

Složenost algoritama

7. predavanje

Saša Singer

`singer@math.hr`

`web.math.hr/~singer`

PMF – Matematički odjel, Zagreb

Zbrajanje matrica

Zbrajanje matrica

Problem: Zadan je prirodni broj $n \in \mathbb{N}$ i 3 realne matrice A , B i C , reda n . Treba izračunati izraz

$$C := C + A + B.$$

Akumulacija (“nazbrajavanje”) zbroja $A + B$ u matrici C ovdje ima samo jednu svrhu:

🔴 “prevariti” optimizaciju kompilera, kod višestrukog ponavljanja eksperimenta.

Ova realizacija napravljena je po ugledu na množenje matrica (v. kasnije).

Zbrajanje matrica — formula

“Matematička” realizacija **matrične** operacije

$$C := C + A + B$$

po **elementima** je trivijalna:

$$c_{ij} := c_{ij} + a_{ij} + b_{ij},$$

za sve indekse

$$i = 1, \dots, n, \quad j = 1, \dots, n.$$

Dakle, “programski” — treba “zavrtiti” **dvije** petlje.

Zbrajanje matrica — potprogram

```
subroutine addij (lda, n, a, b, c)
c
c Matrix addition
c  $C(n, n) = C(n, n) + A(n, n) + B(n, n)$ .
c
c   implicit none
c
c   integer lda, n
c   double precision a(lda, lda), b(lda, lda),
c   $               c(lda, lda)
c
c   integer i, j, nn
```

Zbrajanje matrica — potprogram (nastavak)

```
c
c   IJ loop, inner
c
      nn = n
      do 20, i = 1, nn
        do 10, j = 1, nn
          c(i, j) = c(i, j) + a(i, j) + b(i, j)
10        continue
20      continue
c
      return
      end
```

Permutacija petlji

Prvu varijantu zovemo **ij** — po **poretku** (indeksa) petlji, **izvana** prema **unutra**.

Ove **dvije** petlje možemo **permutirati**, tj. zamijeniti im poredak, pa dobivamo **ji** varijantu:

```
c
c   JI loop, inner
c
      nn = n
      do 20, j = 1, nn
        do 10, i = 1, nn
          c(i, j) = c(i, j) + a(i, j) + b(i, j)
10          continue
20        continue
```

Broj operacija

U svakom prolazu kroz unutarnju petlju imamo dvije operacije
• zbrajanja matričnih elemenata.

Obje petlje imaju (svaka) točno n prolaza.

Dakle, ukupan broj operacija u obje varijante algoritma je:

$$F(n) = 2n^2.$$

Broj ponavljanja $N(n)$ izabran je tako da dobijemo približno konstantno trajanje “okolne” petlje (s ponavljanjem) kojoj mjerimo vrijeme.

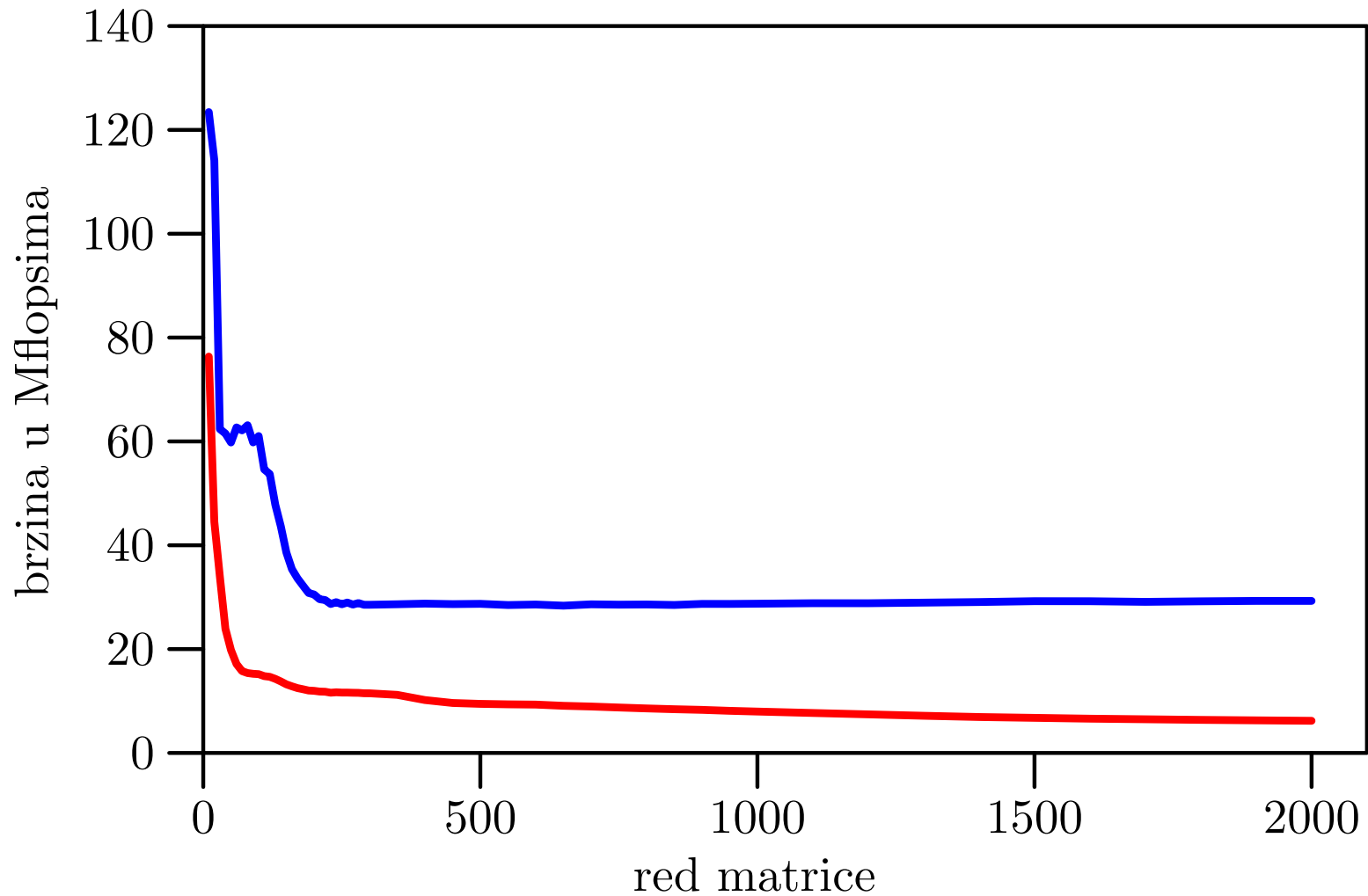
Boje na grafovima

Legenda za čitanje grafova:

- petlja **ij** — **crveno**, sporo;
- petlja **ji** — **plavo**, brzo.

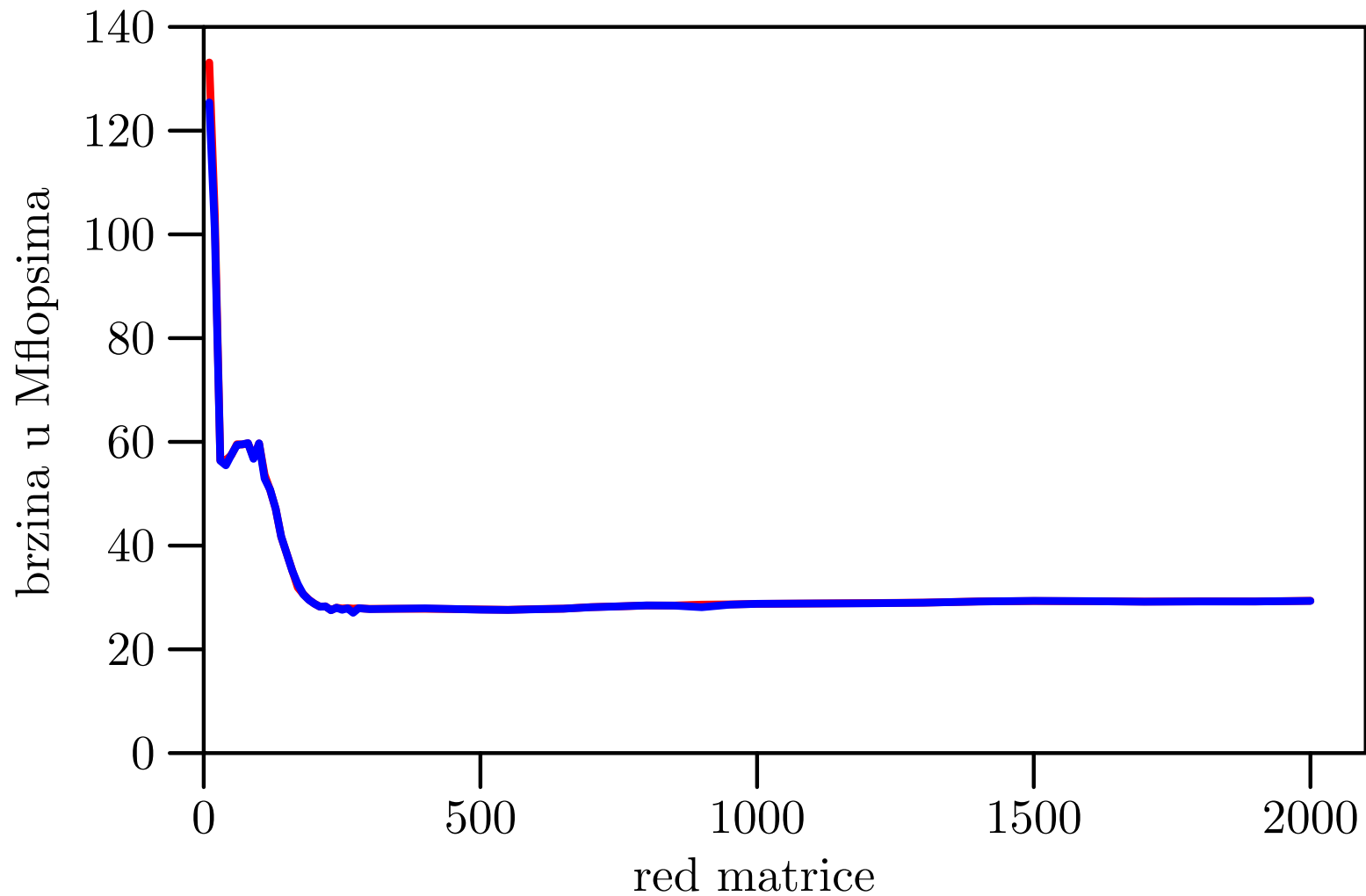
Klamath5, CVF, normal — ij , ji

Pentium III, 500 MHz, CVF, normal – Zbrajanje matrica



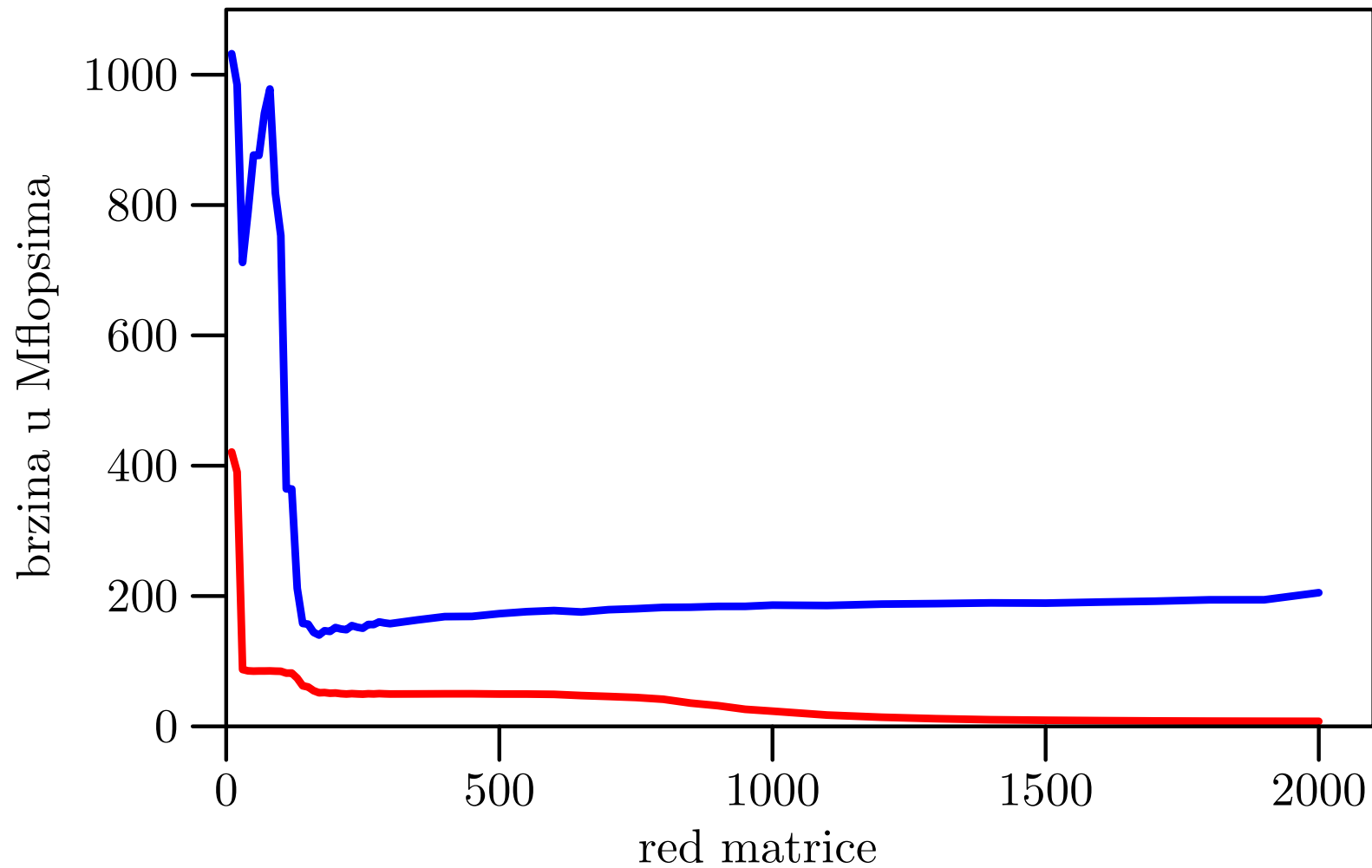
Klamath5, CVF, fast — ij , ji

Pentium III, 500 MHz, CVF, fast – Zbrajanje matrica



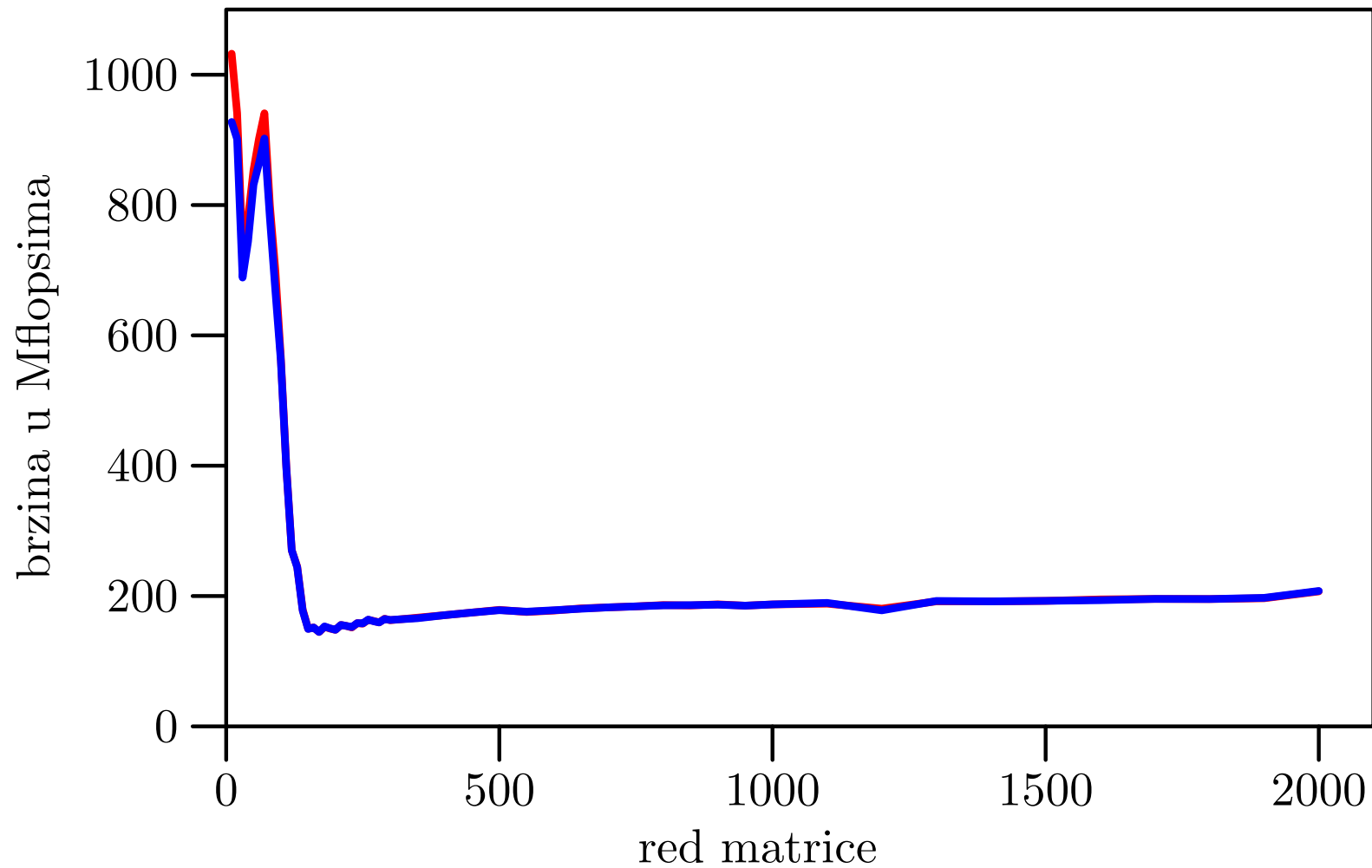
Veliki, CVF, normal — ij , ji

Pentium 4, 3.0 GHz, CVF, normal – Zbrajanje matrica



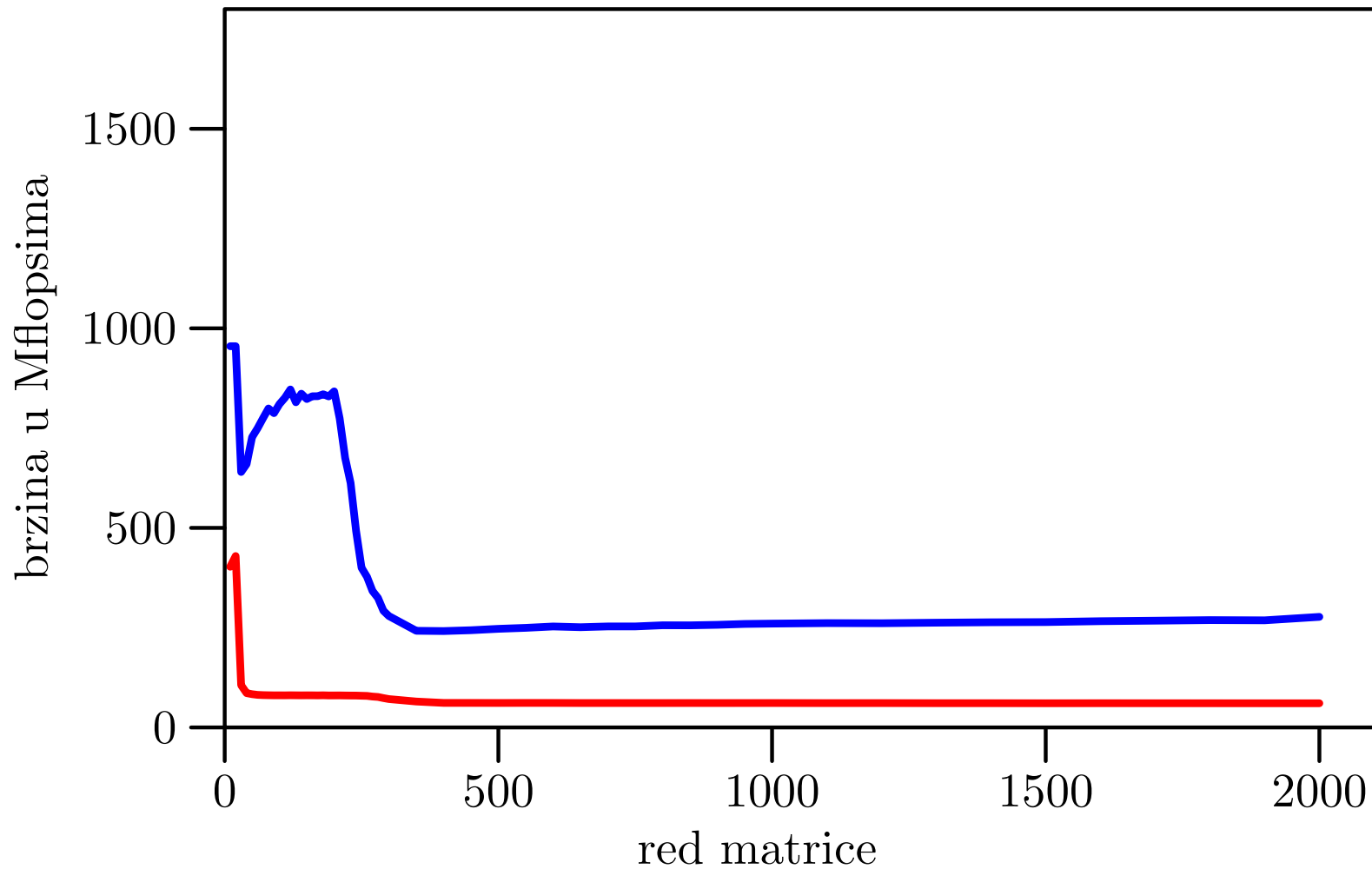
Veliki, CVF, fast — ij , ji

Pentium 4, 3.0 GHz, CVF, fast – Zbrajanje matrica



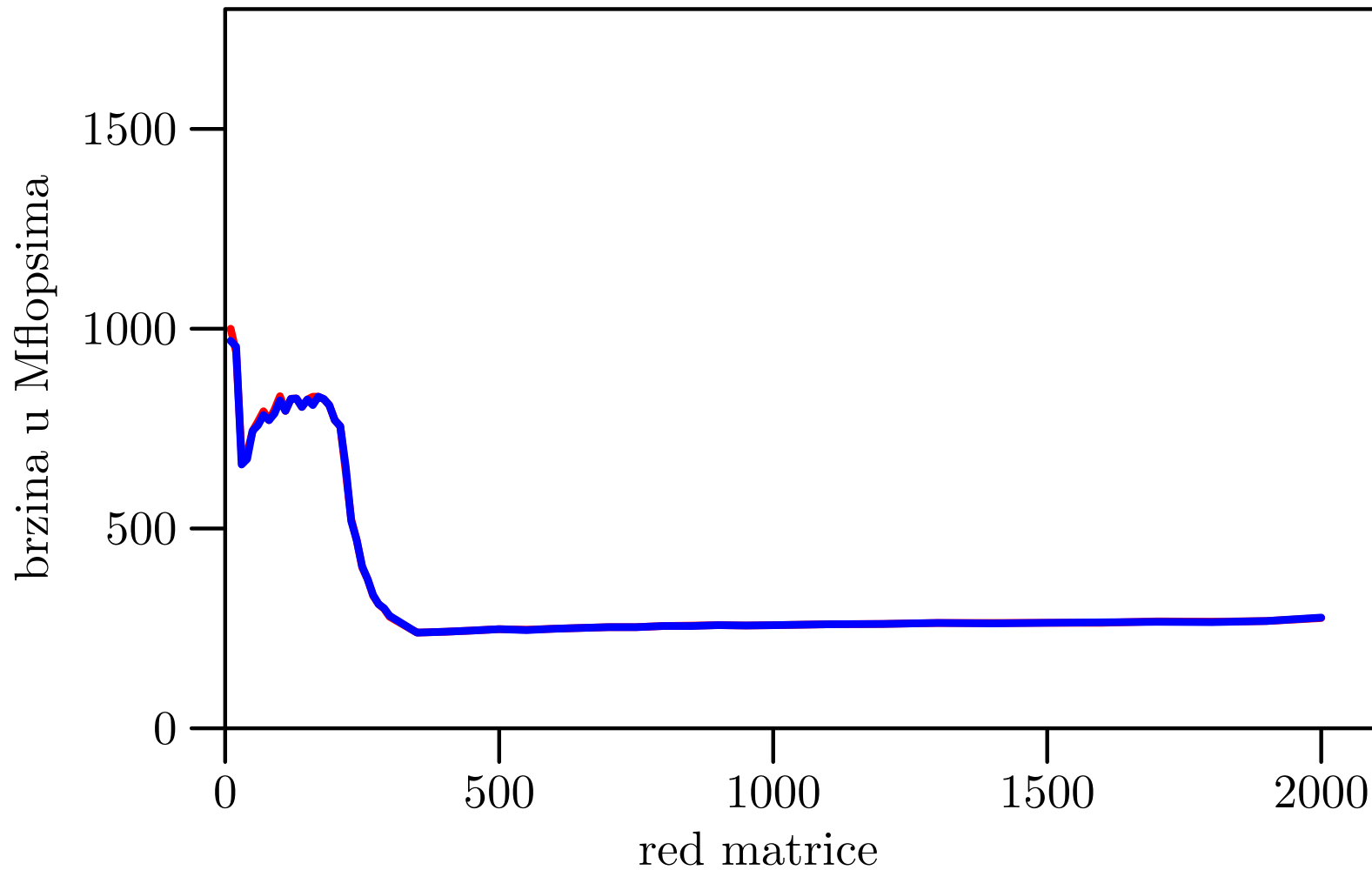
BabyBlue, CVF, normal — ij , ji

Pentium 4/660, 3.6 GHz, CVF, normal – Zbrajanje matrica



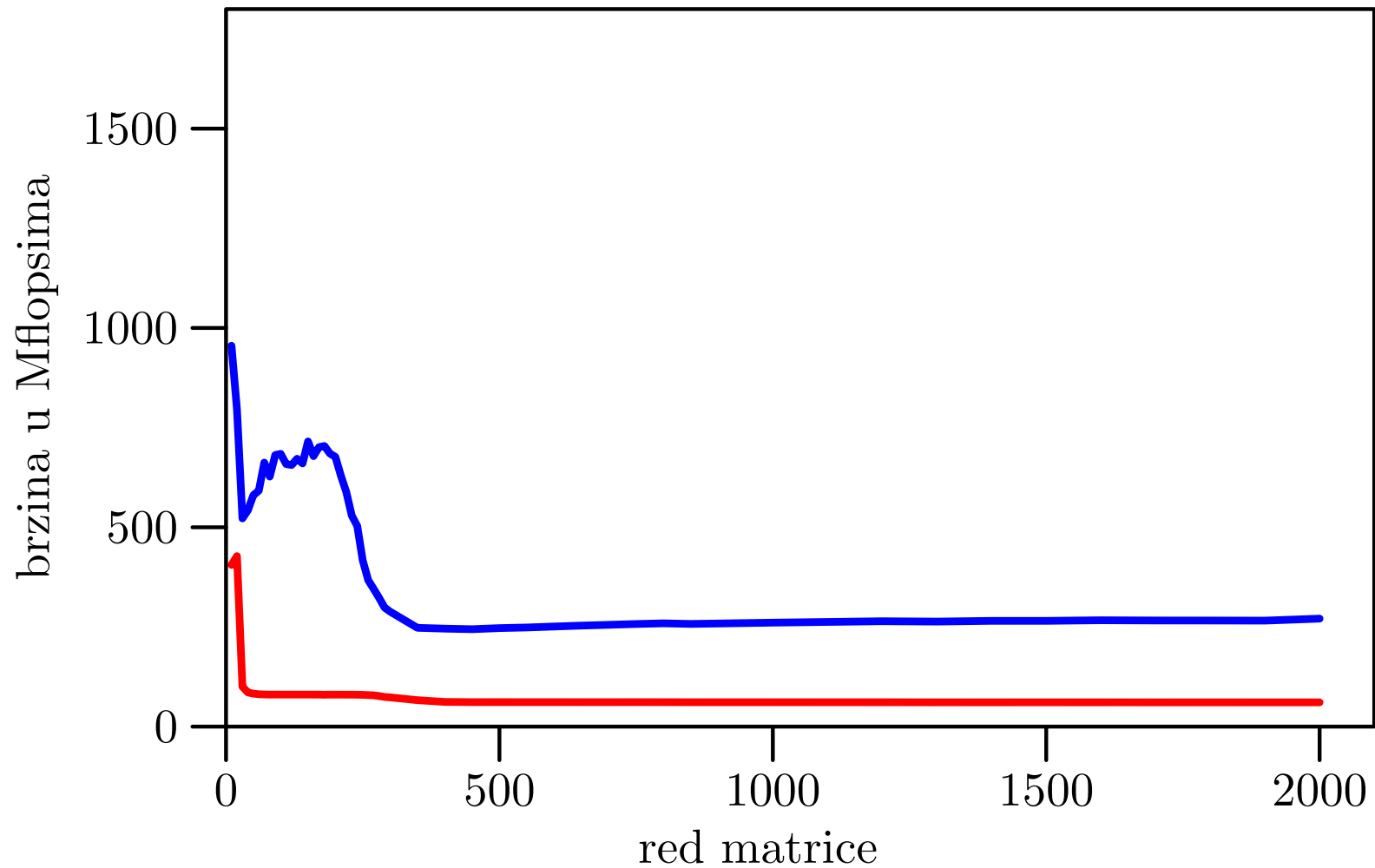
BabyBlue, CVF, fast — ij , ji

Pentium 4/660, 3.6 GHz, CVF, fast – Zbrajanje matrica



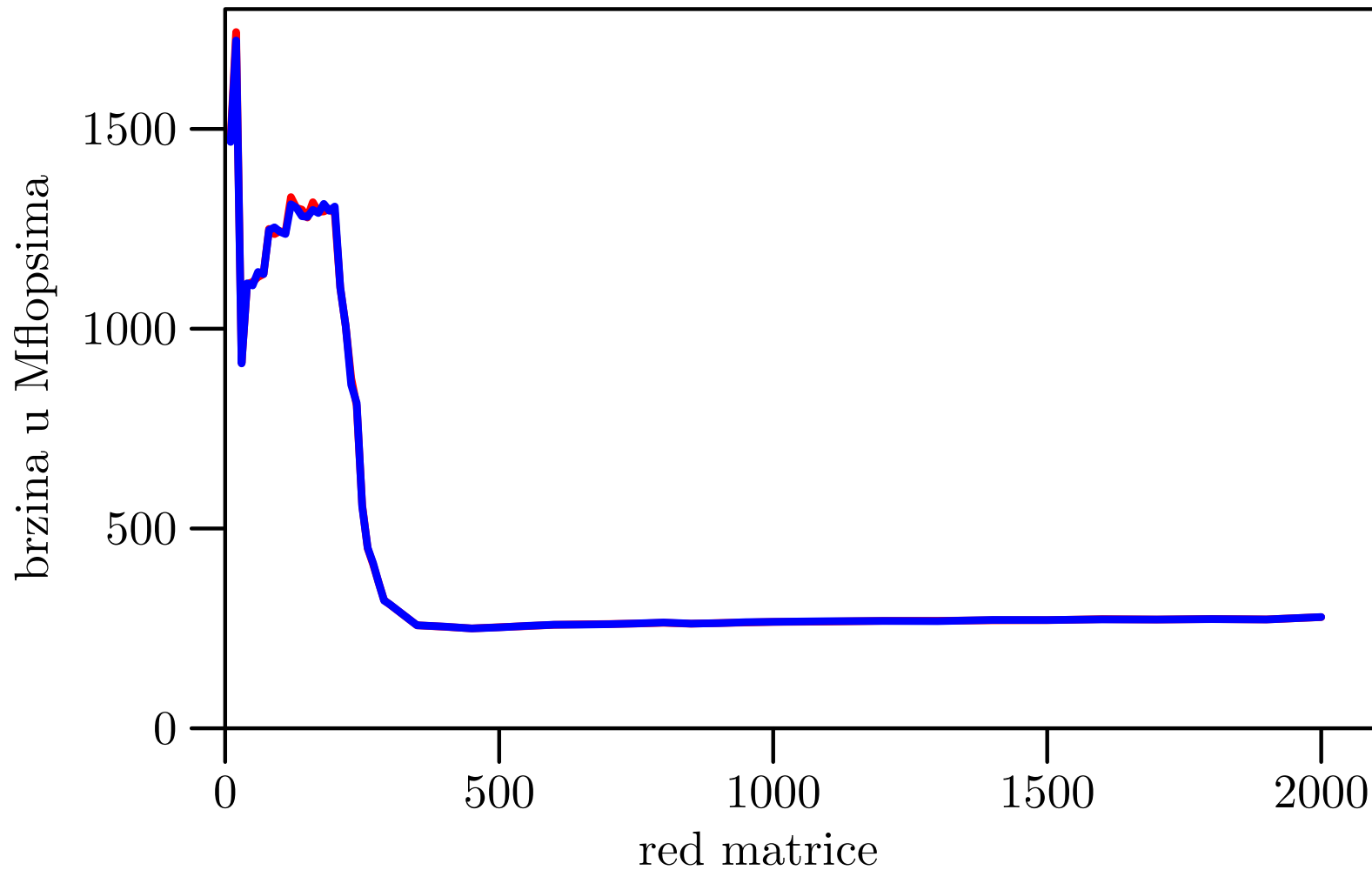
BabyBlue, IVF, normal — ij , ji

Pentium 4/660, 3.6 GHz, IVF, normal – Zbrajanje matrica



BabyBlue, IVF, fast — ij , ji

Pentium 4/660, 3.6 GHz, IVF, fast – Zbrajanje matrica



Tablica brzina za velike n

Razlika u brzini između brže ji petlje i sporije ij petlje je ogromna, a nema neke razlike u brzinama između:

- `fast` i `normal` opcija kompilera za bržu ji petlju.

Usporedba brzina (u Mflops) sporije i brže petlje za razna računala:

Računalo	ij petlja	ji petlja
Klamath5	6.2	29.3
Veliki	7.7	205.1
BabyBlue, CVF	60.5	277.1
BabyBlue, IVF	60.8	270.8

Komentar rezultata — brža i sporija petlja

Opazanje 1. Bez optimizacije dobivamo

● ogromnu razliku u brzini između brže i sporije petlje, za bilo koje redove n .

(O razlici između malih i velikih n — malo kasnije.)

Razlog: Brzina je bitno veća kad

● podacima pristupamo redom, kako su spremljeni u memoriji

(Računanje adresa, “blok”–transfer podataka.)

Kako se spremaju matrice u pojedinim programskim jezicima?

Komentar rezultata — brže/sporije (nastavak)

FORTRAN: matrice se spremaju **po stupcima**, tj.

● “brže” se mijenja **prvi** indeks **i** (za retke).

Za **sekvencijalni** pristup podacima

● indeks **stupca j** mora biti **izvana**, a indeks **retka i** **unutra**.

Zato je **ji** petlja brža od **ij** petlje!

C i Pascal: matrice se spremaju **po recima**, tj.

● “brže” se mijenja **drugi** indeks **j** (za stupce).

Tamo je **obratno** — **ij** petlja je brža od **ji** petlje!

Komentar rezultata — “mali” i “veliki” n

Opažanje 2. Za **male** redove n dobivamo

- bitno **veće** brzine, u usporedbi s onima za **velike** n , i to bez obzira na optimizaciju.

Razlog: “Krivac” za ovo **povećanje** brzine je **cache** memorija.

Međutim, to povećanje brzine

- ne pripada** problemu **zbrajanja** matrica, jer svaki ulazni podatak koristimo **samo jednom**,
- već dolazi **samo** od višestrukog **ponavljanja** eksperimenta (pa matrice ostaju u cacheu).

Posljedica: brzinu za **velike** n **ne možemo** “popraviti” promjenom algoritma.

Stvarni “izgled” računala
ili
“Priča o cacheu”

Sadržaj

- Stvarni “izgled” računala:
 - registri modernog procesora (IA-32),
 - primjer matične ploče, blok-dijagram,
 - hijerarhijska struktura memorije (cache).

Standardni kućni procesori

Standardni **kućni** procesori bazirani su na tzv. **IA-32** arhitekturi (Intel ili AMD, svejedno mi je). Osnovna svojstva:

- **riječ** = 32 bita = 4 B, (toliki je tip **int** u C-u),
- **adresa** = 32 bita (x86) ili, modernije, 64 bita (x64).

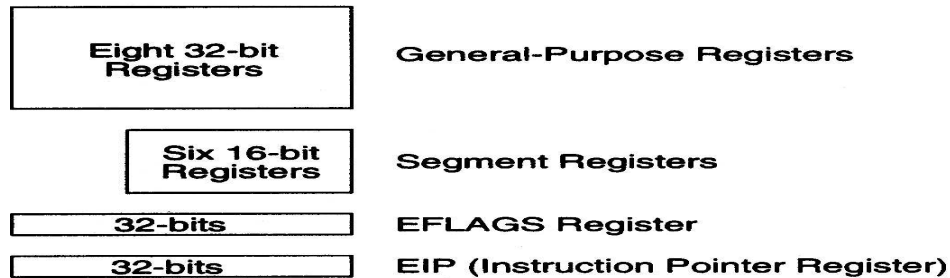
Ovi procesori imaju **gomilu registara**, raznih namjena, koji sadrže razne vrste podataka i instrukcija (ili dijelova instrukcija).

Shematski izgled **svih registara**, a onda samo **registara opće namjene** dan je na sljedeće dvije stranice.

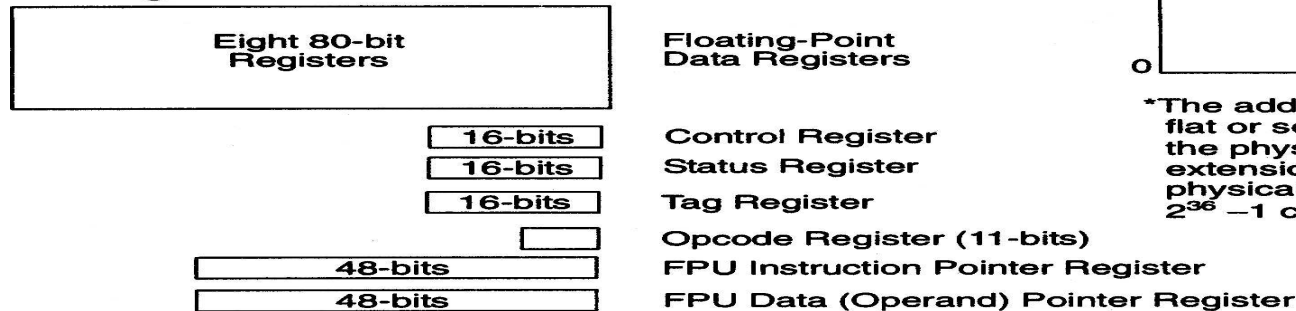
Napomena: slike odgovaraju IA-32 procesoru **Pentium 4**, serija Northwood, podnožje 478 (danas već zastarjelom).

IA-32 — Svi registri i adresni prostor

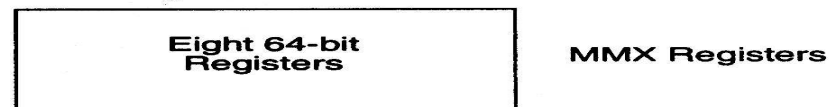
Basic Program Execution Registers



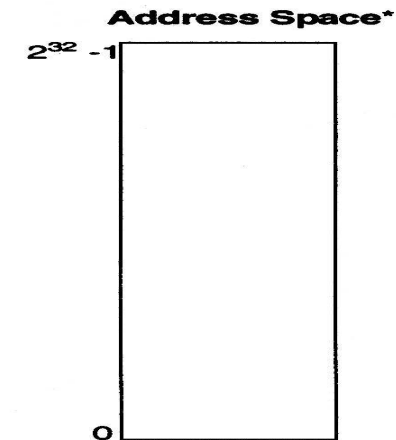
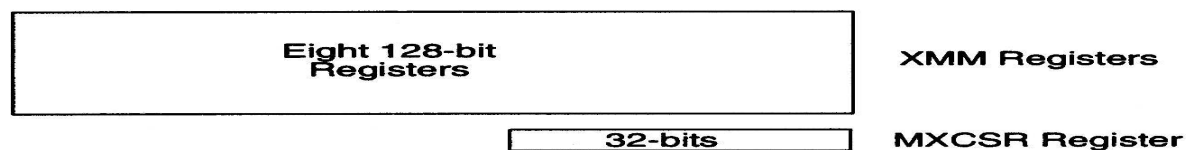
FPU Registers



MMX Registers

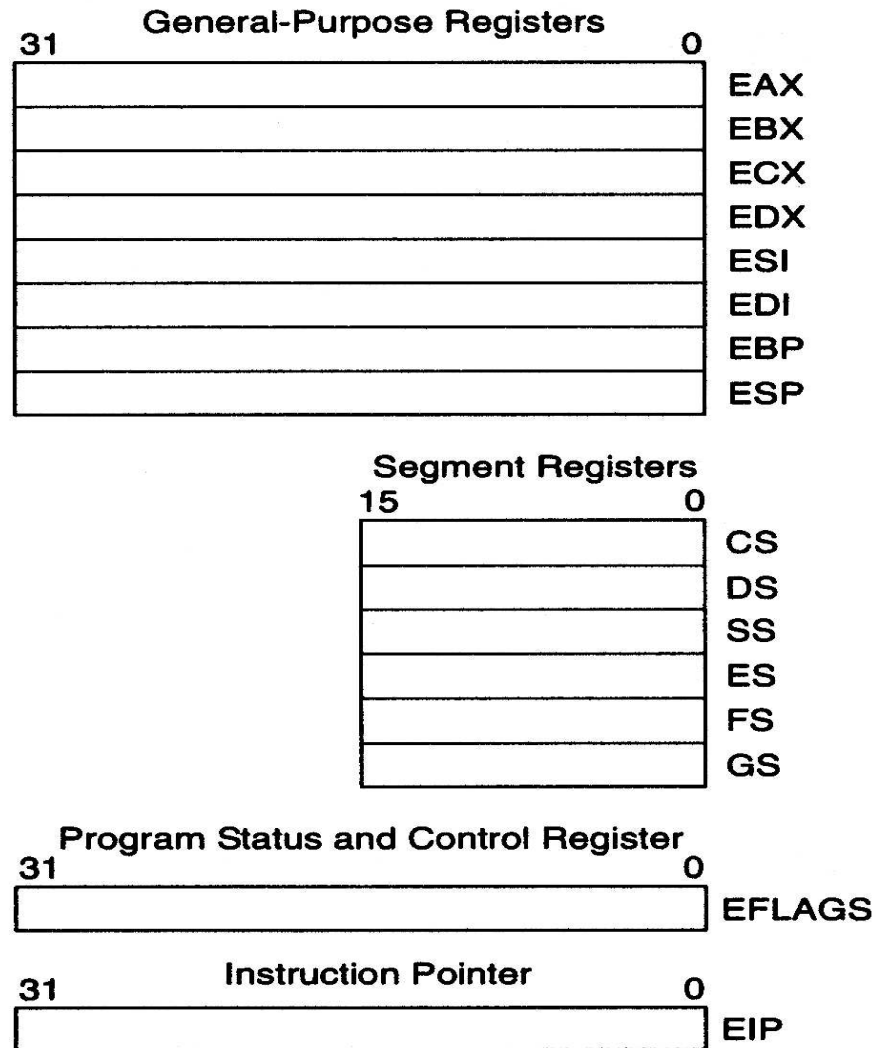


SSE and SSE2 Registers



*The address space can be flat or segmented. Using the physical address extension mechanism, a physical address space of 2³⁶ - 1 can be addressed.

IA-32 — Osnovni izvršni registri



Izgled matične ploče računala

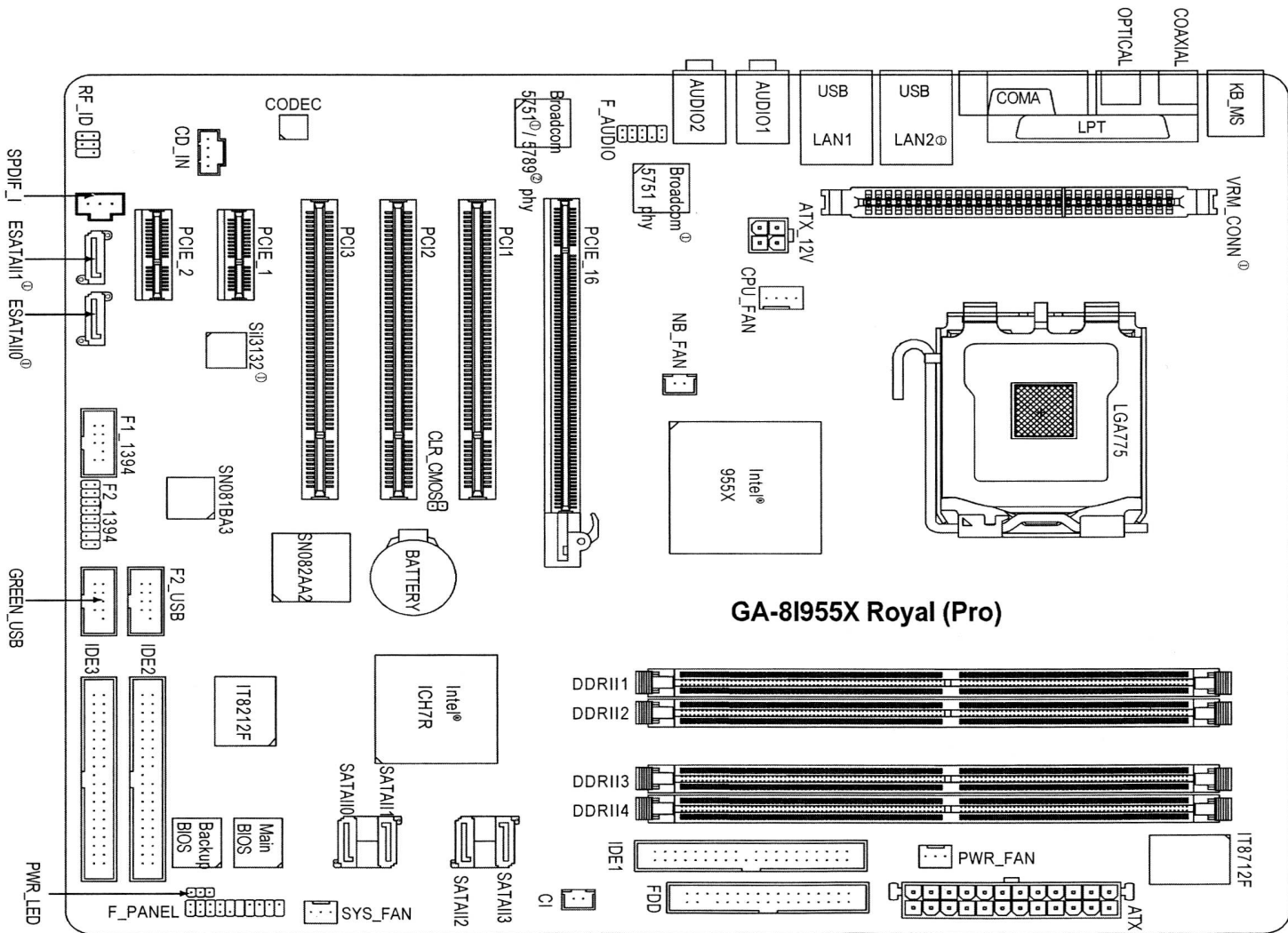
Moderna “kućna” računala, naravno, **imaju** sve standardne dijelove računala.

- Međutim, zbog “**multimedijalne**” namjene, ta računala imaju mogućnost priključivanja **velikog broja** raznih uređaja (“ulaz–izlaz”).
- Gomila toga je **već ugrađena** na modernim tzv. **matičnim pločama** (engl. **motherboard**).
- **Procesor** zauzima relativno “mali” dio površine (ili prostora), a najuočljiviji dio na njemu (nakon ugradnje) je **hladnjak**.
- Utori za **memorijske** “chipove”, također, ne zauzimaju previše prostora.

Matična ploča GA-8I955X Royal — izgled

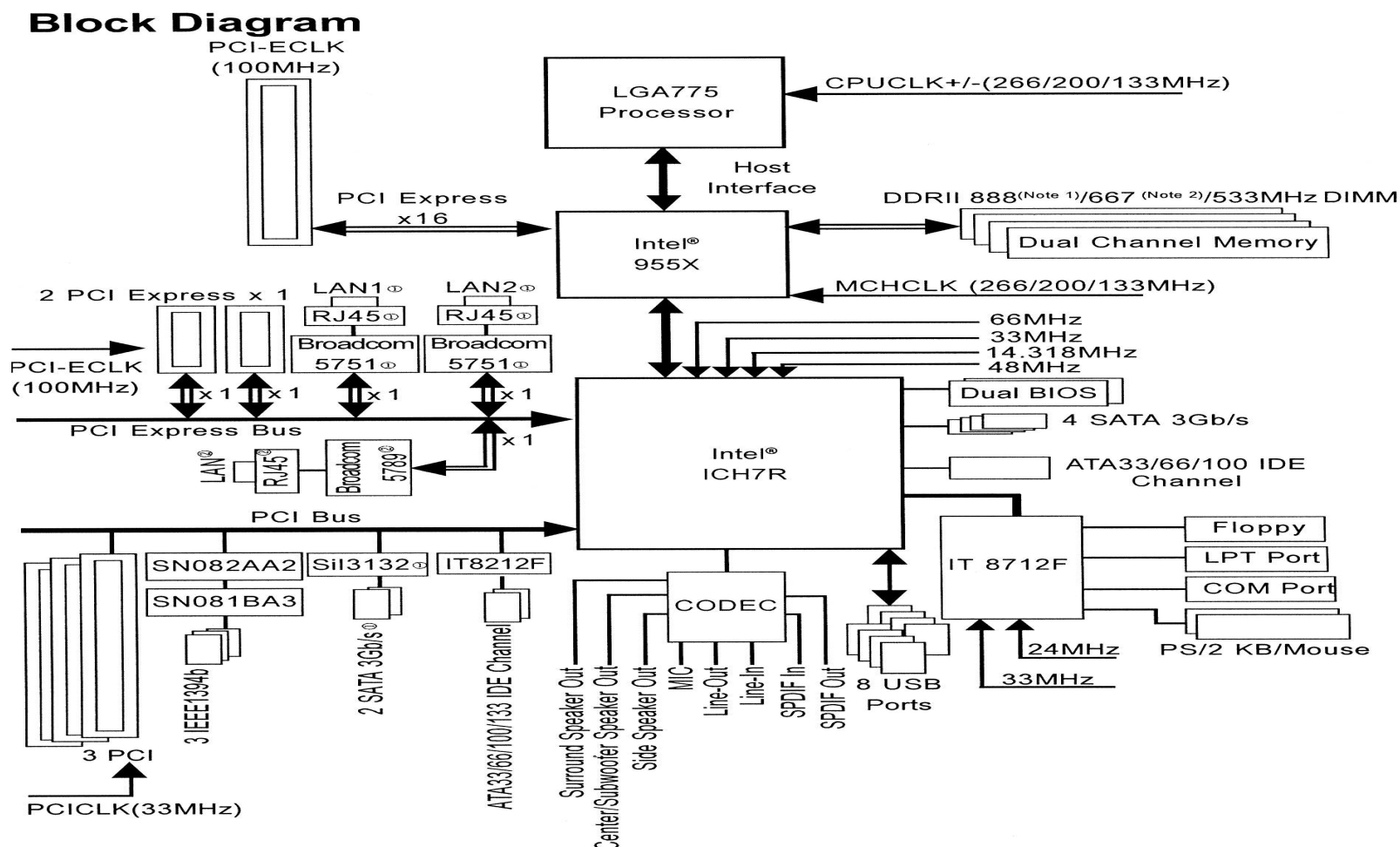


Matična ploča — raspored



GA-8I955X Royal/GA-8I955X Pro Motherboard Layout

Matična ploča — blok dijagram



(Note 1) DDR II memory can be overclocked to 888MHz (must be used with an 1066MHz FSB processor) through overclocking in BIOS. Go to GIGABYTE's website for more information about the supported DDR II memory modules for this feature.

(Note 2) To use a DDR II 667 memory module on the motherboard, you must install an 800/1066MHz FSB processor.

Izgled matične ploče računala (nastavak)

Zbog **bitno različite brzine** pojedinih dijelova računala, postoje još **dva bitna** “chipa” koji povezuju razne dijelove i kontroliraju **komunikaciju** — prijenos podataka između njih. To su:

- Tzv. “**northbridge**” (sjeverni most), koji veže procesor s “**bržim**” dijelovima računala. Standardni brzi dijelovi su:
 - memorija,
 - grafika (grafička kartica).
- Tzv. **southbridge** (južni most), na kojem “visi” većina ostalih “**sporijih**” dijelova ili vanjskih uređaja.

Izgled matične ploče računala (nastavak)

- Tipični uređaji vezani na **southbridge** su:
 - diskovi (koji mogu biti i na dodatnim kontrolerima),
 - DVD i CD uređaji,
 - diskete,
 - komunikacijski portovi,
 - port za pisač (printer),
 - USB (Universal Serial Bus) portovi,
 - tzv. Firewire (IEEE 1394a, b) portovi,
 - mrežni kontroleri,
 - audio kontroleri,
 - dodatne kartice u utorima na ploči (modem), itd.

Izgled matične ploče računala (nastavak)

Veze između pojedinih dijelova idu tzv. “**magistralama**” ili “**sabirnicama**” (engl. **bus**, koji nije autobus).

- Ima **nekoliko** magistrala, **raznih** brzina.
- Na istoj magistrali može biti **više uređaja**, i oni su, uglavnom, **podjednakih** brzina.

Uočite **hijerarhijsku** organizaciju komunikacije pojedinih dijelova:

- najsporiji su vezani na ponešto brže,
- ovi na još brže,
- i tako redom, do najbržeg — procesora.

Ova hijerarhija je **ključna** za efikasnu komunikaciju!

Hijerarhijska struktura memorije

Nažalost, ova hijerarhija komunikacije **nije dovoljna** za efikasnost modernog računala. Grubo govoreći, **fali joj vrh**, koji se ne vidi dobro na izgledu matične ploče.

- Pravo i najgore **usko grlo** u prijenosu podataka je komunikacija između **procesora** i **memorije**.

Gdje je problem?

Podsjetimo: bilo koje **operacije** nad bilo kojim podacima možemo napraviti samo u procesoru — preciznije, u **registrima** procesora. To znači da

- prije same operacije, podatak moramo “dovući” iz obične memorije u neki registar procesora.

Baš to je **sporo!**

Hijerarhijska struktura memorije (nastavak)

Na primjer, ako procesor radi na 3.6 GHz, a memorija na 533 MHz, onda će

- prijenos podatka u registar trajati okruglo 6 puta dulje od operacije na njemu.

Nažalost, isti tehnološki problem se javlja kod svih modernijih računala.

- Obična radna memorija je bitno sporija od procesora.

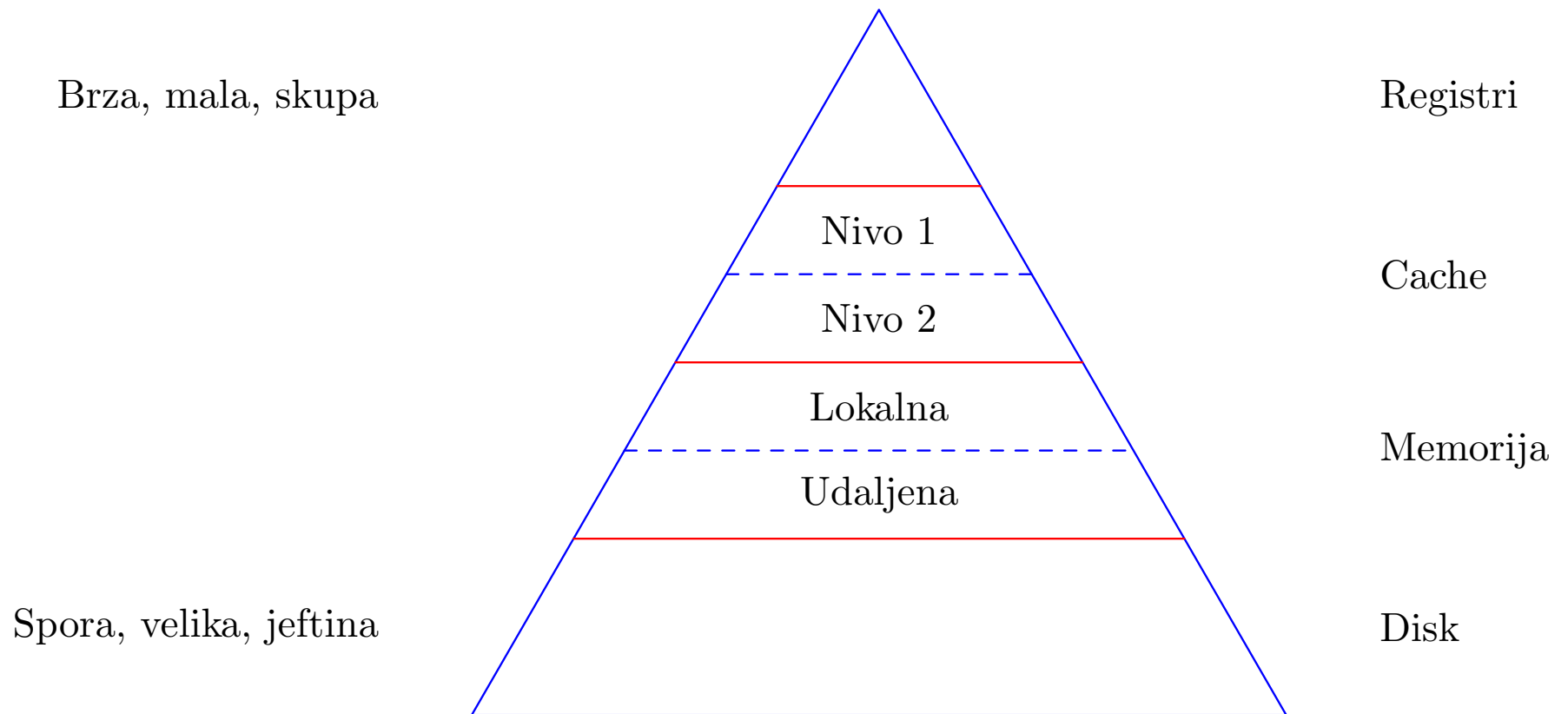
Kako se to izbjegava, ili, barem ublažava?

- Dodatnom hijerarhijskom strukturom memorije, između obične radne memorije (RAM) i registara procesora.

Ta “dodatna” memorija se tradicionalno zove cache.

Hijerarhijska struktura memorije (nastavak)

Globalna struktura memorije u računalu ima oblik:



Cache memorija

Dakle, **cache** je **mala** i **brza** “lokalna” memorija — **bliža** procesoru od obične memorije (RAM). Gdje se nalazi?

- Obično, **na samom procesorskom chipu**, da bude što bliže registrima.

Nadalje, i taj **cache** je **hijerarhijski** organiziran. U modernim procesorima postoji **nekoliko** nivoa (razina) cache memorije.

- **L1** cache za podatke i instrukcije — najbrži, veličina (trenutno) u **KB**.
- **L2** cache za podatke — nešto sporiji, danas obično **na frekvenciji procesora**, veličina već u **MB**.
- Katkad postoji i treća razina — **L3** cache.

Cache memorija (nastavak)

Na primjer, moj “notebook” ima **Intel Pentium 4–M** procesor koji na sebi ima (bez pretjeranih tehničkih detalja):

- L1 cache za podatke — 8 KByte-a,
- L1 cache za instrukcije — 12 K tzv. mikro-operacija,
- L2 cache — 512 KByte-a, na frekvenciji procesora.

Ovo su tipični omjeri veličina za **Intelove** procesore.

Za usporedbu, na **AMDovim** procesorima omjeri su bitno **drugačiji**:

- L1 cache je **veći**,
- L2 cache nešto **manji** (i, katkad, sporiji).

(Ne ulazimo u to što je bolje!)

Cache memorija (nastavak)

Kako (ugrubo) **radi** cache?

Kad računalo (tj. njegov operacijski sustav) **izvršava** neki naš **program**, onda

- uglavnom, **imamo** kontrolu **sadržaja** obične memorije koju taj naš program koristi za podatke i naredbe.

Za razliku od toga,

- **nemamo** nikakvu **izravnu** kontrolu nad sadržajem **cache** memorije.

Naime, cache **nije izmišljen** zato da bude **mala**, **brža** kopija obične memorije i tako ubrza ukupni rad računala.

Cache memorija (nastavak)

Puno je **efikasnije** da

- **cache** sadrži podatke koji se **češće** koriste.

Isto vrijedi i za instrukcije. Dakle, **osnovna ideja** je:

- “Skrati put do onog što ti često treba”.

Naravno, **ključna** stvar za efikasnost je:

- Što znači “češće” korištenje nekog podatka ili instrukcije?

Dobra **globalna** ili **prosječna** efikasnost postiže se samo ako se **to odnosi** na **sve** što računalo izvršava u nekom trenutku, tj. na sve pokrenute korisničke programe i dijelove operacijskog sustava.

Cache memorija (nastavak)

U tom svjetlu, kad malo bolje razmislite,

- zaista bi bilo **nepraktično** da svaki programer određuje što i kada **treba ići** u koju cache memoriju,

jer prosječna efikasnost nipošto **ne ovisi** samo o njegovom programu. Zato **nema posebnih naredbi** za

- **učitavanje** podataka u cache, ili
- **pisanje** podataka iz cachea u običnu memoriju.

Umjesto toga, **sadržajem** cachea upravljaju posebni **cache kontroleri**, koji

- raznim tehnikama “**asocijacije**” na više načina povezuju nedavno korištene podatke i instrukcije s onima koje **tek treba iskoristiti i izvršiti**.

Cache memorija (nastavak)

Bez puno tehničkih detalja, ova **asocijacija** se realizira otprilike ovako:

- Za svaki **sadržaj** (**podatak** ili **instrukciju**) u cacheu, dodatno se pamti i **adresa** (iz RAM-a), s koje je taj **sadržaj** stigao.
- Ako procesor (uskoro) **zatraži** **sadržaj** s te **adrese**, on se “**čita**” iz cachea (tj. ne treba po njega ići u RAM).
- Po istom sistemu, u cacheu se **pamte** i stvari koje se “**pišu**” u običnu memoriju (na putu u RAM).
- Tada se iz cachea **brišu** podaci koji su **najstariji**, odnosno, **najmanje korišteni** (u zadnje vrijeme, otkad su u cacheu).

Cache memorija (nastavak)

Dakle, sadržaj cachea se **stalno obnavlja**, tako da

- cache čuva **najčešće nedavno korištene sadržaje** koji bi **uskoro mogli trebati**.

Iskustvo pokazuje da se **isti sadržaji** vrlo često koriste **više puta**, pa se ovo isplati.

Očiti primjer:

- **instrukcije u petljama** se ponavljaju puno puta!

Ne zaboravimo da je upravo to svrha programiranja i osnovna korist računala.

Cache memorija (nastavak)

Malo kompliciranije je s **podacima**.

- Ako naš **algoritam** ne koristi iste podatke **puno puta**, onda nam cache **neće ubrzati** postupak.
- U suprotnom, isplati se **preurediti** algoritam tako da **iste podatke** koristi **puno puta**, ali u **kratkom vremenskom razmaku** — da ne “izlete” iz cachea. (To je **neizravna** kontrola nad sadržajem **cachea**.)

Primjeri iz **linearne algebre**:

- **zbrajanje** matrica, $C = A + B$ — cache **ne pomaže** puno;
- **množenje** matrica, $C = C + A * B$ — dobro korištenje **cachea** može ubrzati množenje matrica i za **5 puta**.