo dobili iole razumne rezultate.

Kako testiramo (mjerimo vrijeme)?

Realizacija: za svaki fiksni n

- "isti" posao ponavljamo u petlji puno puta,
- a vrijeme mjerimo jednom, za cijelu petlju.

To, naravno, nije isto kao bez ponavljanja, ali je jedino praktično izvedivo!

Neka je $N(n)$ izabrani broj ponavljanja posla, za dani n.

Brzinu $c(n)$ računamo iz

$$c(n) = \frac{N(n) \cdot F(n)}{T(N(n) \text{ puta ponovi posao za } n)}.$$

Što želimo dobiti?

Prije nego što "vidimo" rezultate na nekom konkretnom računalu,

- nije baš jasno da ćemo dobiti nešto "zanimljivo".

Hoćemo, hoćemo — obećavam! Bit' će zanimljivi i na jednom računalu.

Dakle, čemu još rezultati mogu poslužiti?

Ovako dobivene brzine $c(n)$, naravno,

- bitno ovise o brzini računala na kojem testiramo.

Zato je zgodno test napraviti na

- nekoliko raznih računala i usporediti rezultate.

Svrha eksperimenta (nastavak)

Osim za usporedbu računala, stvar može poslužiti i za

- usporedbu različitih programskih produkata, na pr. compilera, kao odgovor na pitanje:
  - koliko efikasan kôd generiraju, posebno s raznim opcijama?

Zato cijeli eksperiment radimo na

- nekoliko računala,
- a usput ćemo usporediti i dva FORTRAN compilera s različitim opcijama za optimizaciju.

"Zvjerinjak" (računala)

"Stradalnici" u eksperimentu su (ukratko):

- Pentium 3 na 500 MHz, zvani Klamath5,
  rođen 1998., (nekad Pentium 2 na 333 MHz);
- Pentium 4 na 3.0 GHz, zvani Veliki,
  rođen 2003., (danas ima P4 na 3.2 GHz);
- Pentium 4/660 na 3.6 GHz, zvan(a) BabyBlue,
  rođen(a) 2005., (najmlađa "beba" u kući);
- ovaj notebook s Pentium 4 Mobile na 2.2 GHz,
  rođen 2003.

Podrobnije "osobne iskaznice" slijede!

FORTRAN compileri i opcije

Zašto FORTRAN (i to 77)?

- Zato što je to, još uvijek, osnovni jezik za numeričko računanje (recimo, BLAS je originalno napisan u FORTRANu),
- i zato što imam tzv. MKL (Math Kernel Library) koji se zove iz FORTRANa.

FORTRAN compileri u testu:

- Compaq Visual Fortran, verzija 6.6C3 (konačna), oznaka CVF,
- Intel Visual Fortran, verzije razne, oznaka IVF.

FORTRAN compileri i opcije (nastavak)

Opcije za optimizaciju:

- puna optimizacija, skraćeno "fast":
  - `/optimize:5` za CVF,
  - `/fast` za IVF.
- normalna optimizacija, bez posebnih opcija, skraćeno "normal".

Parcijalne sume reda

Parcijalne sume reda za $\ln 2$

Problem: Zadan je prirodni broj $n \in \mathbb{N}$. Treba izračunati n-tu parcijalnu sumu alternirajućeg reda za $\ln 2$

$$S(n) = \sum_{i=1}^{n} (-1)^{i-1} \frac{1}{i}$$

$$= 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \cdots + \frac{(-1)^{n-1}}{n}.$$

Ovu sumu računamo na dva načina:

- zbrajanjem unaprijed,
- zbrajanjem unatrag.

Zbrajanje unaprijed

```
      subroutine sumfwd (n, sum, cumsum)
c
c   Forward sum of ln2 alternat. series, n terms.
c
      implicit none
c
      integer n
      double precision sum, cumsum
c
      integer i, nn
c
c   I loop, inner
c
```

Zbrajanje unaprijed (nastavak)

```
      nn = n
      sum = 0.0d0
      do 10, i = 1, nn
         if (mod(i, 2) .eq. 1) then
            sum = sum + 1.0d0 / i
         else
            sum = sum - 1.0d0 / i
         end if
   10    continue
      cumsum = cumsum + sum
c
      return
      end
```

Zbrajanje unatrag

Potprogram dobivamo zamjenom petlje unaprijed:

```
      do 10, i = 1, nn
```

sljedećom petljom unatrag:

```
      do 10, i = nn, 1, -1
```

Sve ostalo (osim imena potprograma) ostaje isto.

Broj operacija

U svakom prolazu kroz petlju imamo dvije operacije

- dijeljenje s `i`,
- zbrajanje ili oduzimanje.

Tome treba dodati jedno zbrajanje na dnu, izvan petlje, za tzv. kumulativnu sumu (optimizacija).

Dakle, ukupan broj operacija u oba algoritma je:

$$F(n) = 2n + 1.$$

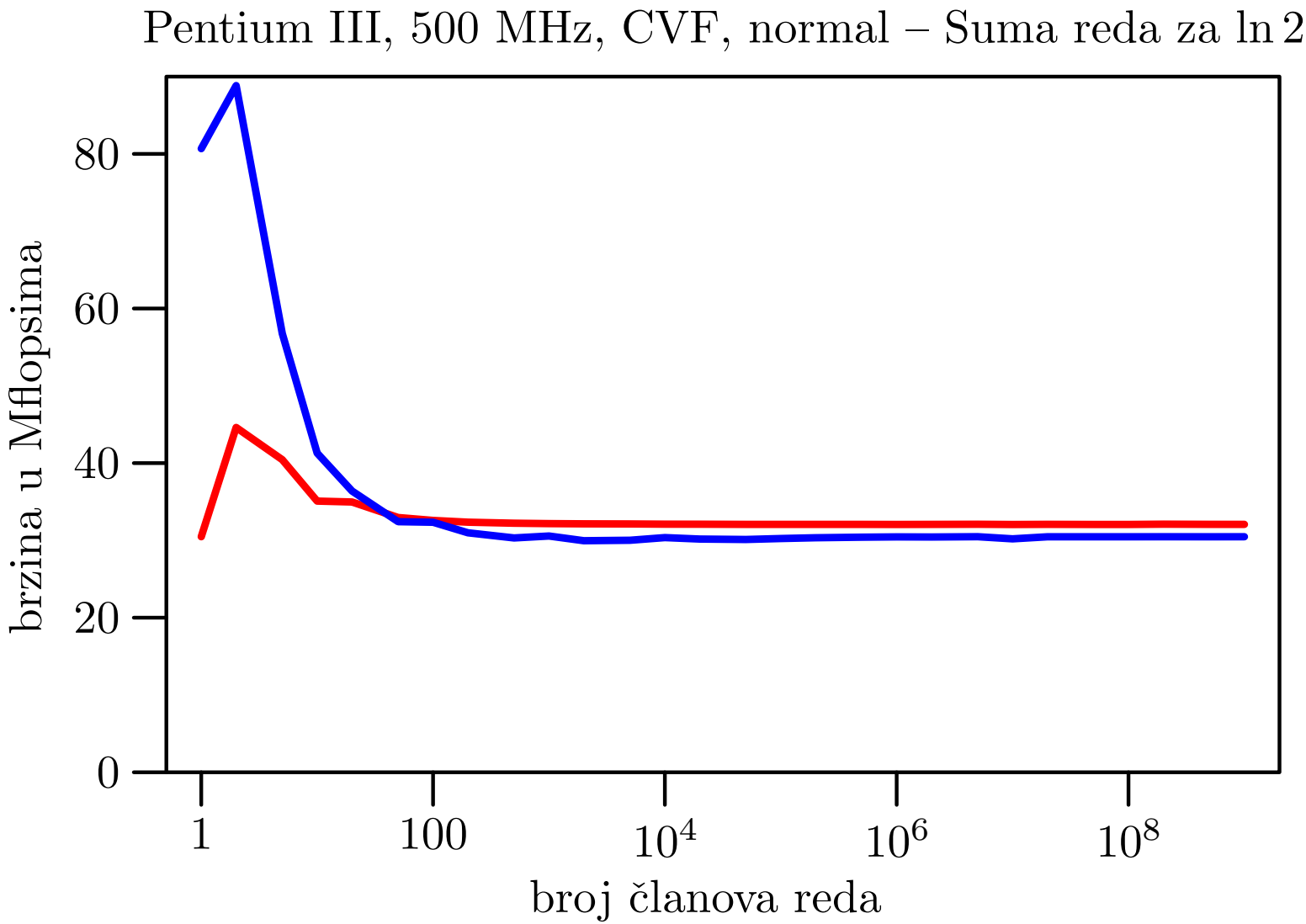
Broj ponavljanja je $N(n) = 10^9/n$, tako da dobijemo približno konstantno trajanje "vanjske" petlje kojoj mjerimo vrijeme (ona s ponavljanjem).

Boje na grafovima

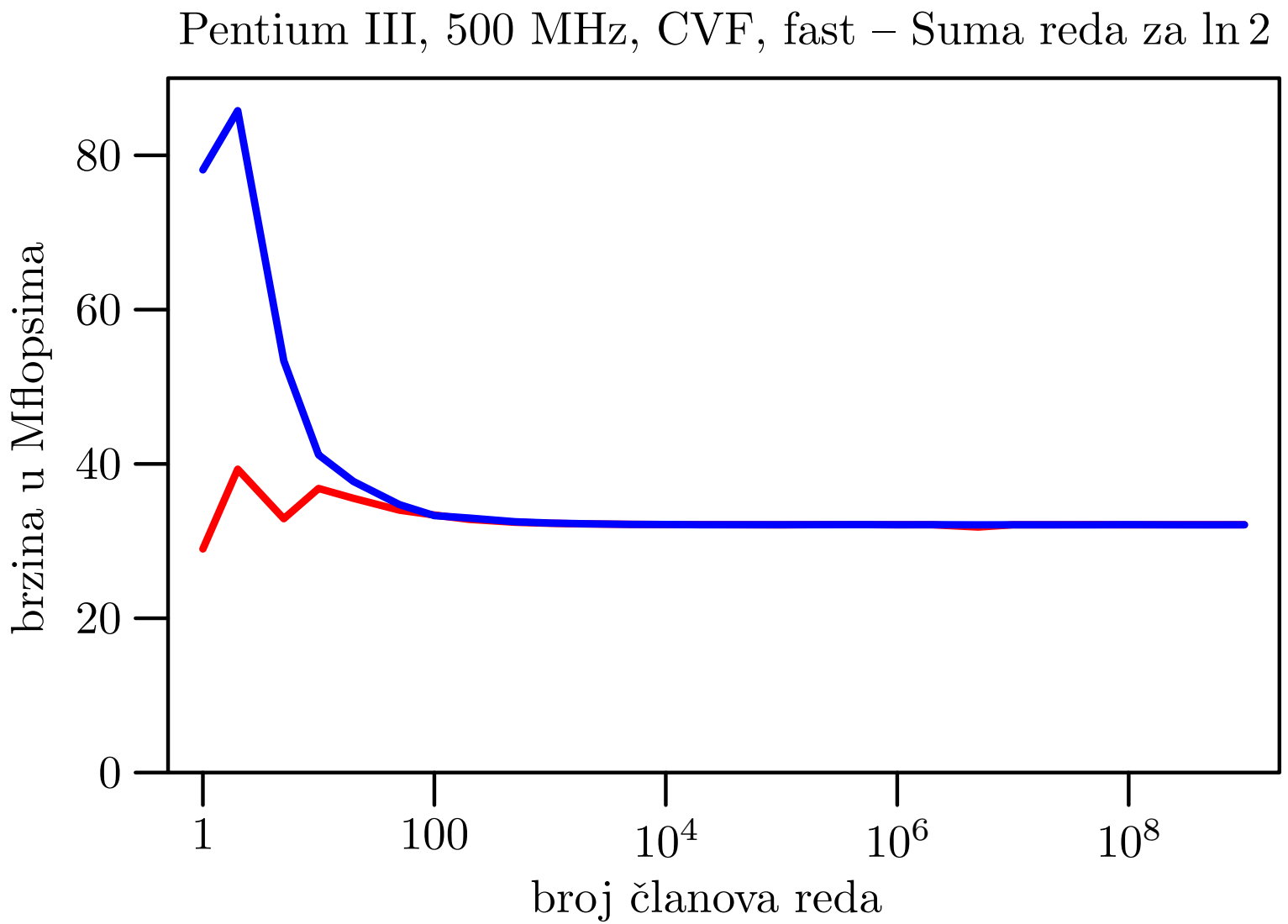
Legenda za čitanje grafova:

- zbrajanje unaprijed — plavo, brže,
- zbrajanje unazad — crveno, sporije.

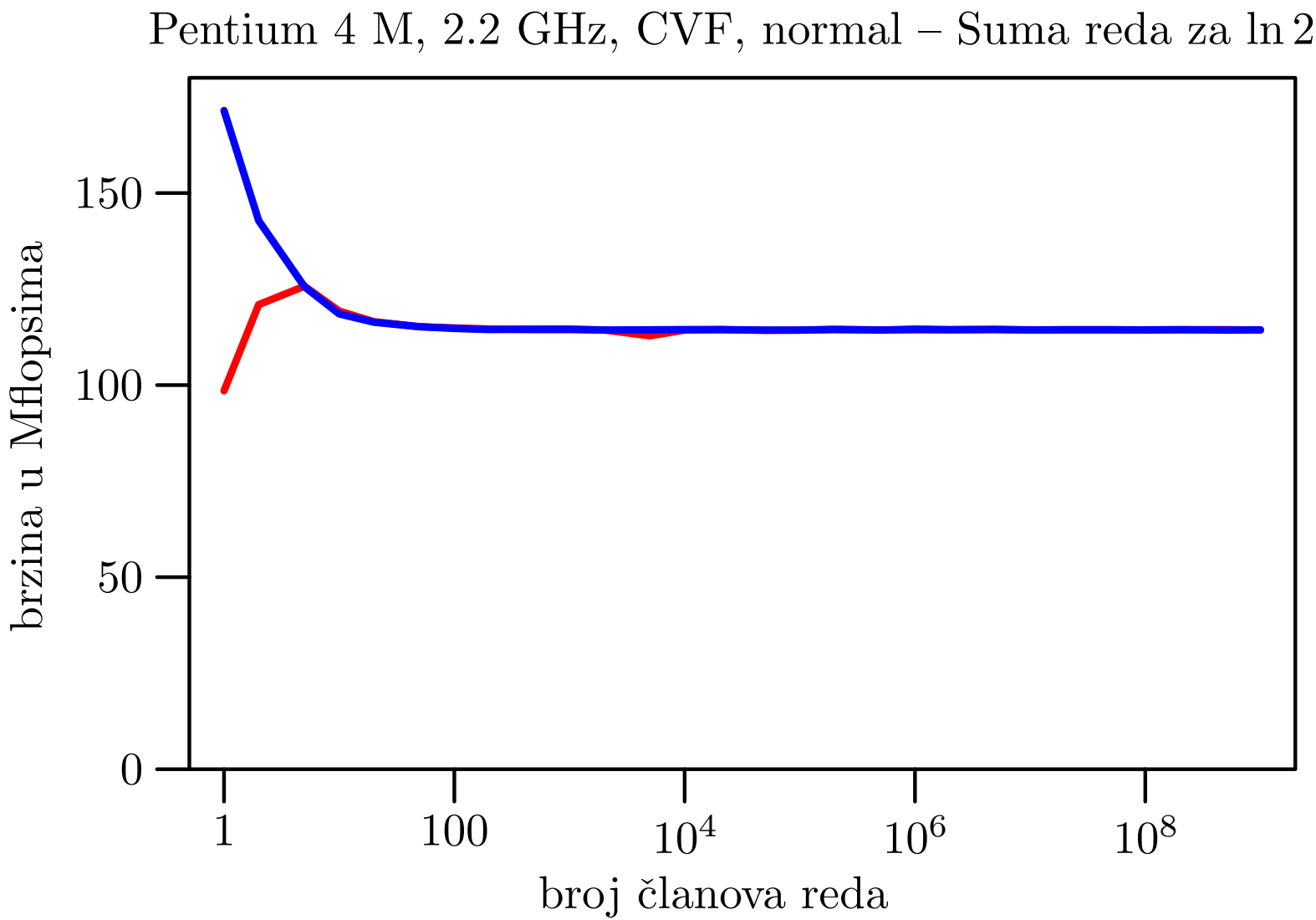
Klamath5, CVF, normal — unaprijed, unatrag



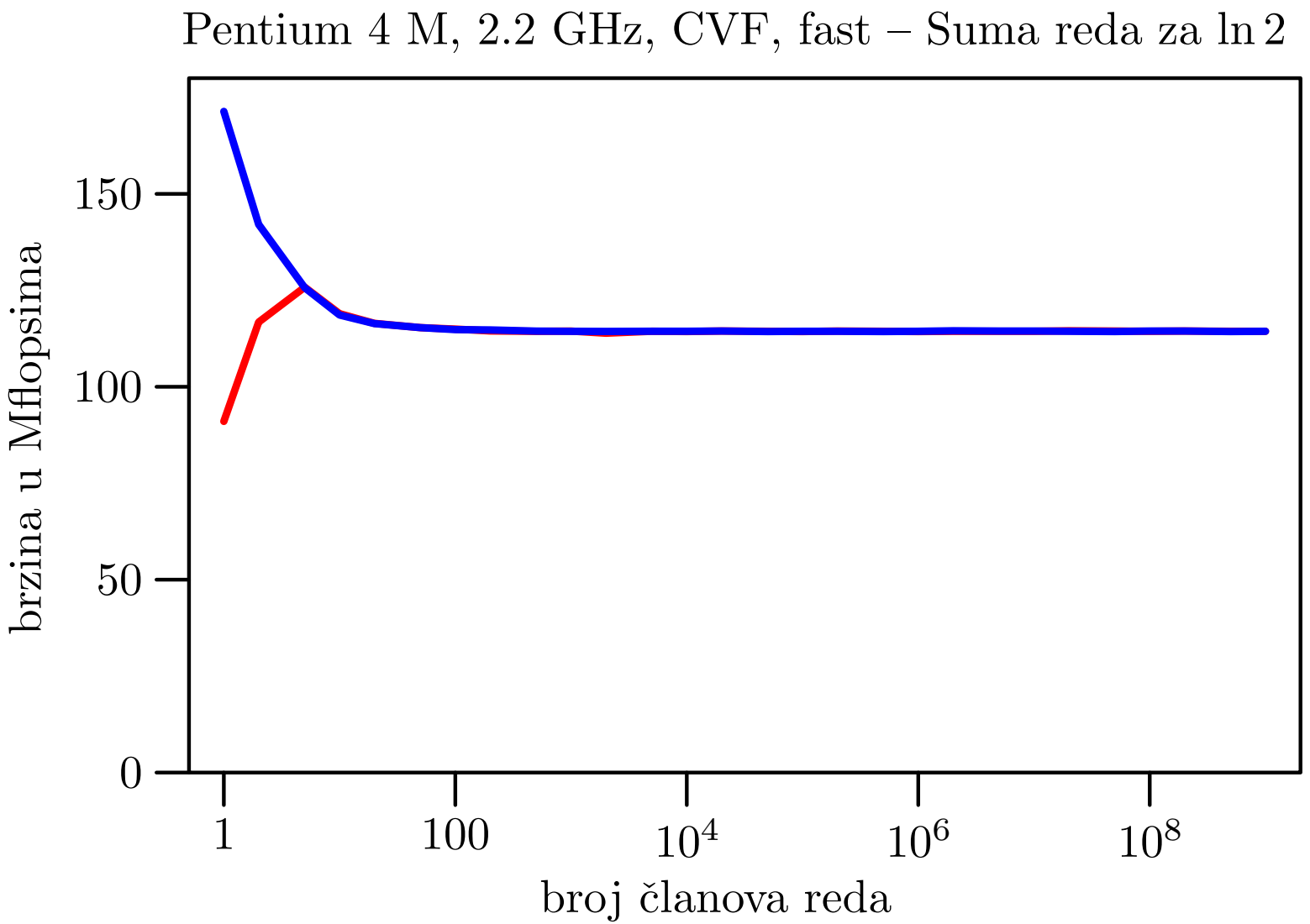
Klamath5, CVF, fast — unaprijed, unatrag



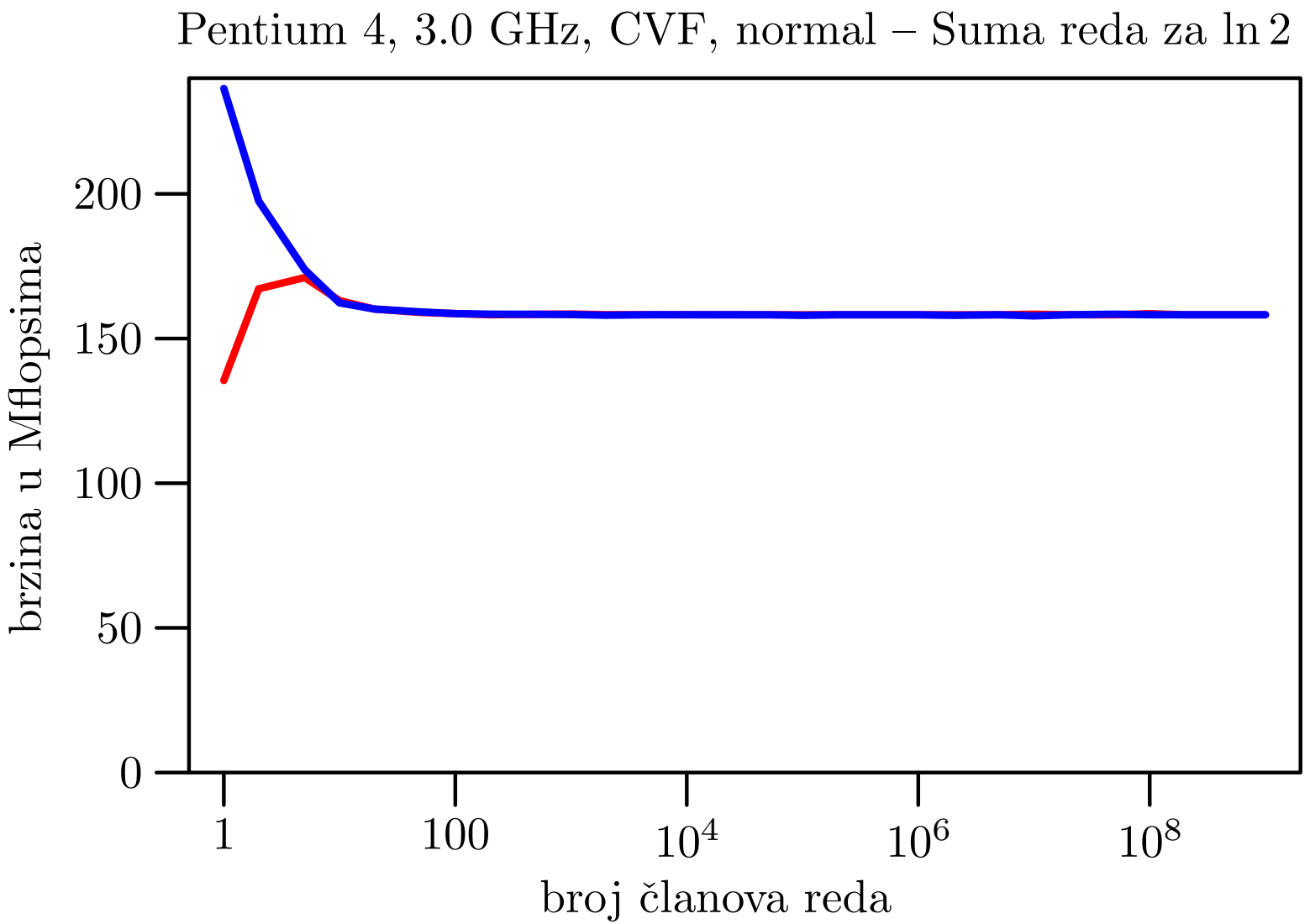
Notebook, CVF, normal — unaprijed, unatrag



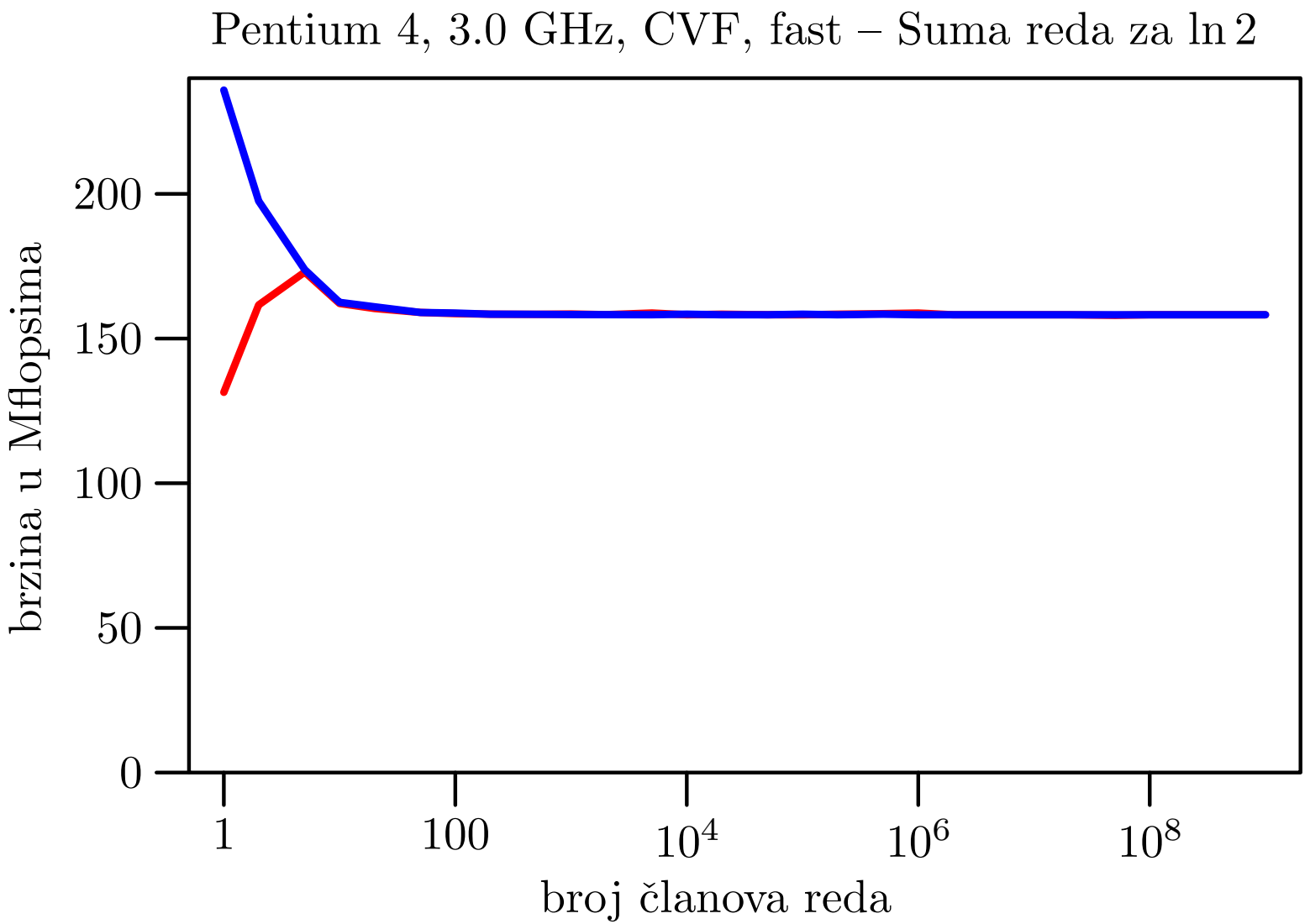
Notebook, CVF, fast — unaprijed, unatrag



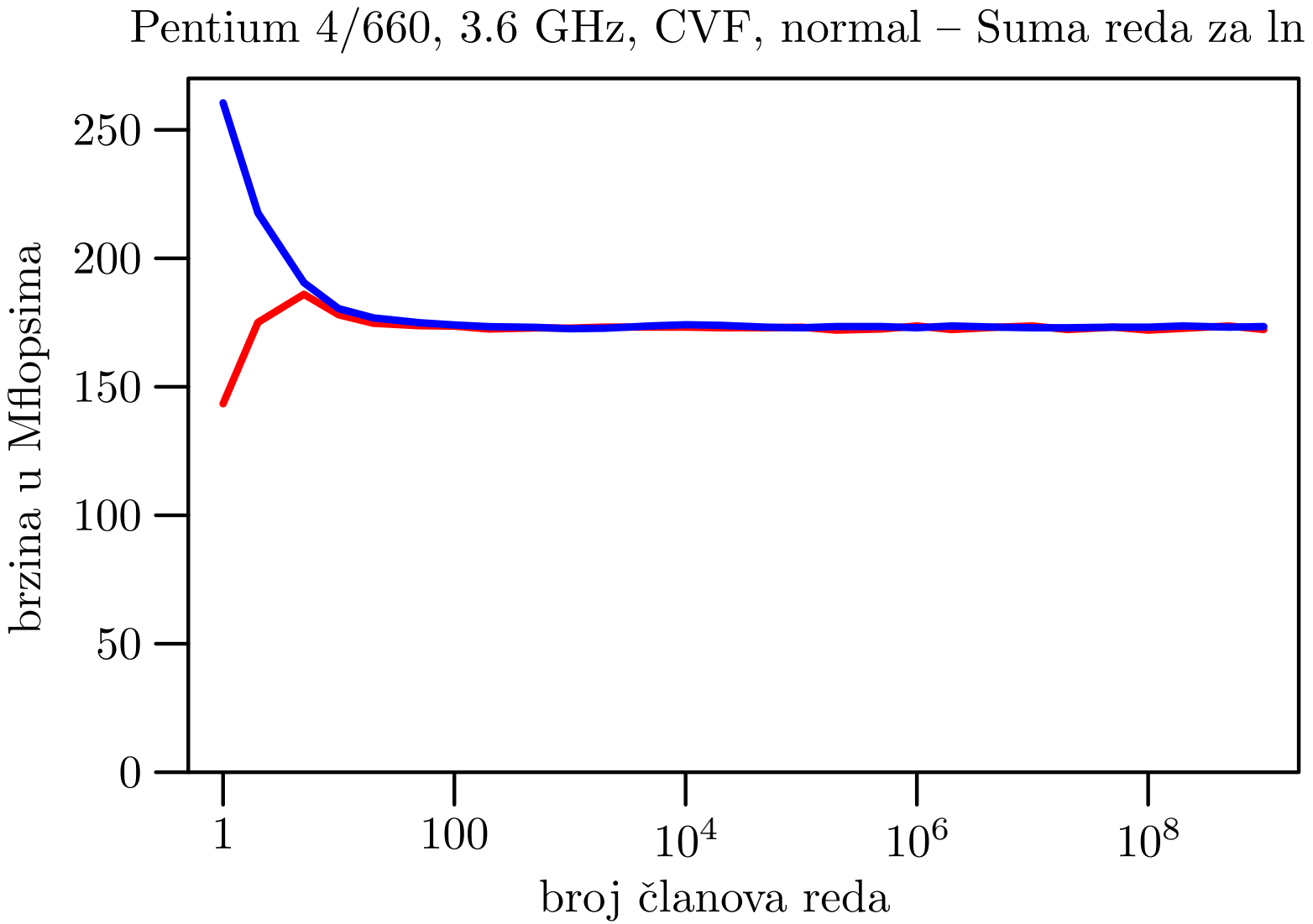
Veliki, CVF, normal — unaprijed, unatrag



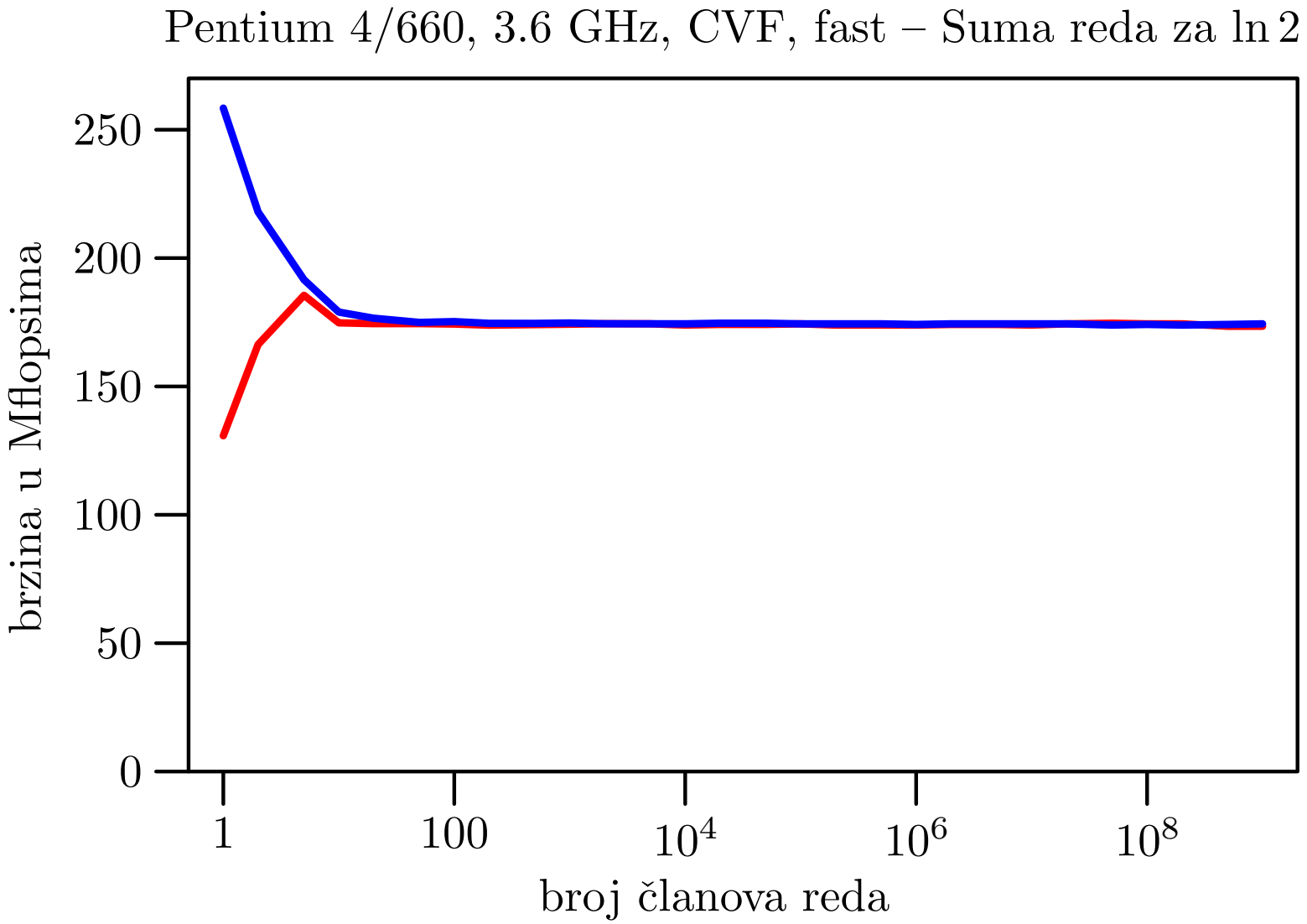
Veliki, CVF, fast — unaprijed, unatrag



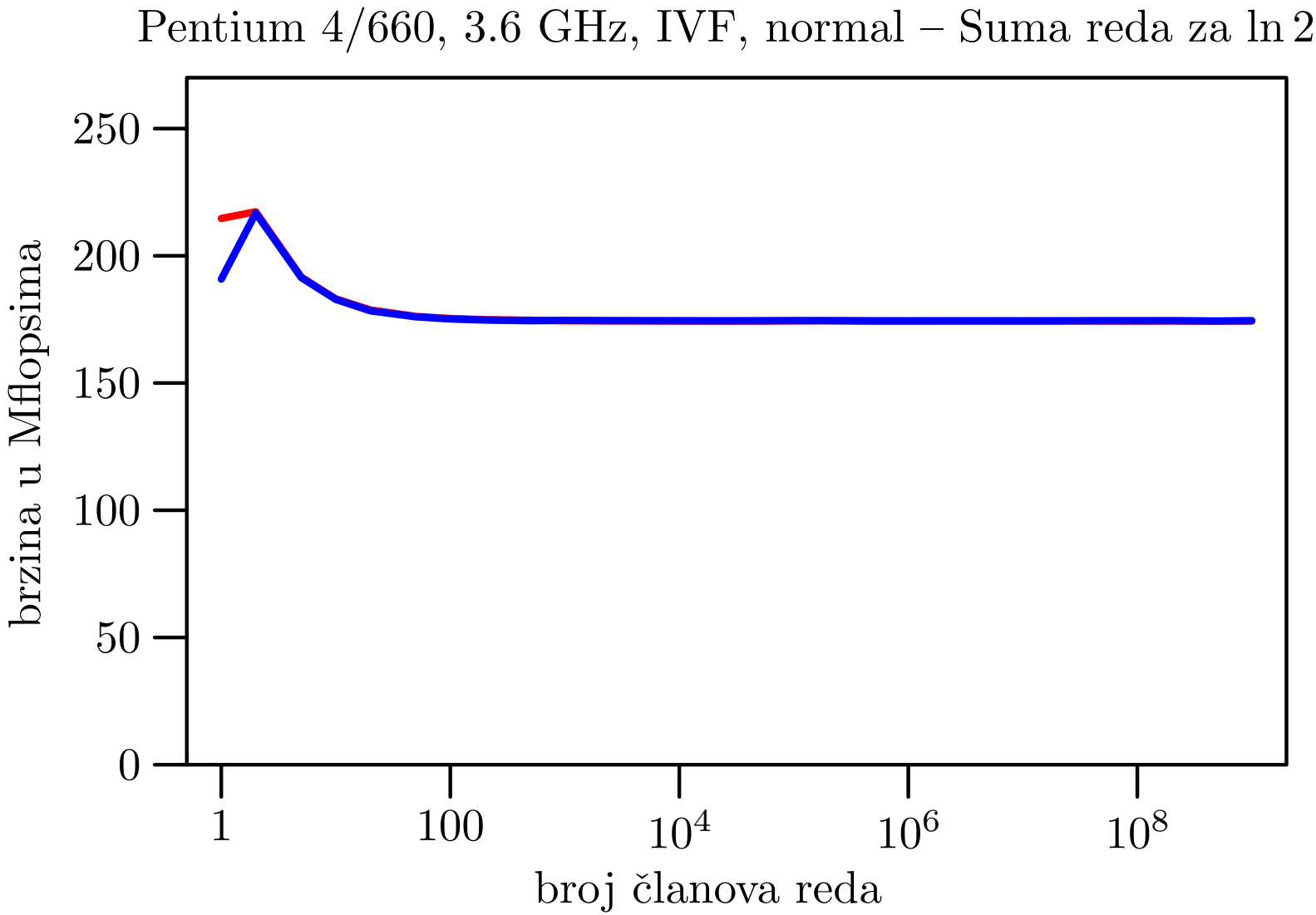
BabyBlue, CVF, normal — unaprijed, unatrag



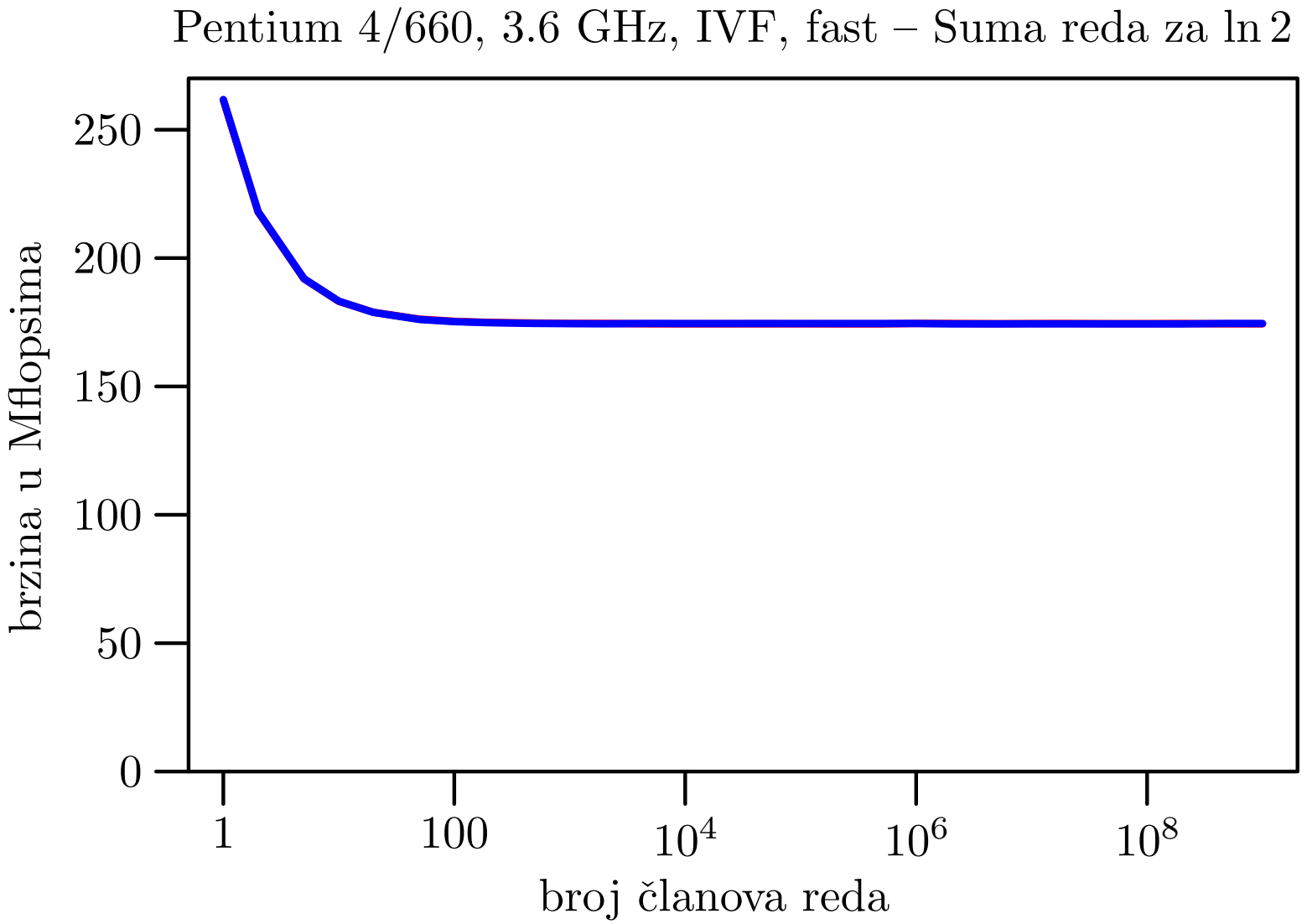
BabyBlue, CVF, fast — unaprijed, unatrag



BabyBlue, IVF, normal — unaprijed, unatrag



BabyBlue, IVF, fast — unaprijed, unatrag



Tablica brzina za velike n

Vidimo da nema neke razlike u brzinama između:

- zbrajanja unaprijed i unazad,
- fast i normal opcija compilera.

Usporedba brzina (u Mflops) za razna računala:

Računalo	Brzina
Klamath5	32.1
Notebook	114.4
Veliki	158.2
BabyBlue	174.4

Brzina raste sporije od frekvencije!

Komentar rezultata

Podatke o brzinama treba gledati samo relativno (za usporedbu), a ne apsolutno.

Naš model složenosti (brojimo floating–point operacije)

$$F(n) = 2n + 1$$

je poprilično neprecizan. Zato dobijemo "čudno" ponašanje grafova brzine za male n.

Izmjerena vremena imaju puno "pitomije" ponašanje.

- Trajanje vanjske petlje s ponavljanjem je praktički konstantno, i dobivamo "razuman" broj u sekundama,

što znači da smo dobro izabrali broj ponavljanja $N(n)$.

BabyBlue, IVF, fast — Izračunate brzine

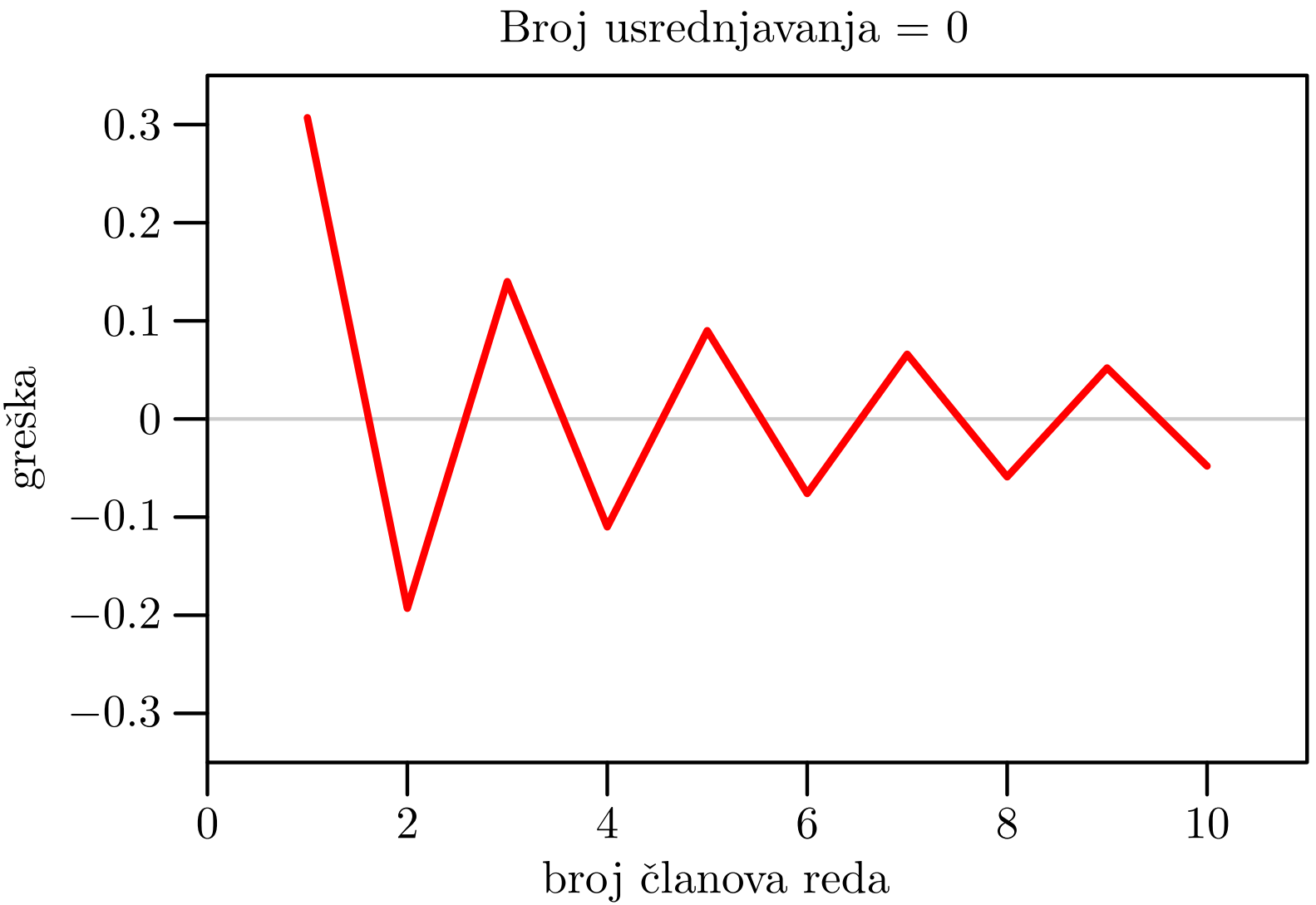
n =	1	264.488	264.340
n =	2	220.347	220.321
n =	5	193.859	193.873
n =	10	185.037	185.059
n =	20	173.545	173.563
n =	50	175.152	175.132
...			
n =	10000000	176.242	176.241
n =	20000000	176.245	176.245
n =	50000000	176.242	176.243
n =	100000000	176.228	176.247
n =	200000000	176.245	176.236
n =	500000000	176.249	176.235
n =	1000000000	176.246	176.249

BabyBlue, IVF, fast — Izmjerena vremena

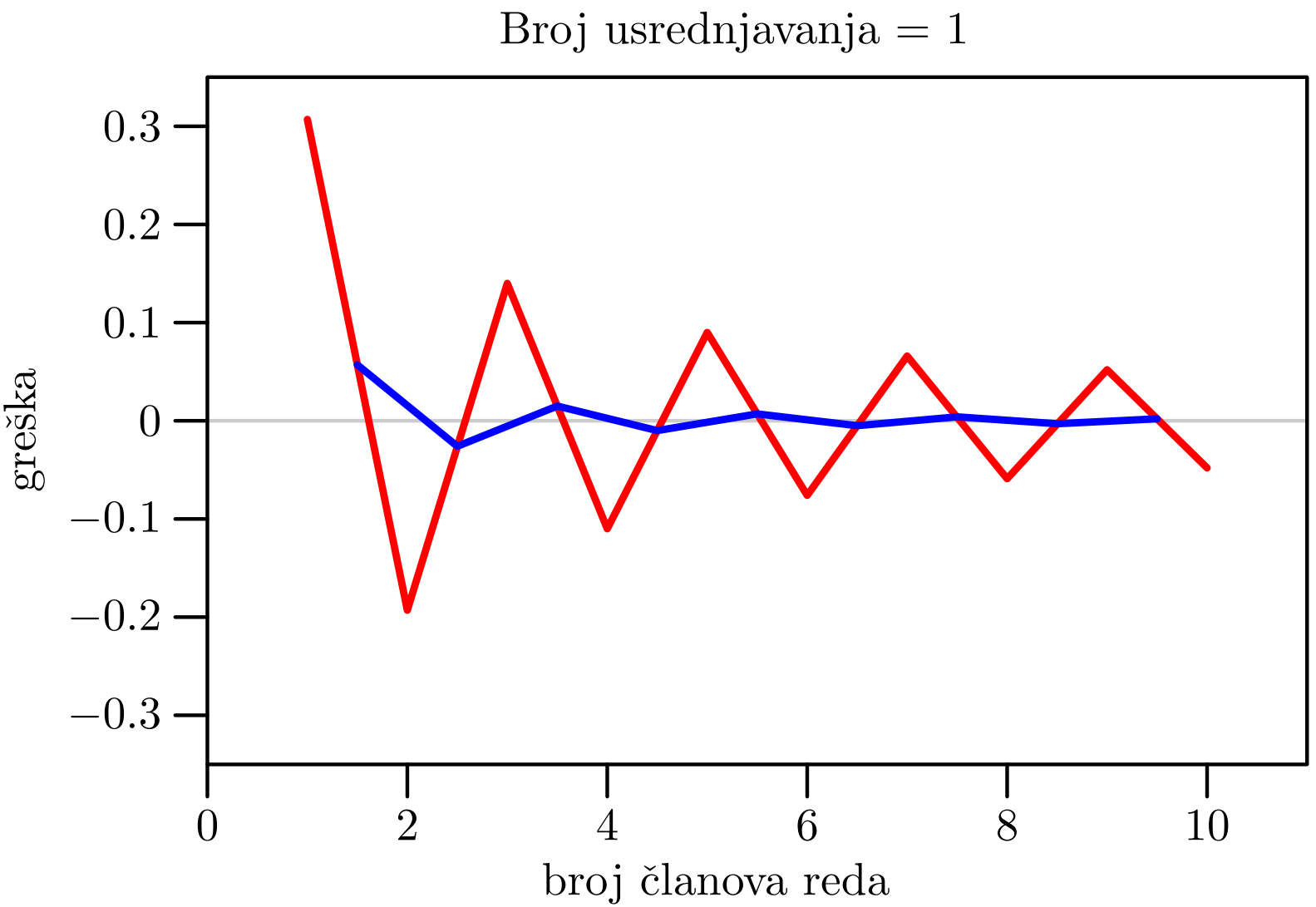
n = 1	11.343	11.349
n = 2	11.346	11.347
n = 5	11.348	11.348
n = 10	11.349	11.348
n = 20	11.813	11.811
n = 50	11.533	11.534
...		
n = 10000000	11.348	11.348
n = 20000000	11.348	11.348
n = 50000000	11.348	11.348
n = 100000000	11.349	11.348
n = 200000000	11.348	11.348
n = 500000000	11.348	11.348
n = 1000000000	11.348	11.348

Parcijalne sume reda
Ubrzanje algoritma

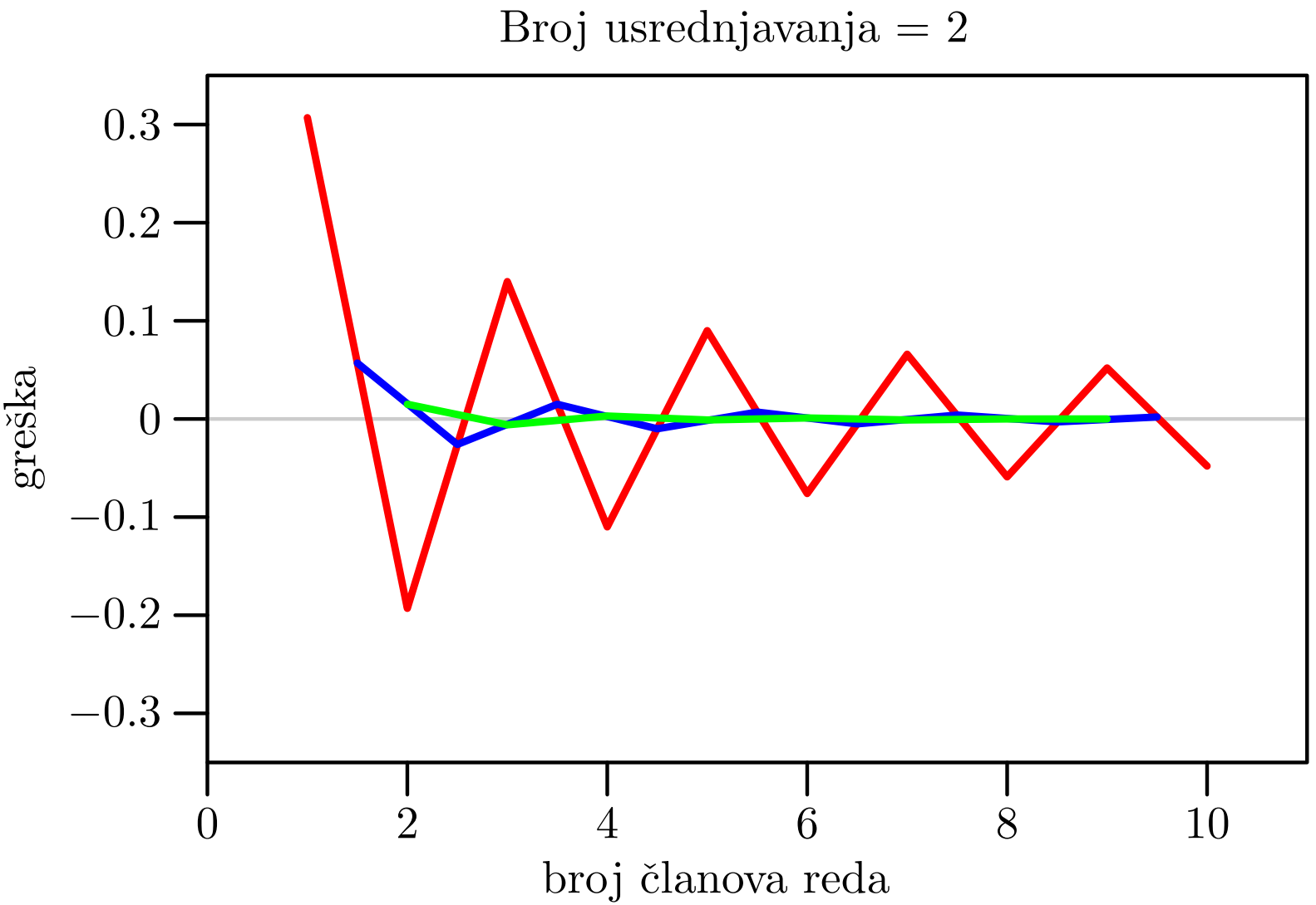
Greške nakon 0 usrednjavanja



Greške nakon 1 usrednjavanja



Greške nakon 2 usrednjavanja



Tablica grešaka nakon 9 usrednjavanja

Vrijednosti grešaka $S(n,k) - S$ za prvih 10 članova reda:

k	1	2	3	4	5	6	7	8	9	10
0	0.307	−0.193	0.140	−0.110	0.090	−0.076	0.066	−0.059	0.052	−0.048
1	0.057	−0.026	0.015	−0.010	0.007	−0.005	0.004	−0.003	0.002	
2	0.015	−0.006	0.003	−0.001	0.001	−0.001	0.000	−0.000		
3	0.005	−0.001	0.001	−0.000	0.000	−0.000	0.000			
4	0.002	−0.000	0.000	−0.000	0.000	−0.000				
5	0.001	−0.000	0.000	−0.000	0.000					
6	0.000	−0.000	0.000	−0.000						
7	0.000	−0.000	0.000							
8	0.000	−0.000								
9	0.000									