Programiranje 2
4. predavanje — dodatak

Saša Singer

singer@math.hr
web.math.pmf.unizg.hr/~singer

PMF – Matematički odsjek, Zagreb

Sadržaj predavanja — dodatka

- Brzina množenja kvadratnih matrica:
  - Matrica — dimenzionirana na maksimalni red.
  - Alocirani vektor — adresiranje kao u matrici maksimalne dimenzije.
  - Alocirani vektor — adresiranje kao u matrici stvarne dimenzije.

Brzina množenja kvadratnih matrica

Množenje matrica

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

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

Akumulacija ("nazbrajavanje") produkta $A * B$ u matrici C

- standardni je oblik BLAS–3 rutine `xGEMM` za množenje matrica,

tj. baš ova operacija se često koristi u praksi.

Usput, to će opet

- "prevariti" optimizaciju compilera,

kod višestrukog ponavljanja eksperimenta.

Množenje matrica — formula

"Matematička" realizacija matrične operacije

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

po elementima je trivijalna:

$$c_{ij} := c_{ij} + \sum_{k=1}^{n} a_{ik} \cdot b_{kj},$$

za sve indekse

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

Dakle, "programski" — treba "zavrtiti" tri petlje.

Množenje matrica — ijk

```
void mulijk ( int n, double a[][LDA],
              double b[][LDA], double c[][LDA] )
{
    int i, j, k;

    for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
            for (k = 0; k < n; ++k)
                c[i][j] += a[i][k] * b[k][j];

    return;
}
```

LDA = Last Dimension of A, iz rezervacije memorije za matrice.

Permutacija petlji

Ovu varijantu algoritma zovemo `ijk` — opet, po poretku (indeksa) petlji, izvana prema unutra.

Sve tri petlje možemo permutirati, tj. napisati ih u bilo kojem poretku. Na taj način dobivamo ukupno 6 varijanti algoritma, koje zovemo leksikografskim redom:

- `ijk`,
- `ikj`,
- `jik`,
- `jki`,
- `kij`,
- `kji`.

Opis eksperimenta

Napravit ćemo 4 klase eksperimenata.

U prva dva eksperimenta koristimo "prave" matrice,

- dimenzionirane (u glavnom programu) na maksimalnu potrebnu dimenziju `[LDA][LDA]`,
- Pristup elementu M_{ij} je `M[i][j]`.

Uspoređujemo dvije različite optimizacije prevoditelja:

- normalna optimizacija — bez ikakvih opcija, oznaka (n),
- fast optimizacija — s odgovarajućim opcijama,
  - tako da prevoditelj vektorizira i permutira petlje.

  Zato dobijemo grupe petlji s gotovo istom brzinom.

Opis eksperimenta (nastavak)

U druga dva eksperimenta, umjesto matrica, koristimo

- dinamički alocirane vektore,
- s tim da se alokacija radi samo jednom, na maksimalnu potrebnu dimenziju `LDA * LDA`.

To je "klasični" načina rada s "matricama" u C-u (C90).

Uspoređujemo dva različita načina adresiranja "matričnih" elemenata u vektoru:

- "po recima", kao u matrici maksimalnog reda `LDA`,
  - pristup elementu M_{ij} je `M[i * LDA + j]`.
- "u bloku", kao u matrici trenutnog reda n,
  - pristup elementu M_{ij} je `M[i * n + j]`.

Množenje matrica, vektor "po recima" — ijk

```
void mulijk ( int lda, int n, double a[],
              double b[], double c[] )
{
    int i, j, k;

    for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
            for (k = 0; k < n; ++k)
                c[i * lda + j] += a[i * lda + k]
                                * b[k * lda + j];

    return;
}
```

Množenje matrica, vektor "u bloku" — ijk

```
void mulijk ( int n, double a[],
              double b[], double c[] )
{
    int i, j, k;

    for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
            for (k = 0; k < n; ++k)
                c[i * n + j] += a[i * n + k]
                              * b[k * n + j];

    return;
}
```

Opis eksperimenta (nastavak)

U zadnja dva eksperimenta koristimo

- samo fast optimizaciju.

Normalna optimizacija je, možda, zanimljiva — ali prespora!

Zadnji detalji eksperimenta:

- "obična" C90 sintaksa, bez polja varijabilne duljine (C99),
- prevoditelj: Intel C compiler — verzija 9.1.032,
- računalo: Intel Pentium 4/660 (single core), na 3.6 GHz,
- memorija: 2 GB,
- idemo do maksimalnog reda `LDA = 2000`.

Opis slika za svaki eksperiment

Za svaki eksperiment ima 8 slika — grafova brzine:

- prvo ide 6 pojedinačnih slika, leksikografskim redom, po petljama,
- zatim ide zajednički graf za svih 6 petlji,
- na kraju je usporedba:
  - najbrže petlje `ikj` i
  - MKL-ovog algoritma `DGEMM`.

Intelov MKL (Math Kernel Library) je posebno optimiziran, tako da

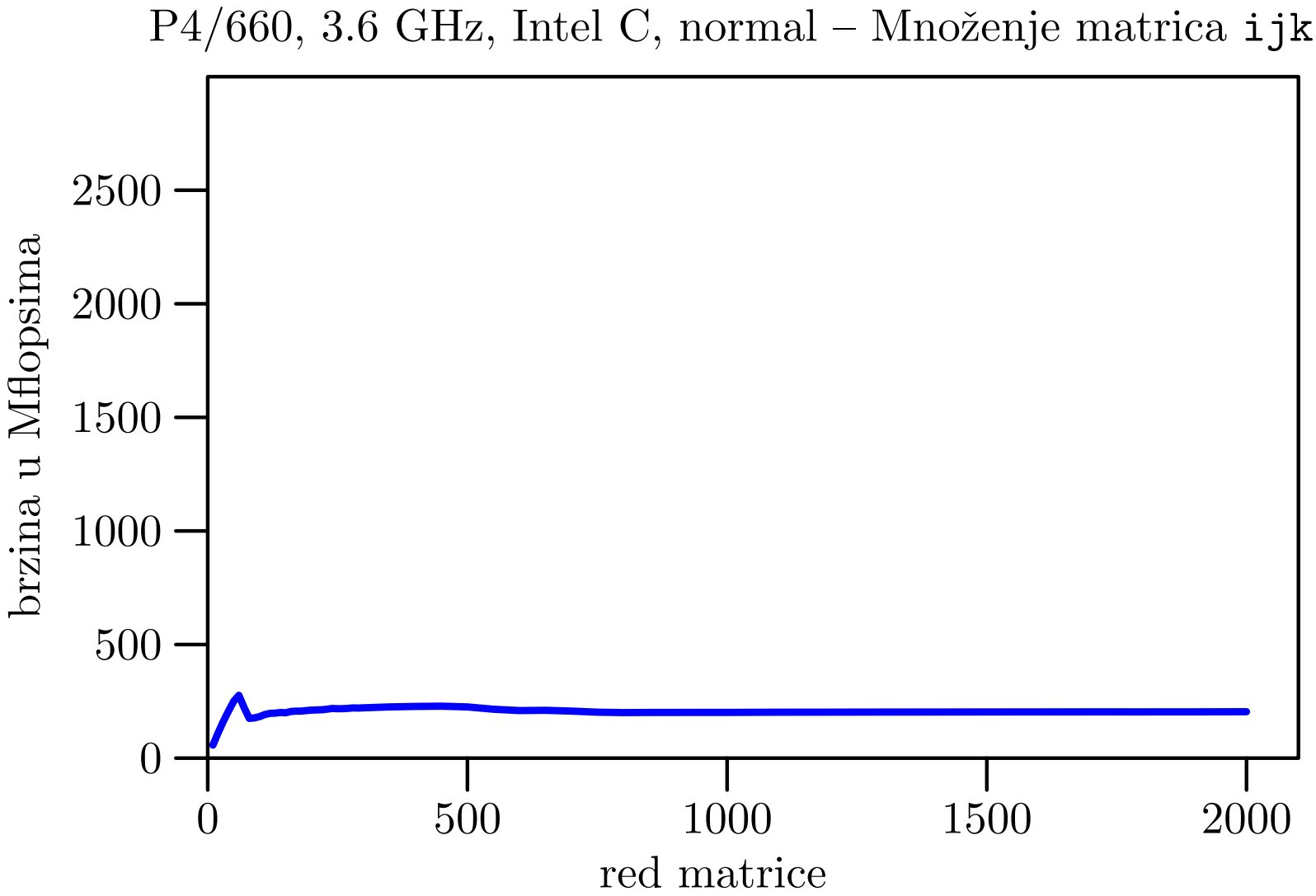
- efikasno koristi cache memoriju.

Boje na grafovima i rangovi

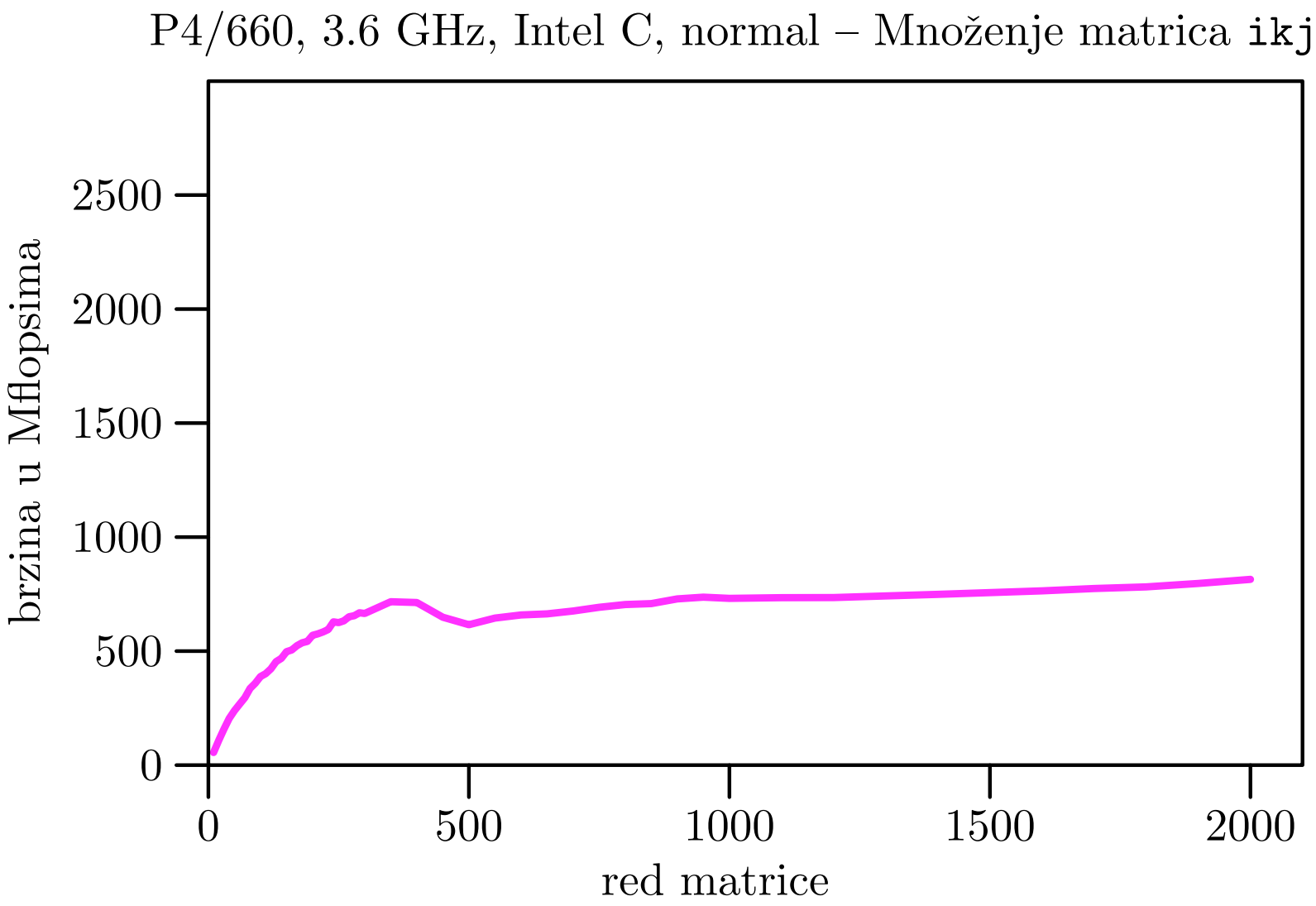
Legenda za čitanje grafova i rang u svakom od 4 eksperimenta:

- petlja `ijk` — plavo, rang: 4, 1–2, 1–2, 2–5.
- petlja `ikj` — ljubičasta, rang: 1, 1–2, 1–2, 1;
- petlja `jik` — crveno, rang: 3, 3–4, 5, 2–5;
- petlja `jki` — narančasta, rang: 5, 3–4, 6, 2–5;
- petlja `kij` — žuta, rang: 2, 5–6, 3–4, 6;
- petlja `kji` — zeleno, rang: 6, 5–6, 3–4, 2–5;

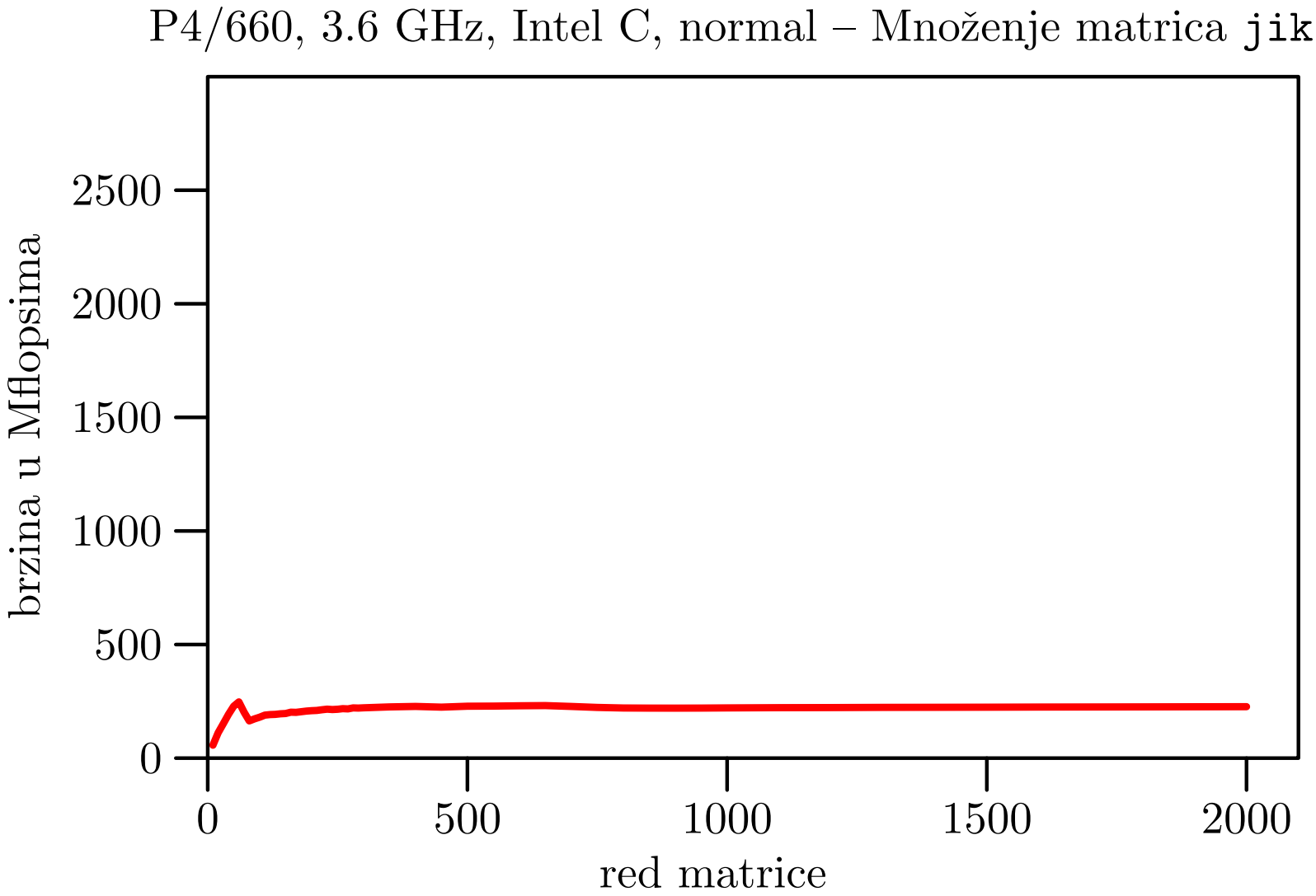
Matrica $M[i][j]$ reda 2000, (n) — `ijk`



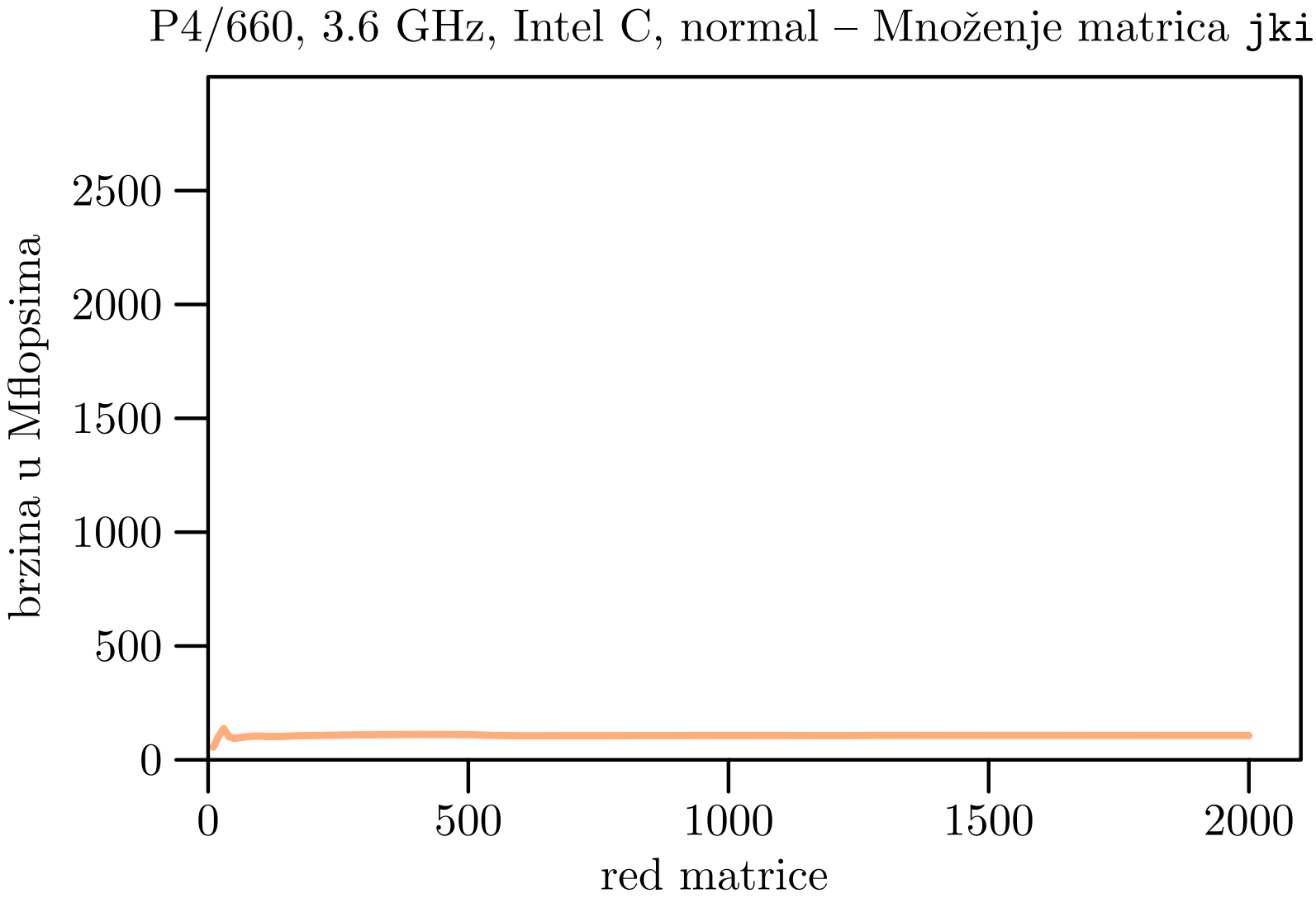
Matrica $M[i][j]$ reda 2000, (n) — `ikj`



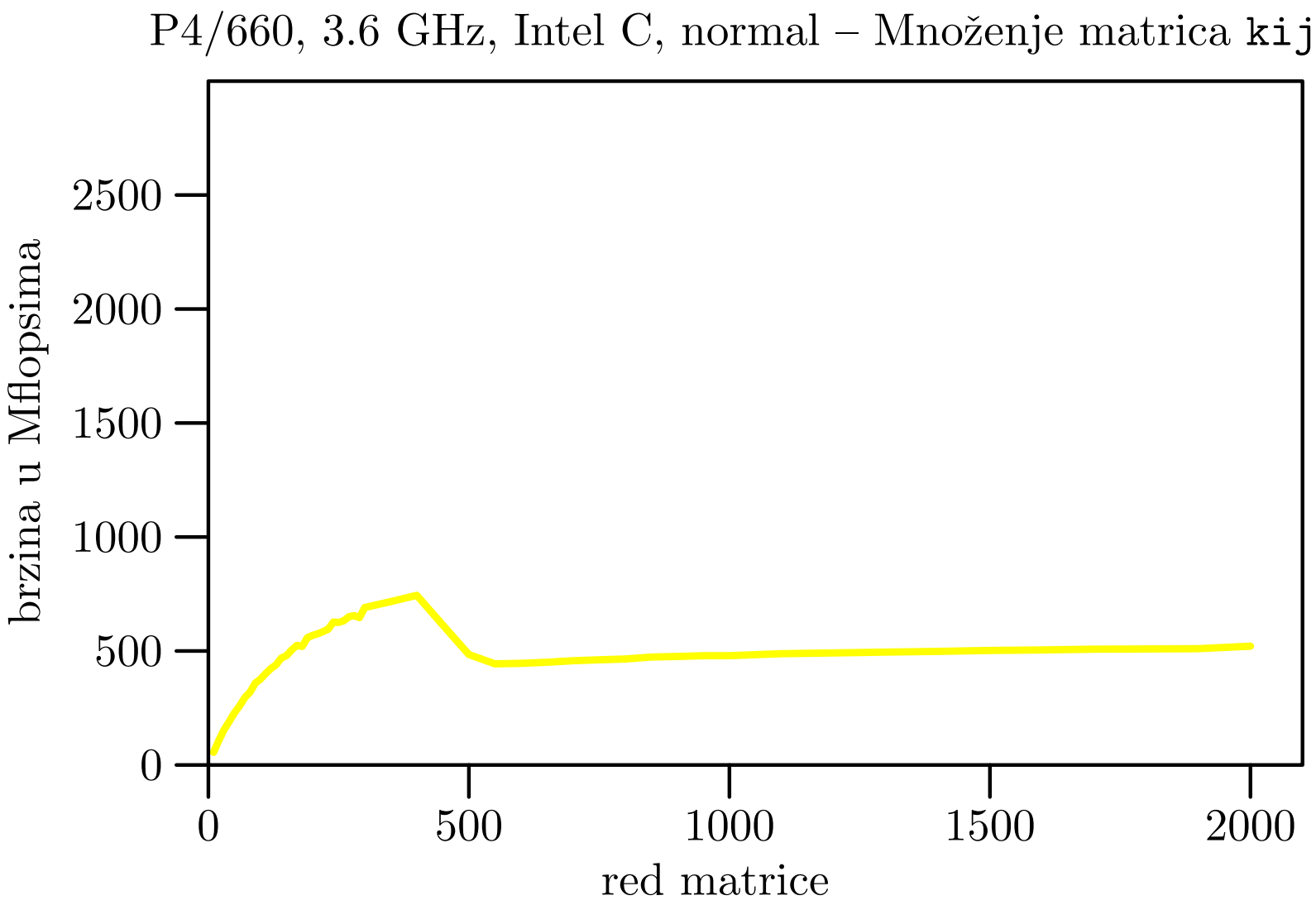
Matrica $M[i][j]$ reda 2000, (n) — `jik`



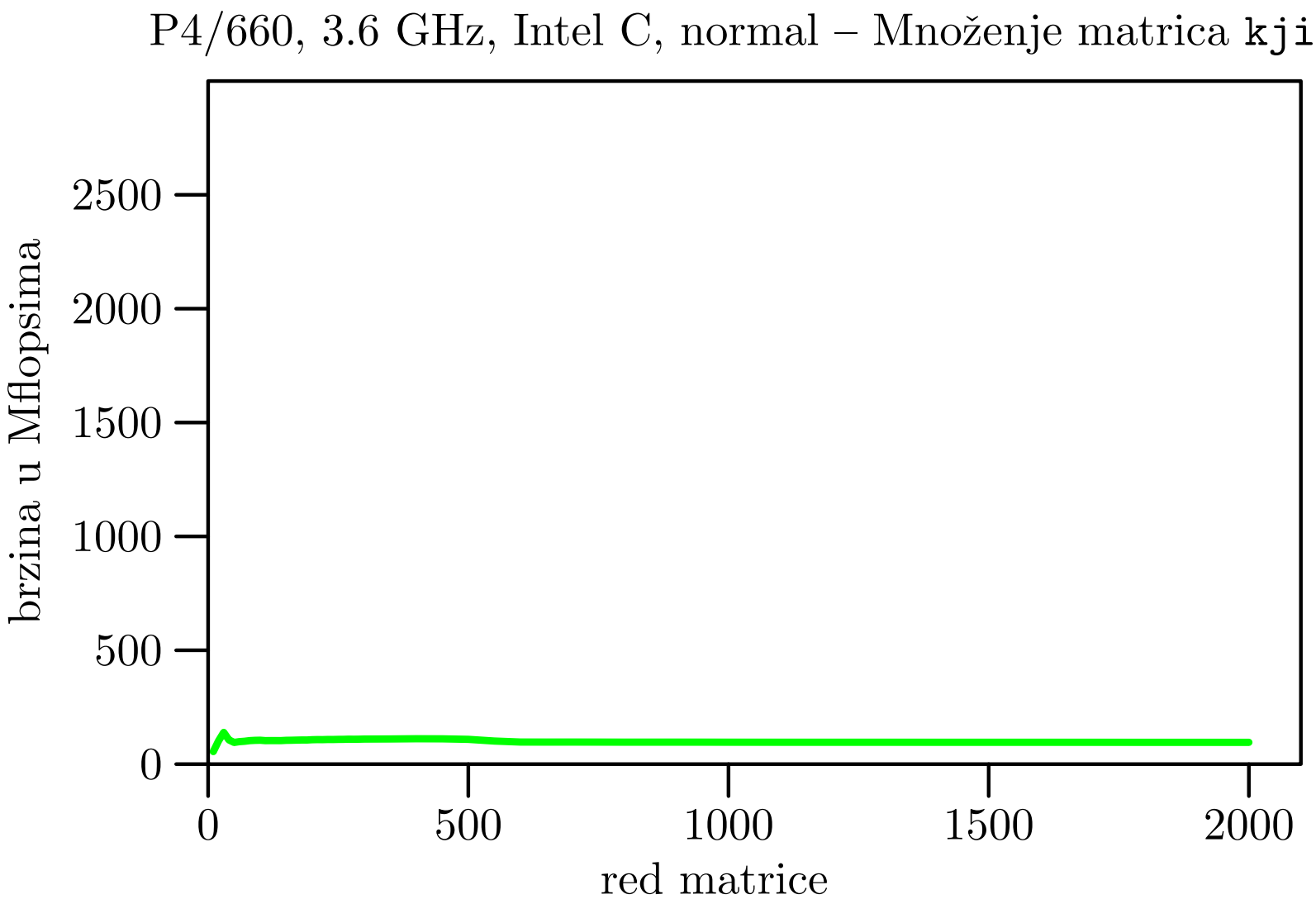
Matrica $M[i][j]$ reda 2000, (n) — `jki`



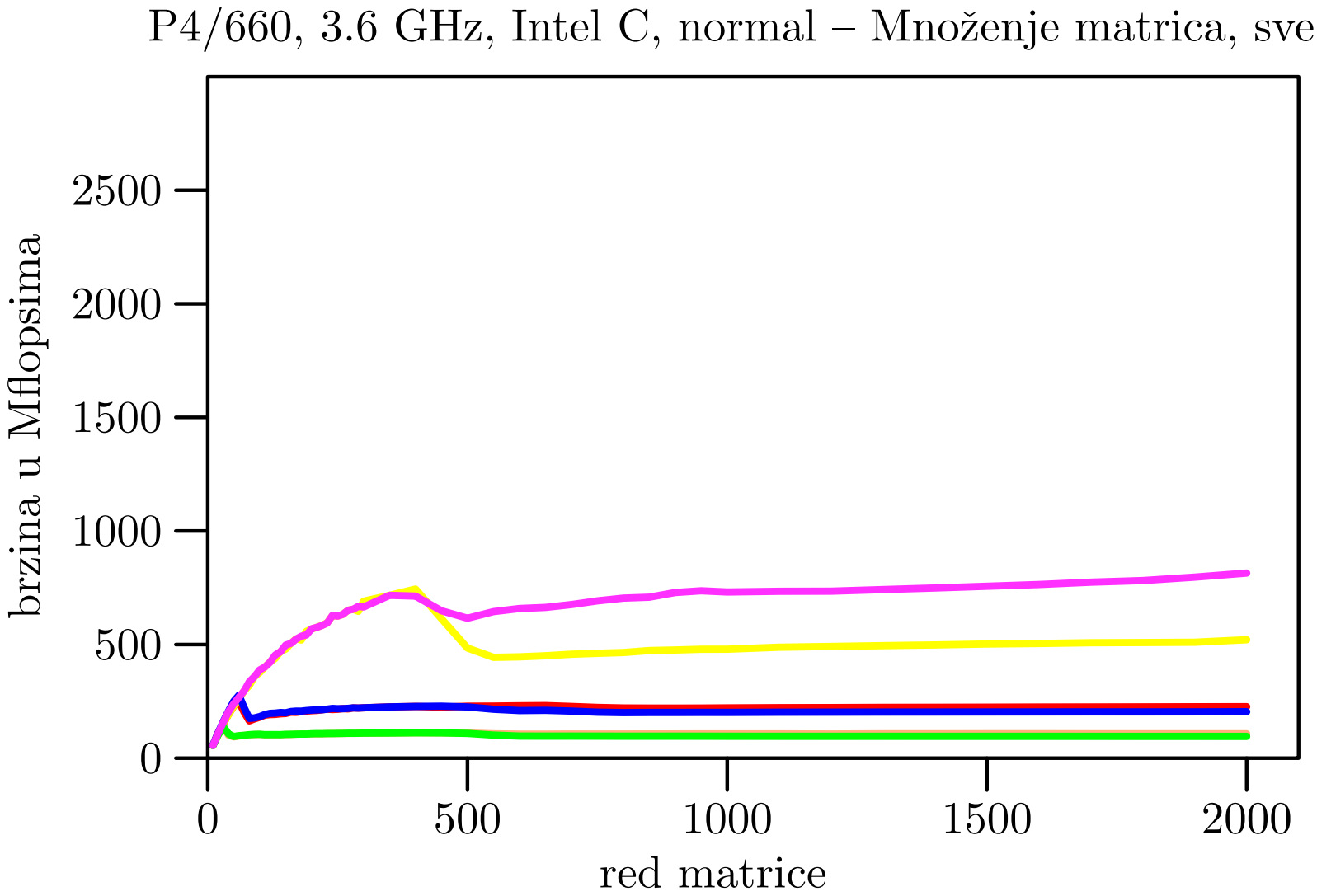
Matrica $M[i][j]$ reda 2000, (n) — `kij`



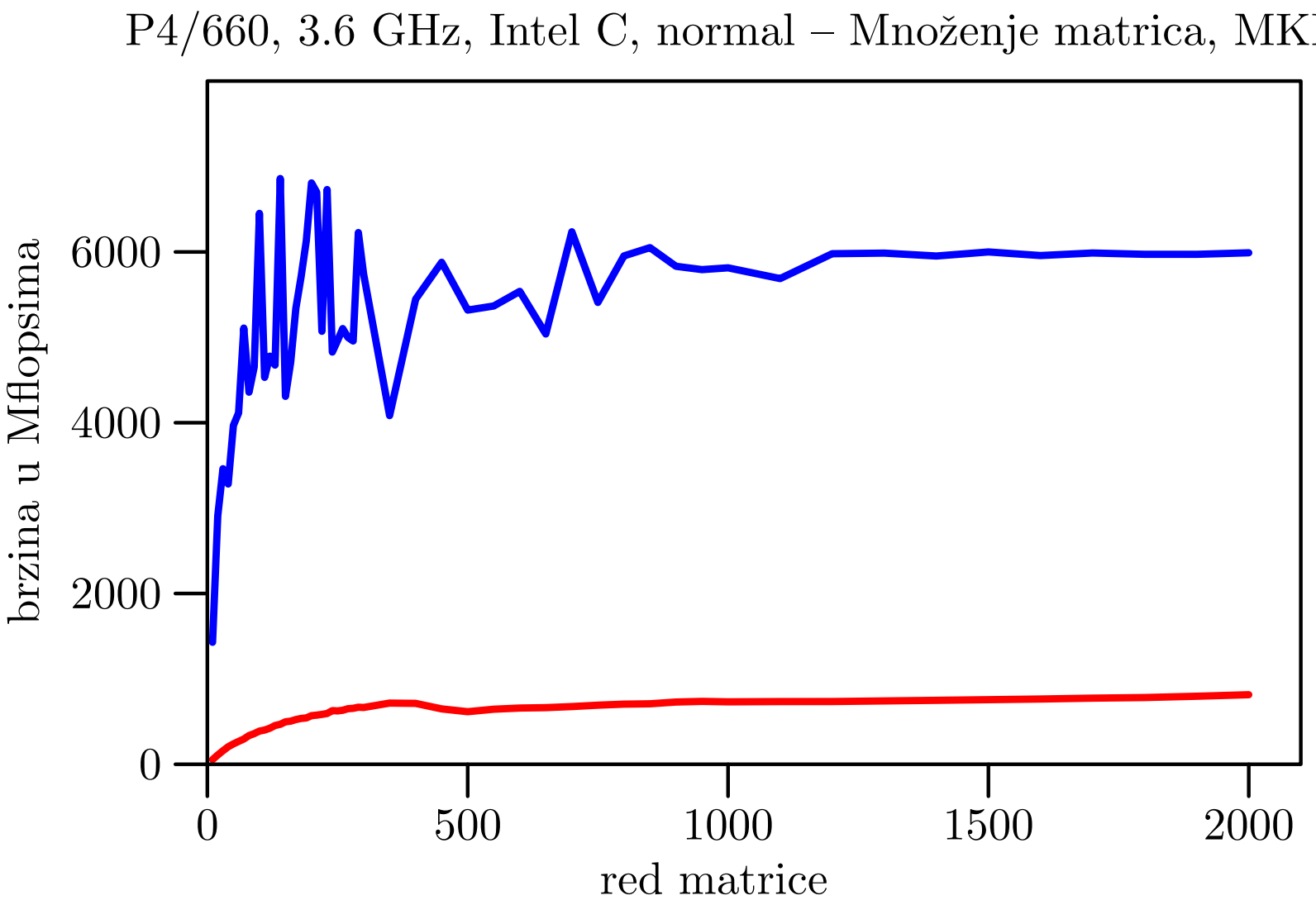
Matrica $M[i][j]$ reda 2000, (n) — `kji`



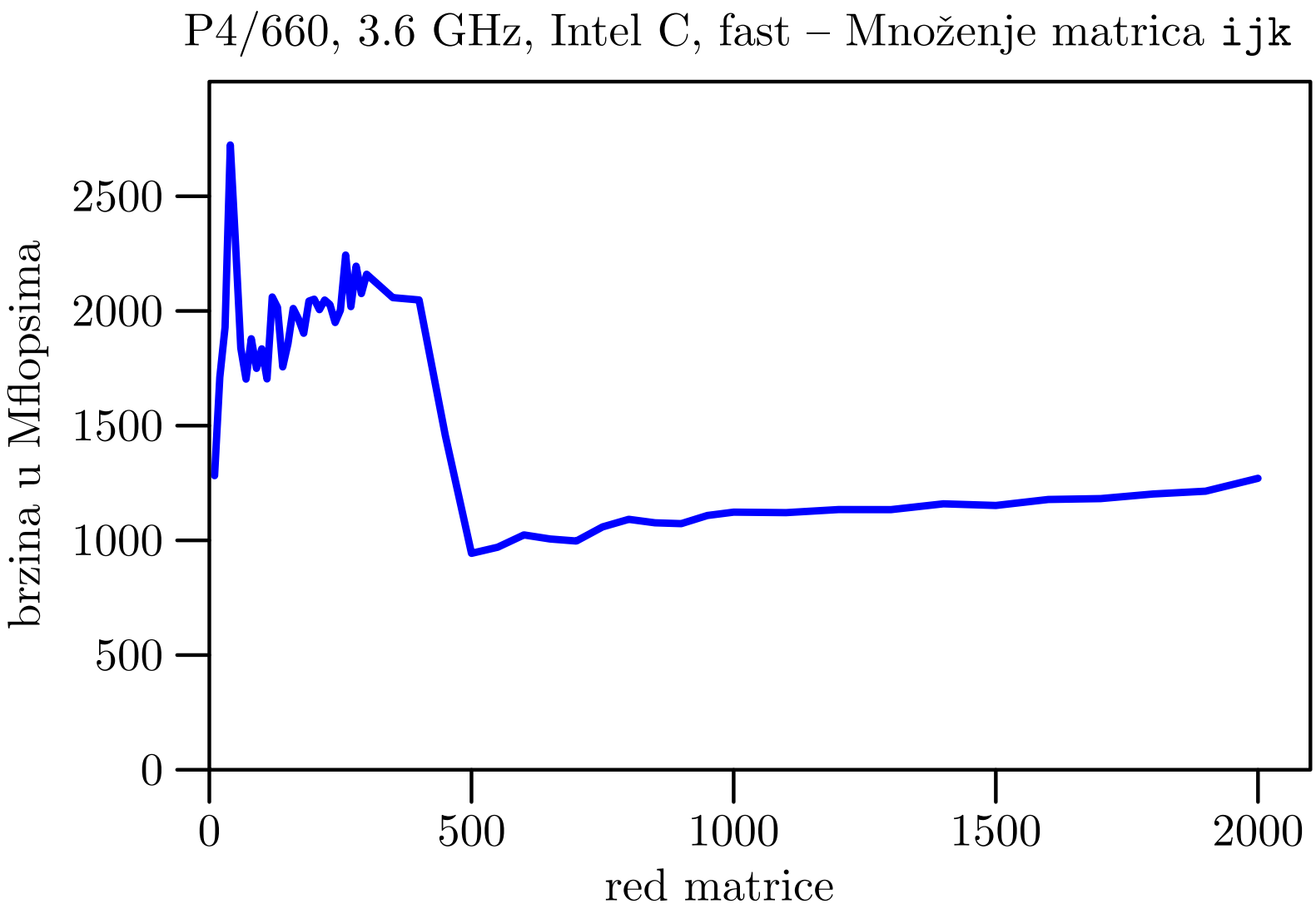
Matrica $M[i][j]$ reda 2000, (n) — sve zajedno



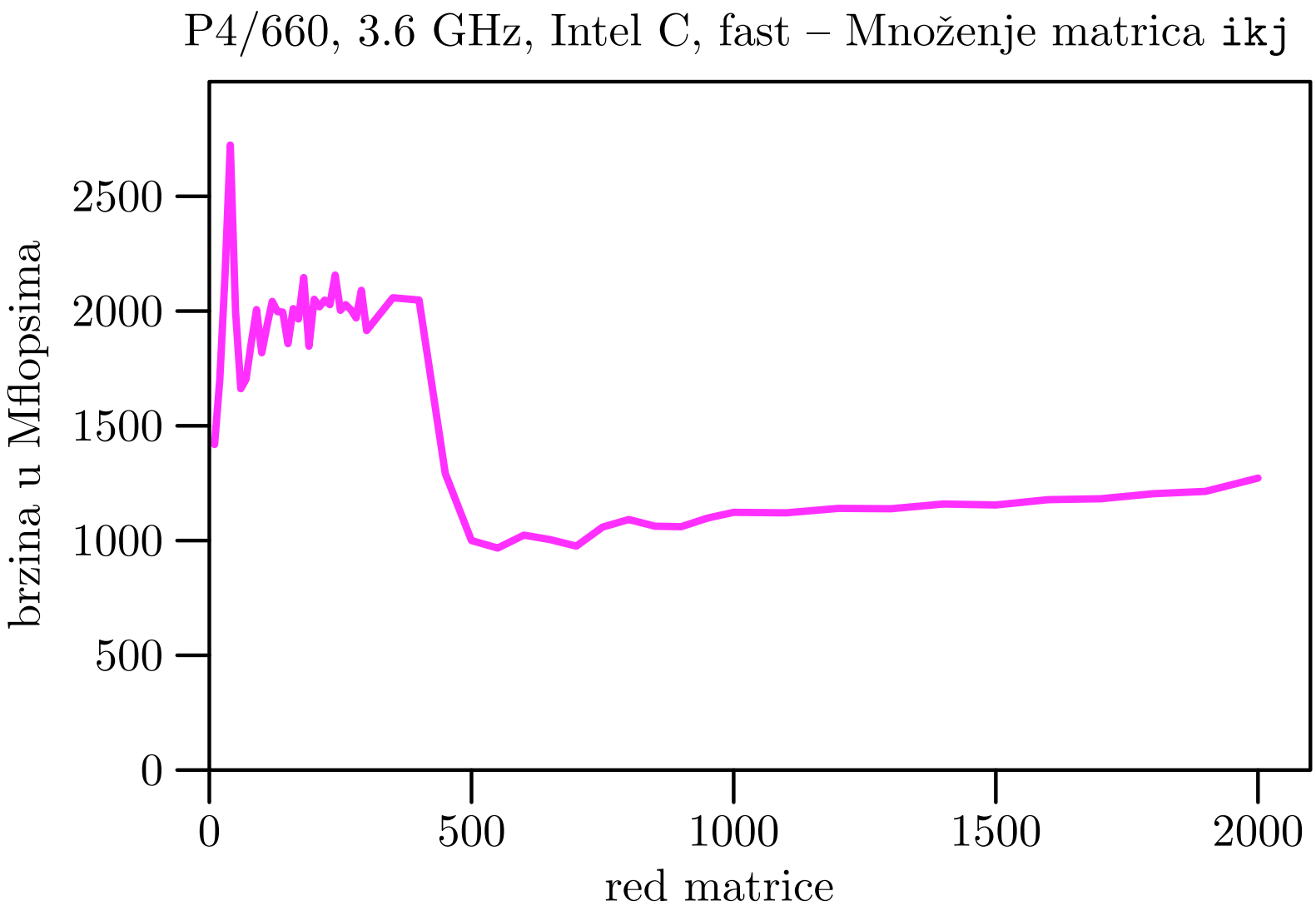
Matrica $M[i][j]$ reda 2000, (n) — najbrži i MKL



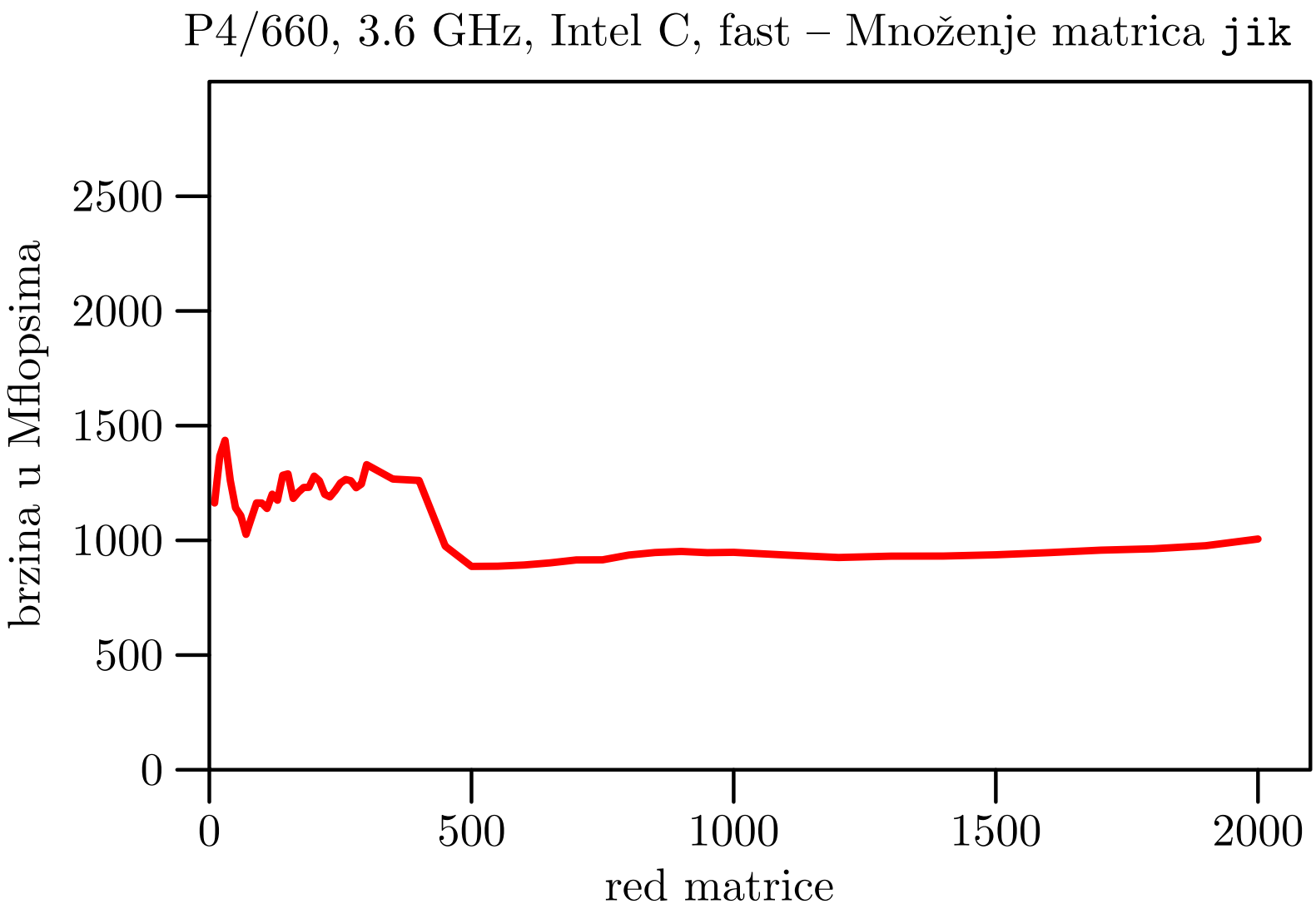
Matrica $M[i][j]$ reda 2000 — `ijk`



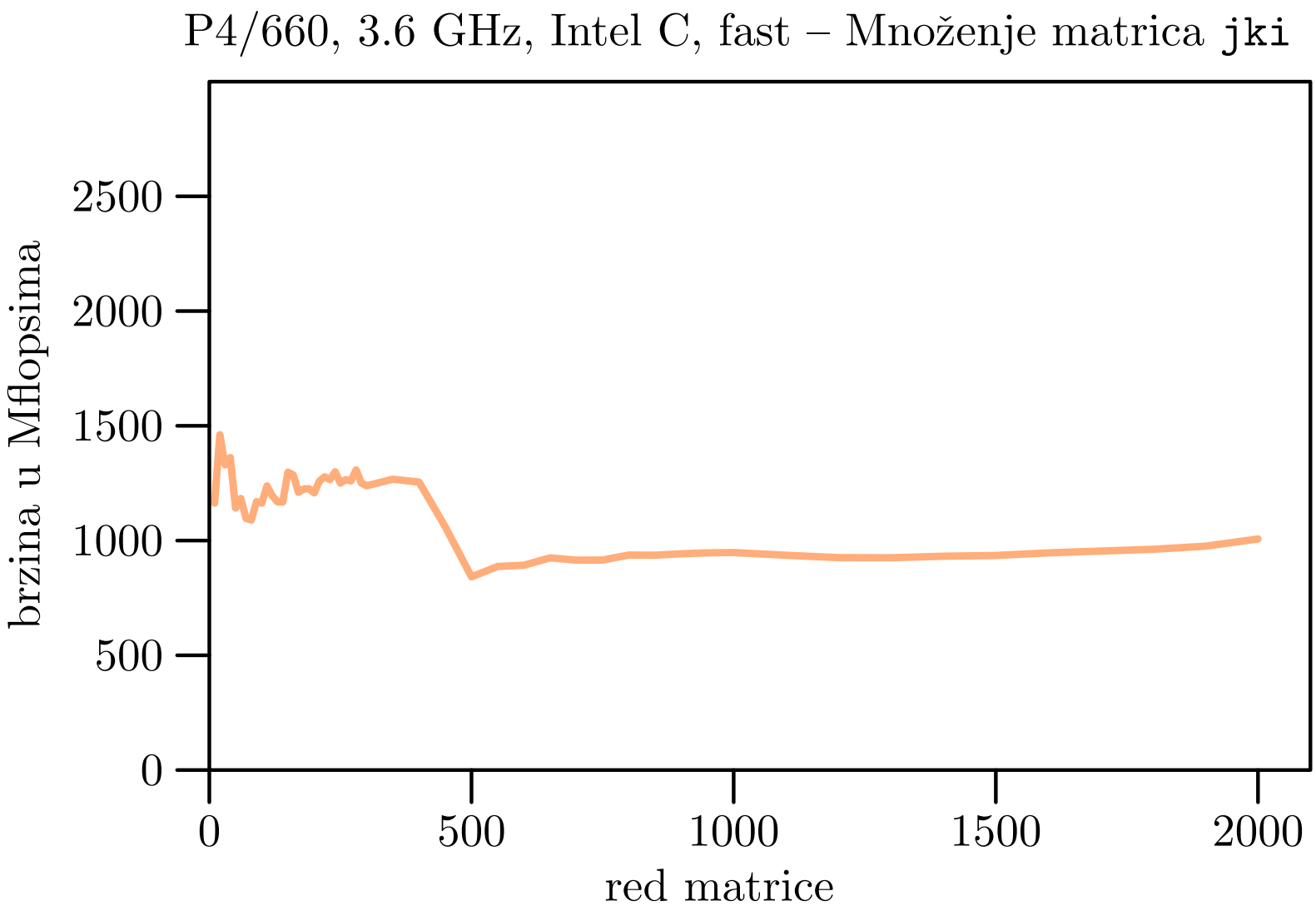
Matrica $M[i][j]$ reda 2000 — `ikj`



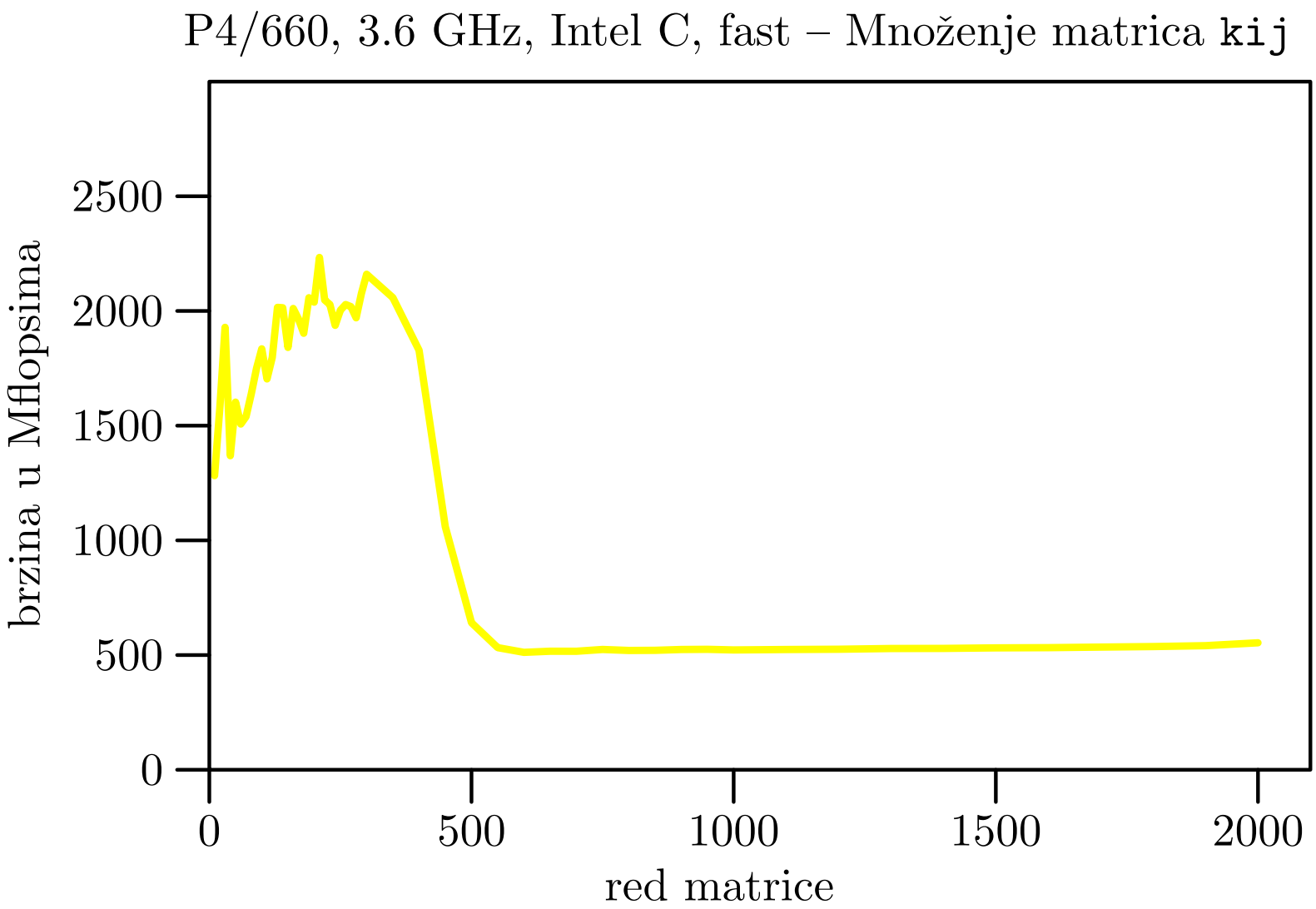
Matrica $M[i][j]$ reda 2000 — `jik`



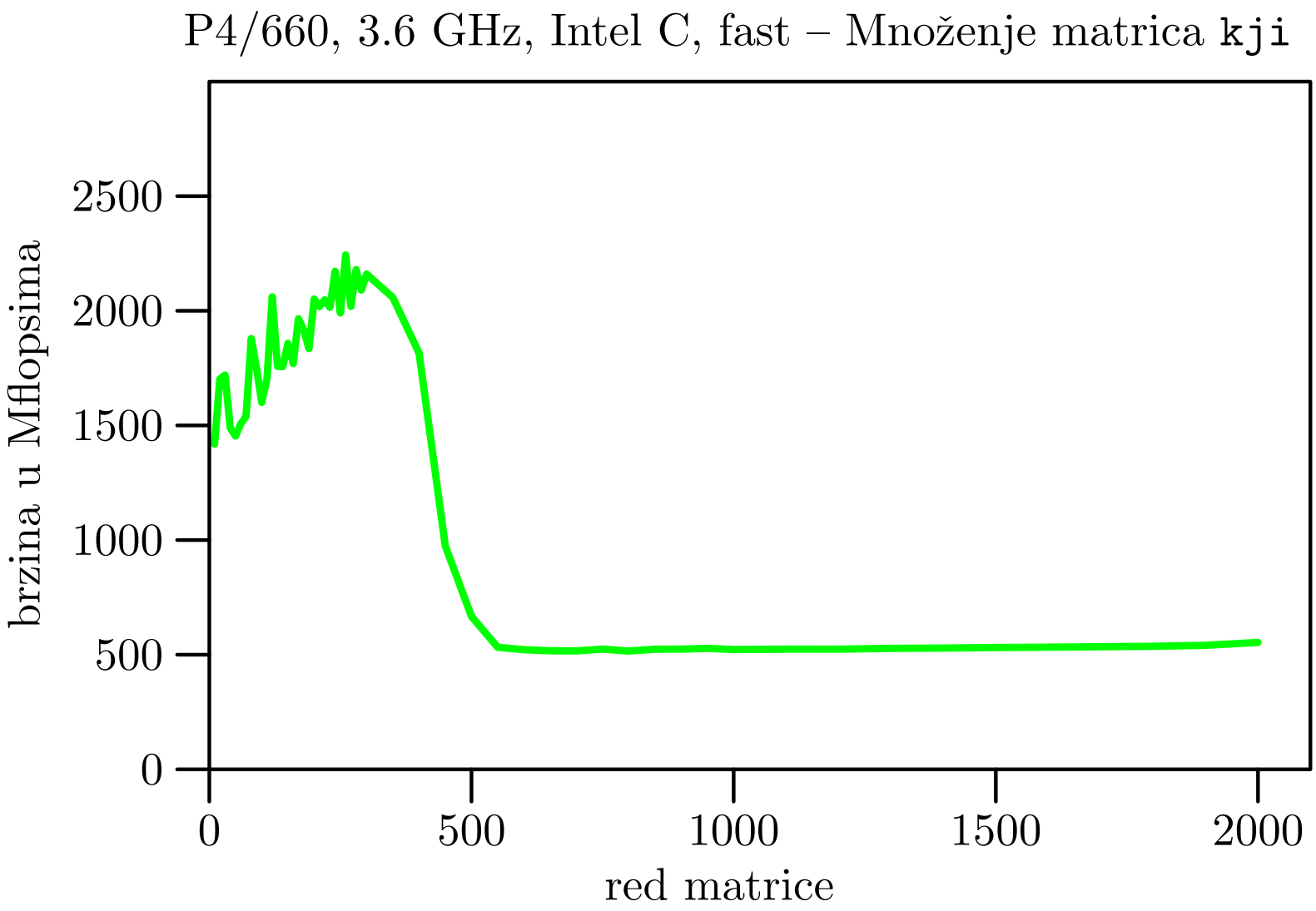
Matrica $M[i][j]$ reda 2000 — `jki`



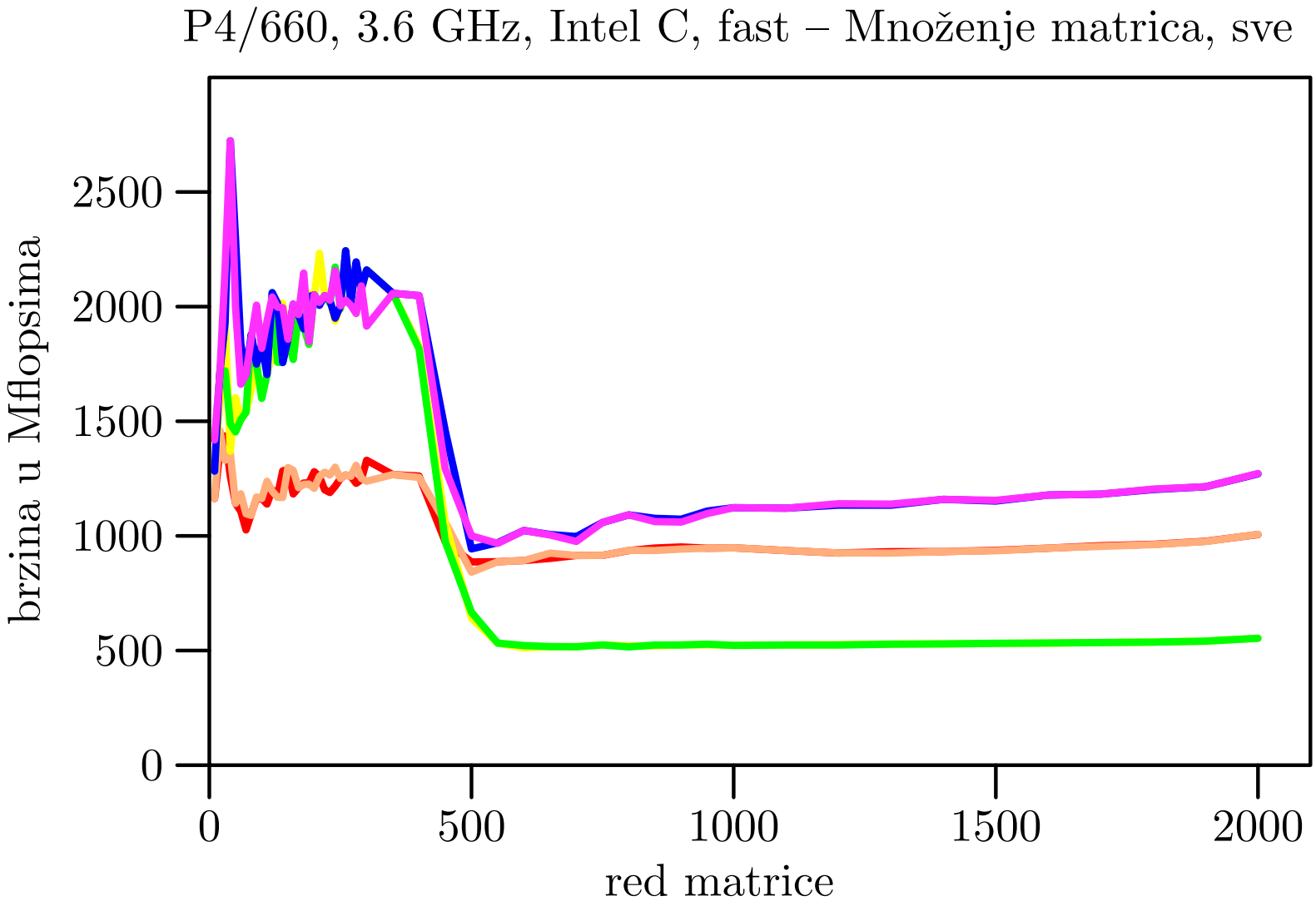
Matrica $M[i][j]$ reda 2000 — `kij`



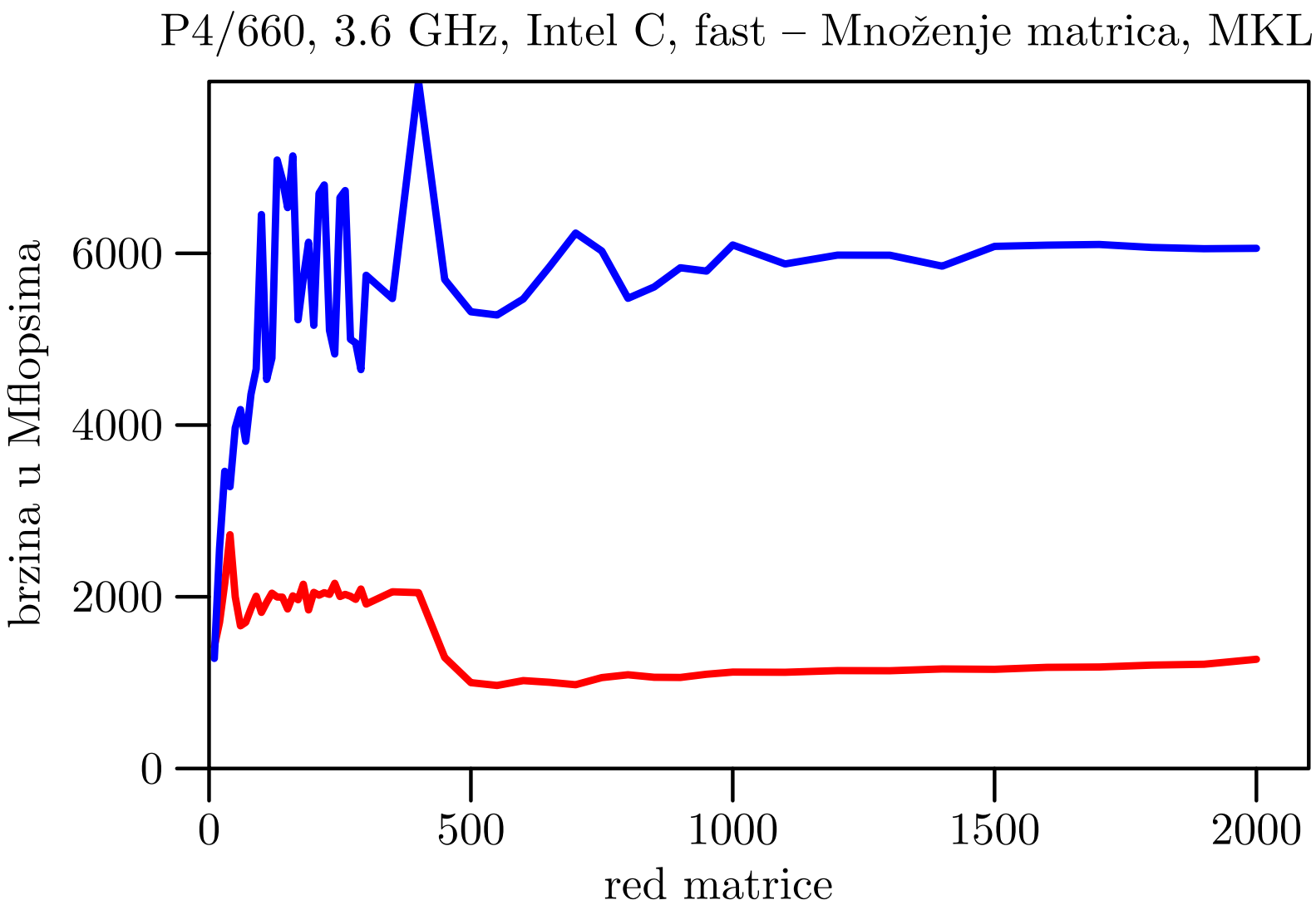
Matrica $M[i][j]$ reda 2000 — `kji`



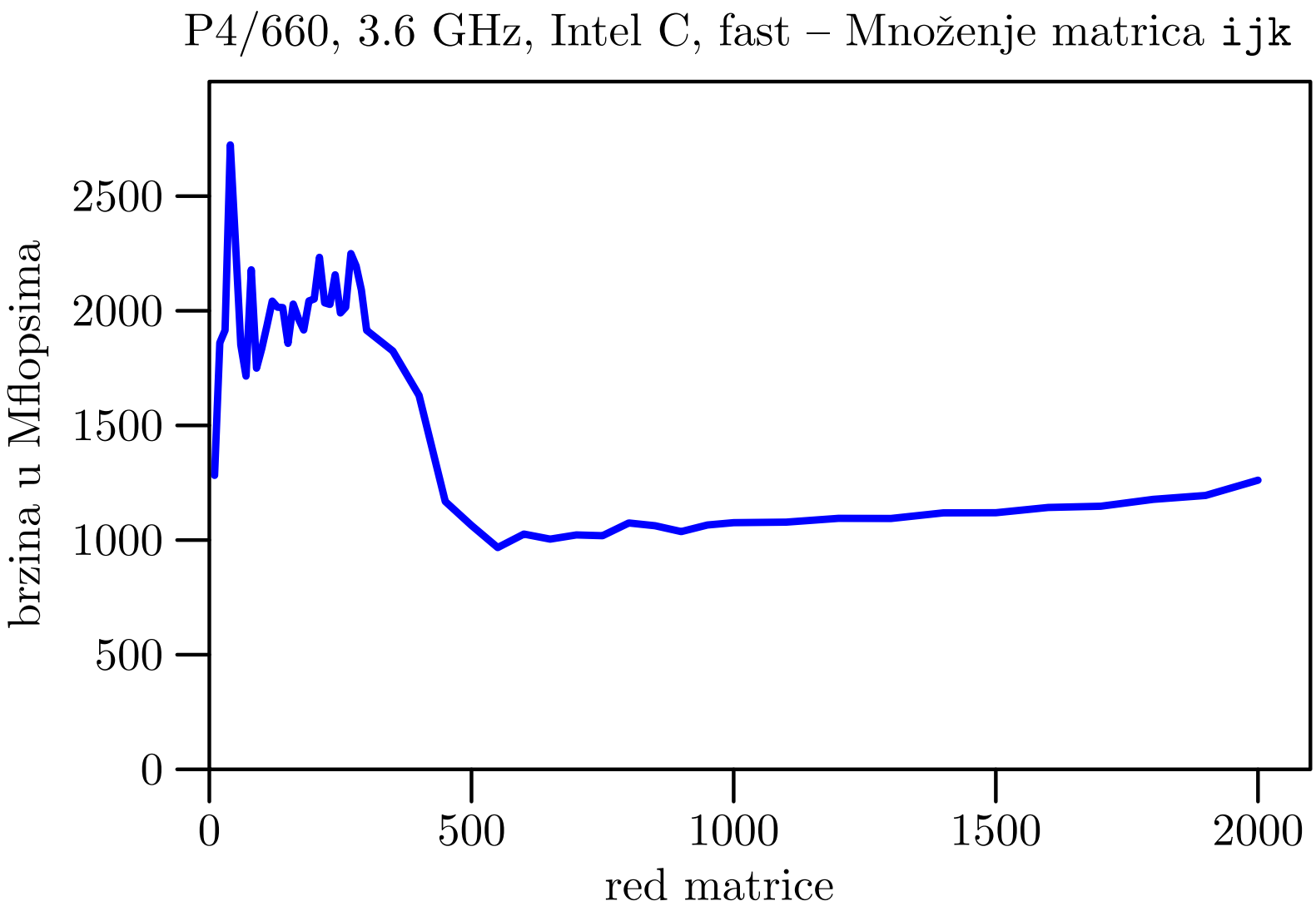
Matrica $M[i][j]$ reda 2000 — sve zajedno



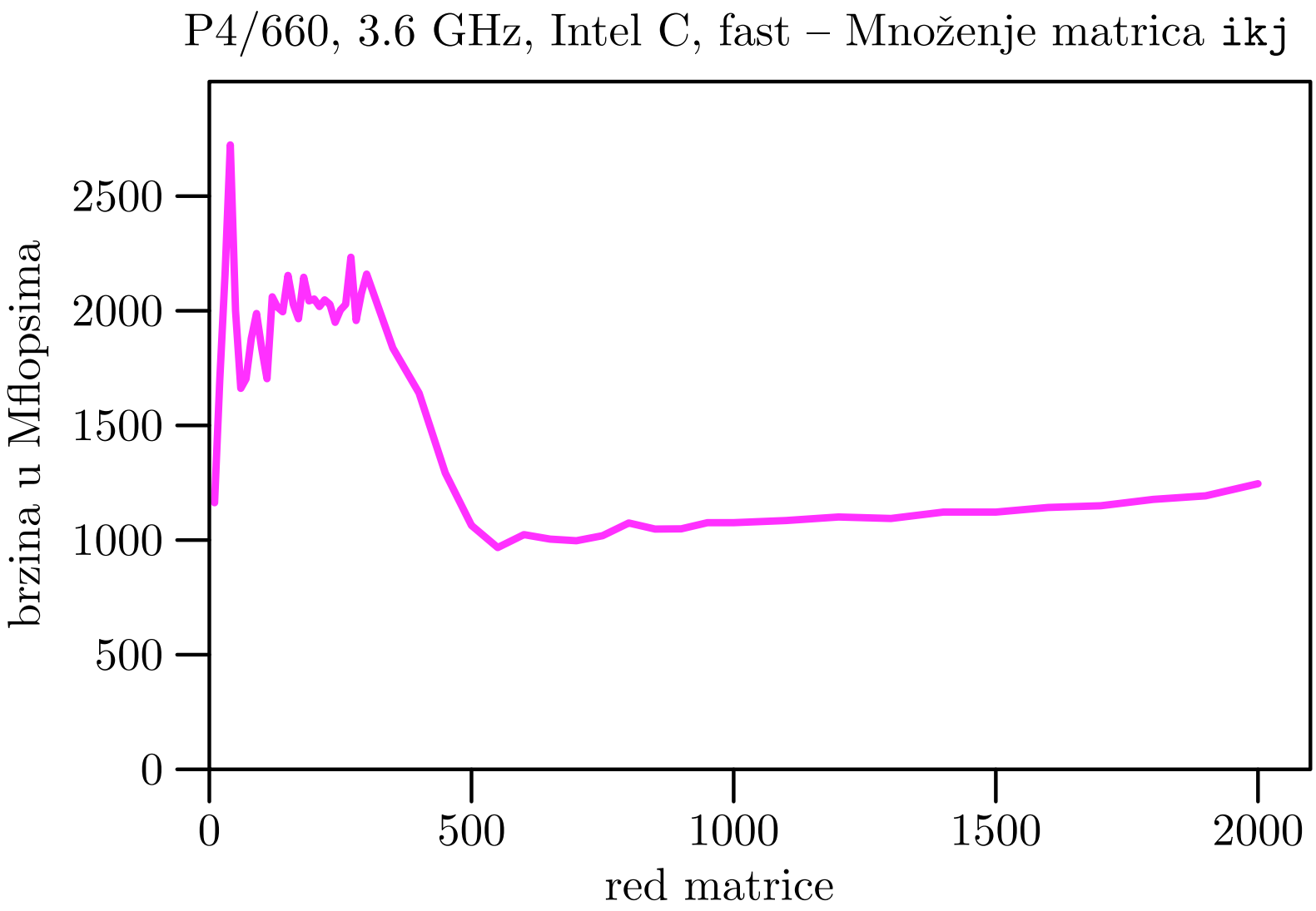
Matrica $M[i][j]$ reda 2000 — najbrži i MKL



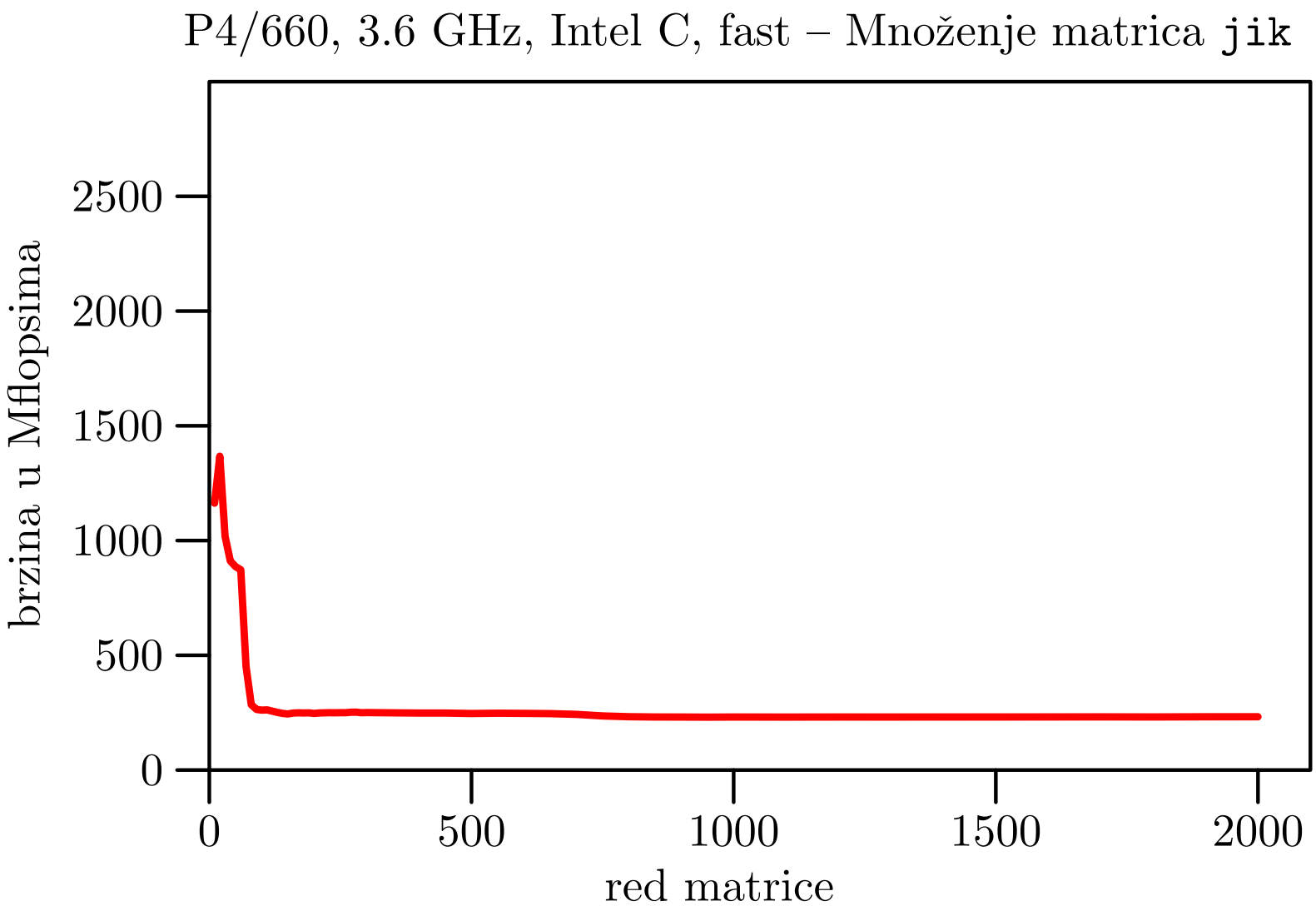
Alocirani vektor $M[i * 2000 + j]$ — `ijk`



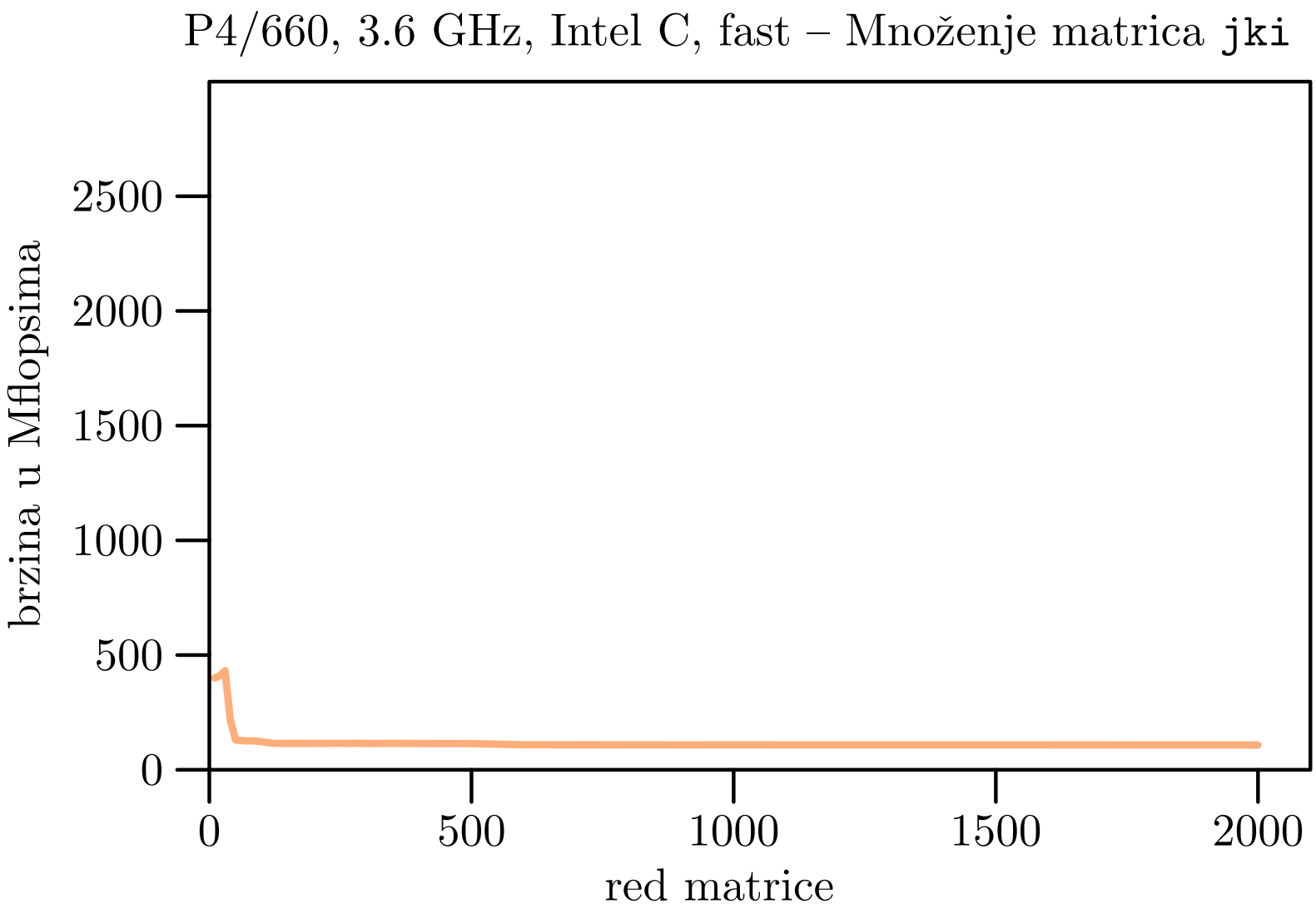
Alocirani vektor $M[i * 2000 + j]$ — `ikj`



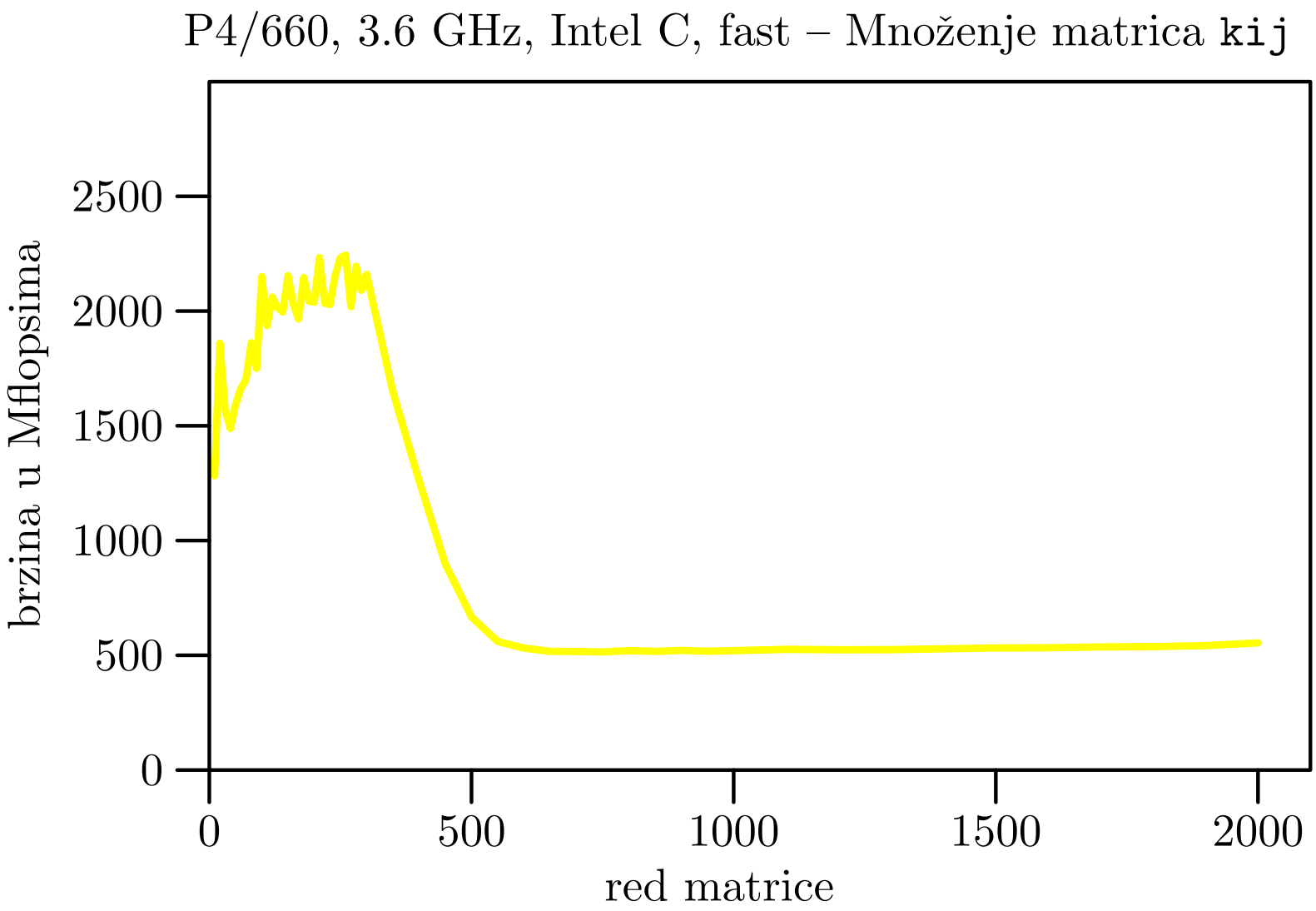
Alocirani vektor $M[i * 2000 + j]$ — `jik`



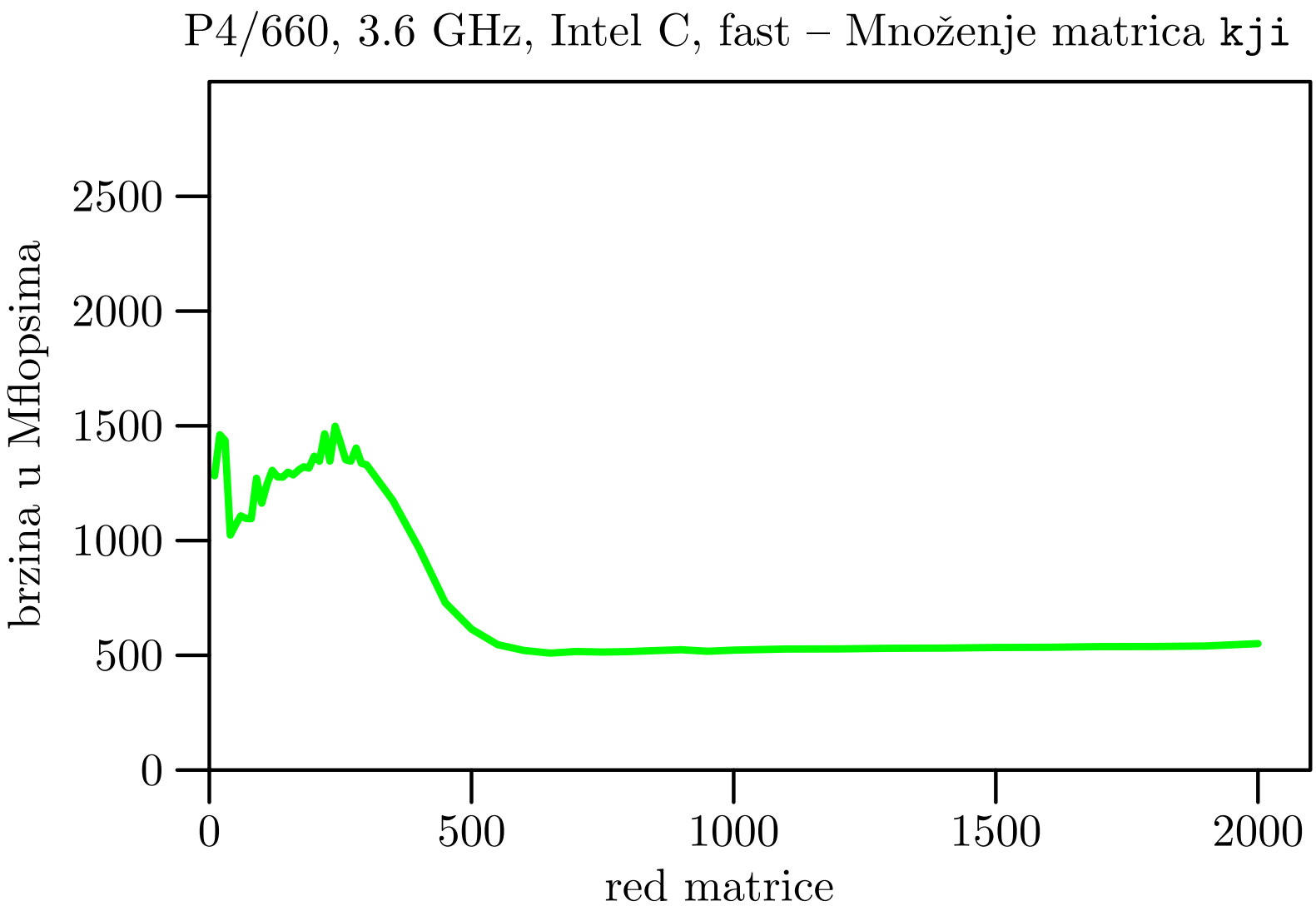
Alocirani vektor $M[i * 2000 + j]$ — `jki`



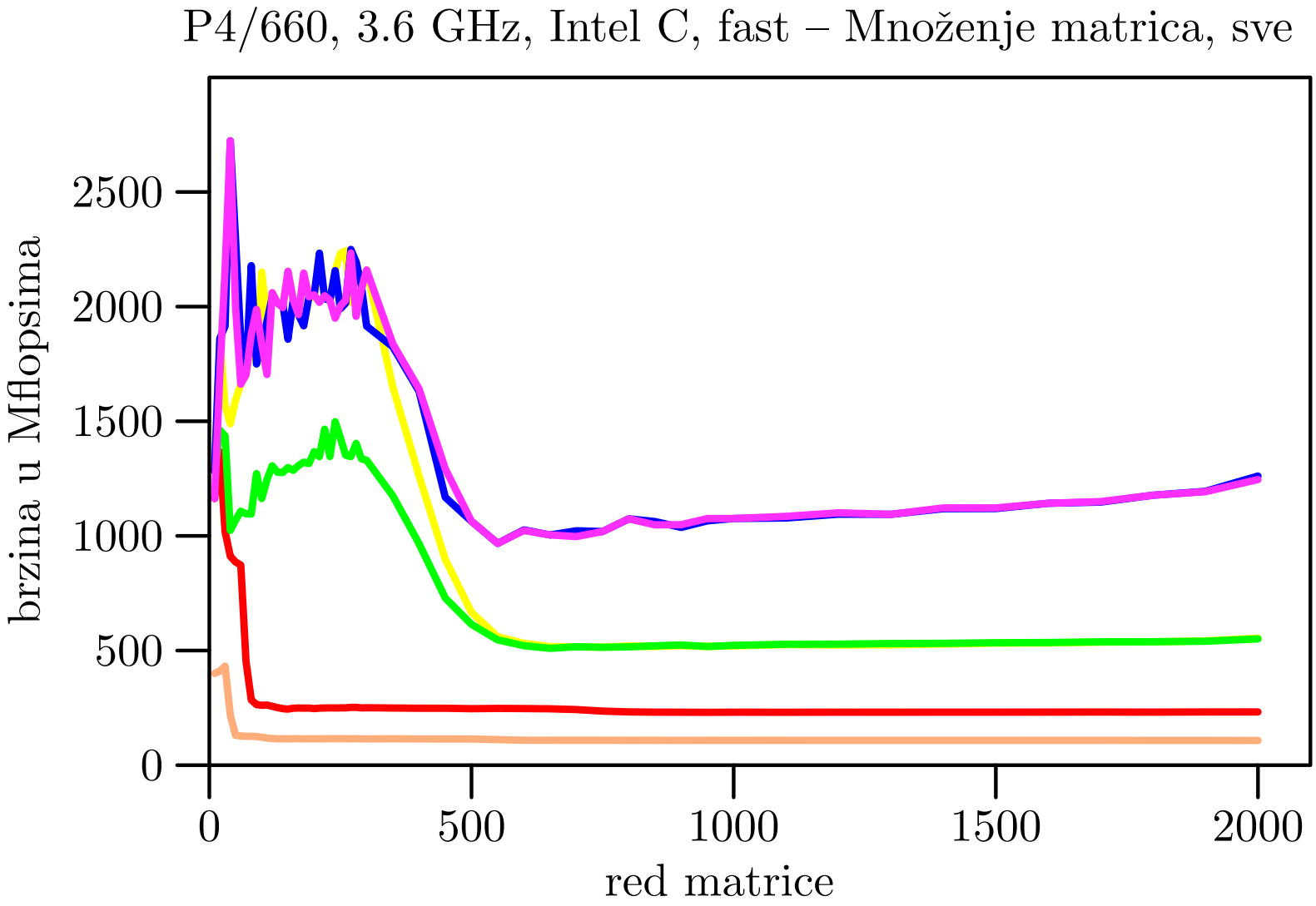
Alocirani vektor $M[i * 2000 + j]$ — `kij`



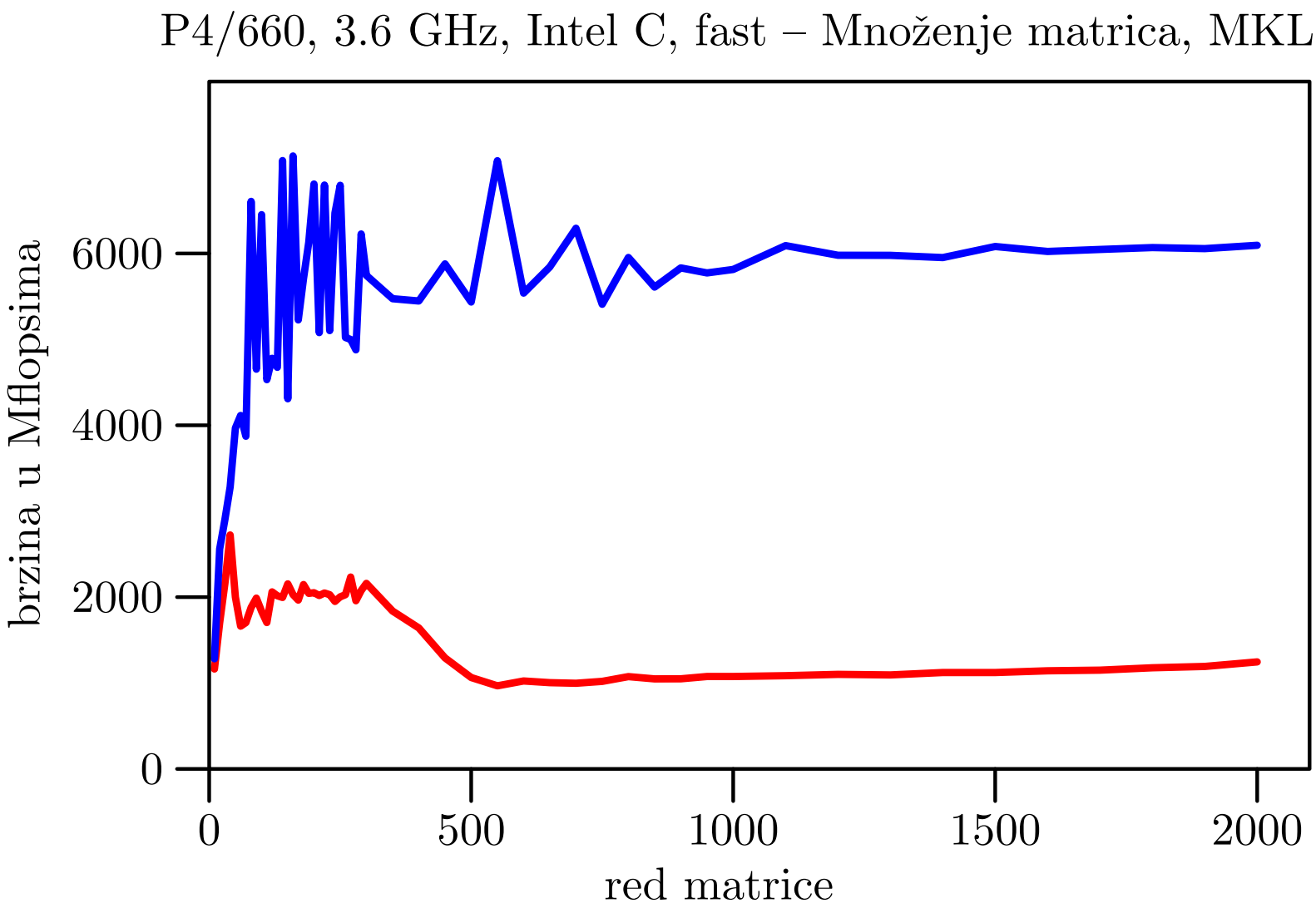
Alocirani vektor $M[i * 2000 + j]$ — `kji`



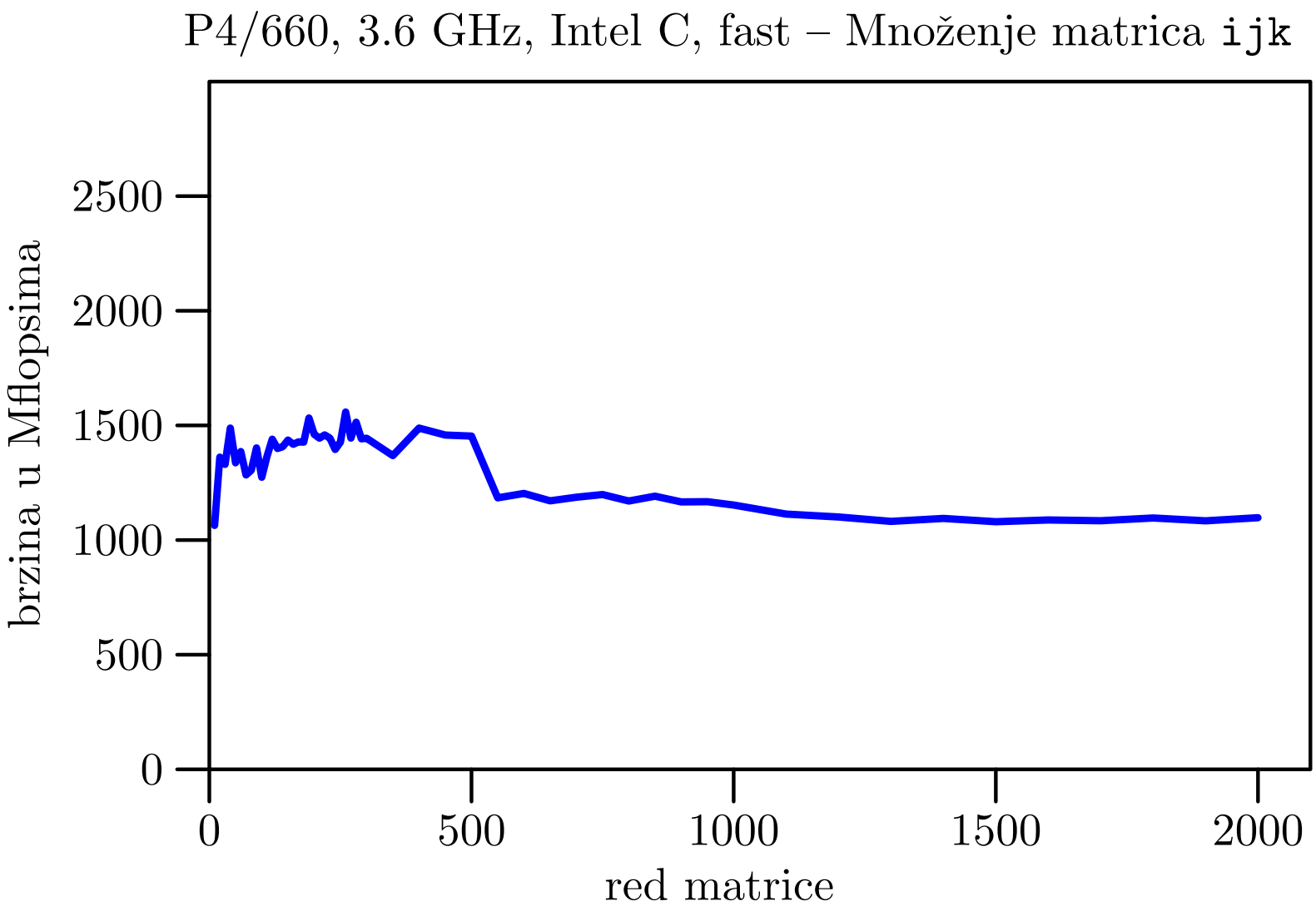
Alocirani vektor $M[i * 2000 + j]$ — sve zajedno



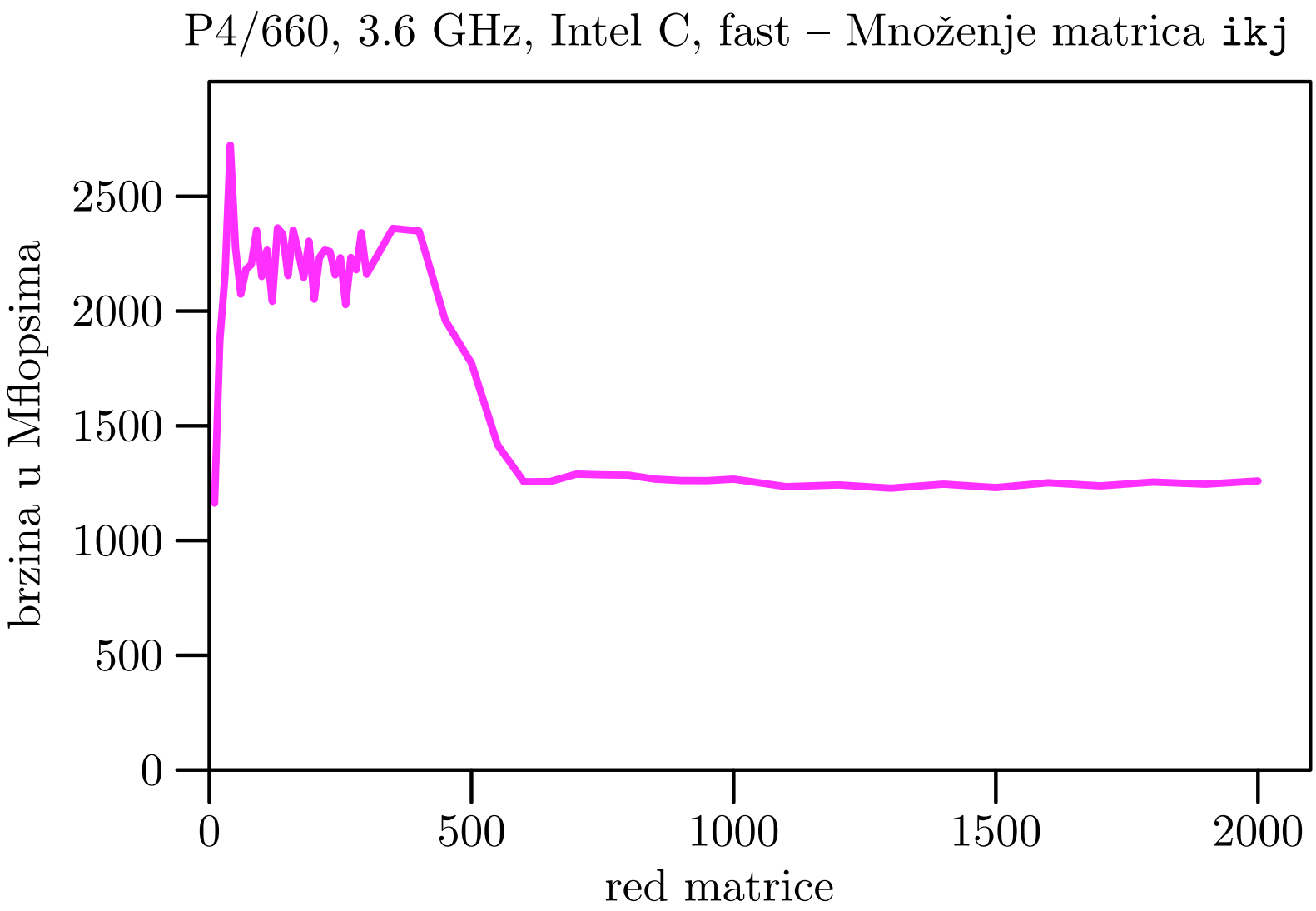
Alocirani vektor $M[i*2000+j]$ — najbrži i MKL



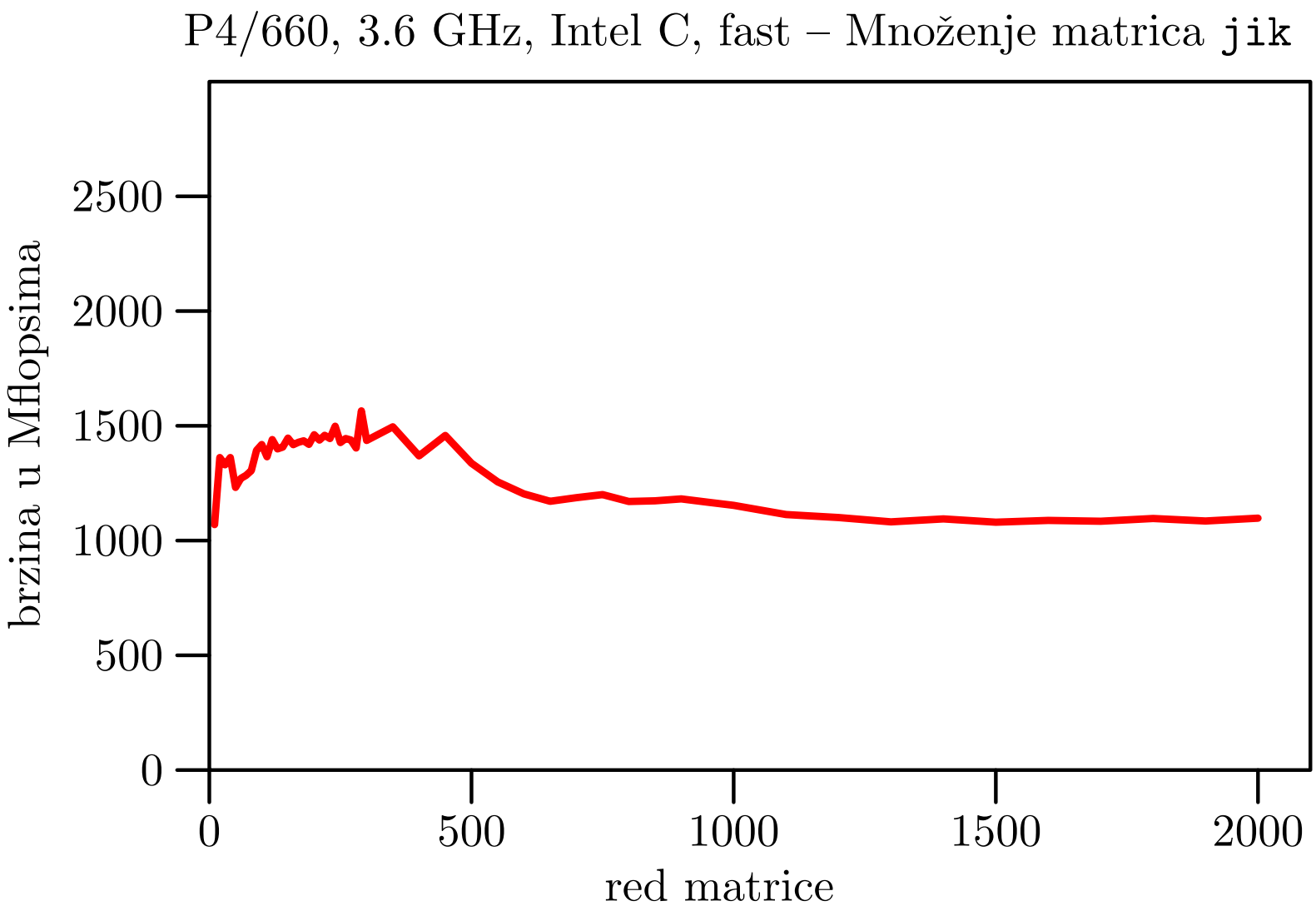
Alocirani vektor $M[i * n + j]$ — `ijk`



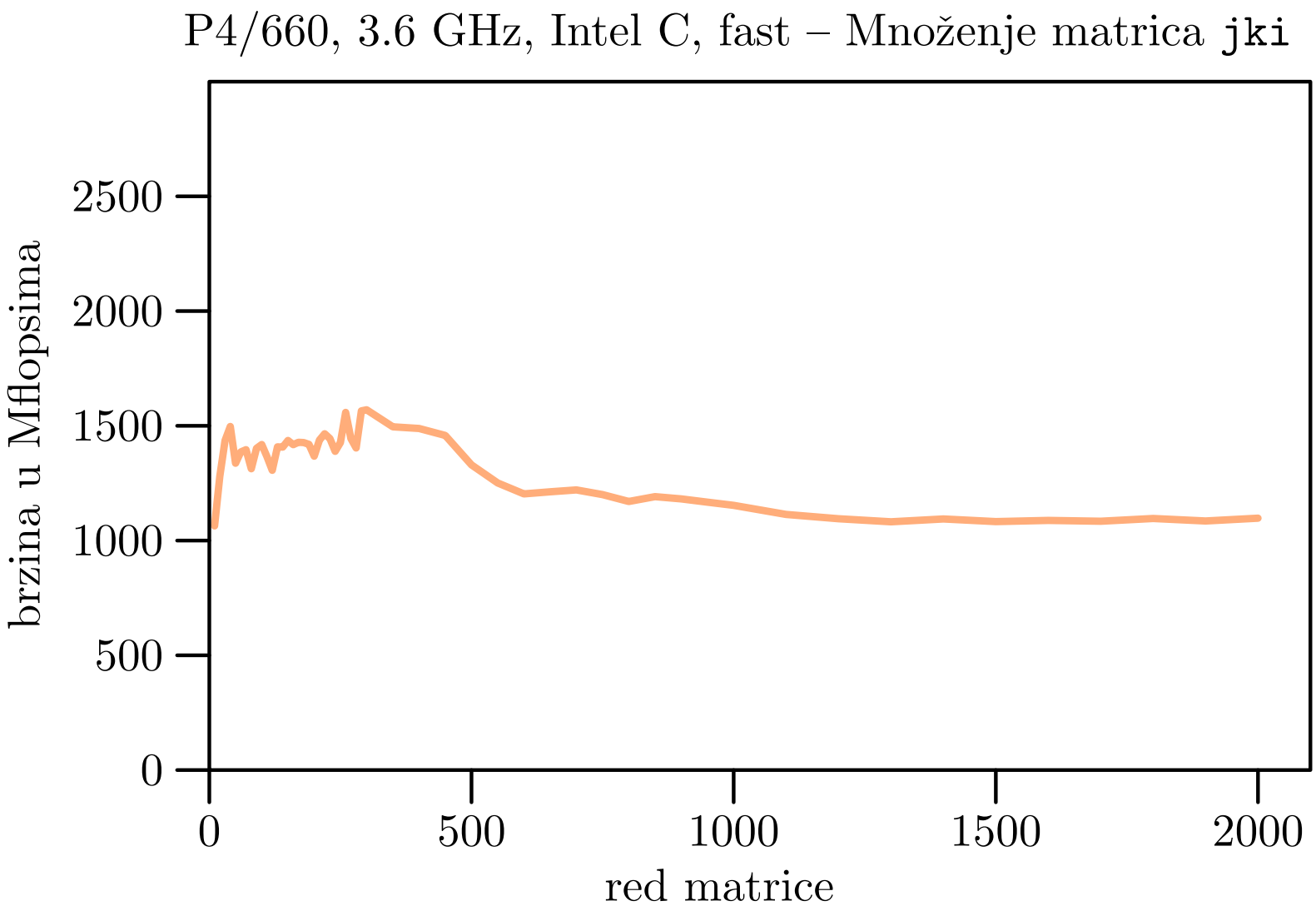
Alocirani vektor $M[i * n + j]$ — `ikj`



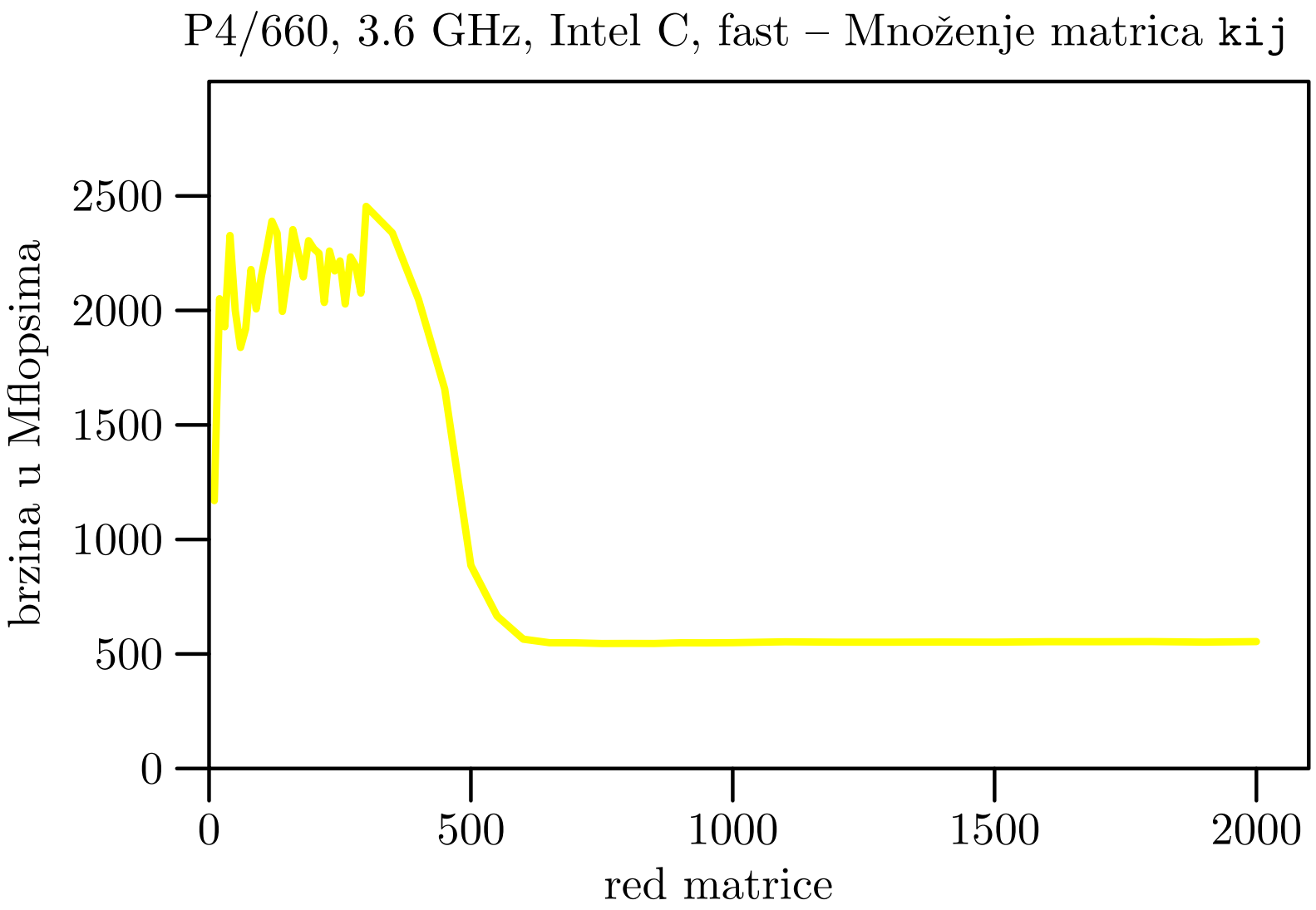
Alocirani vektor $M[i * n + j]$ — `jik`



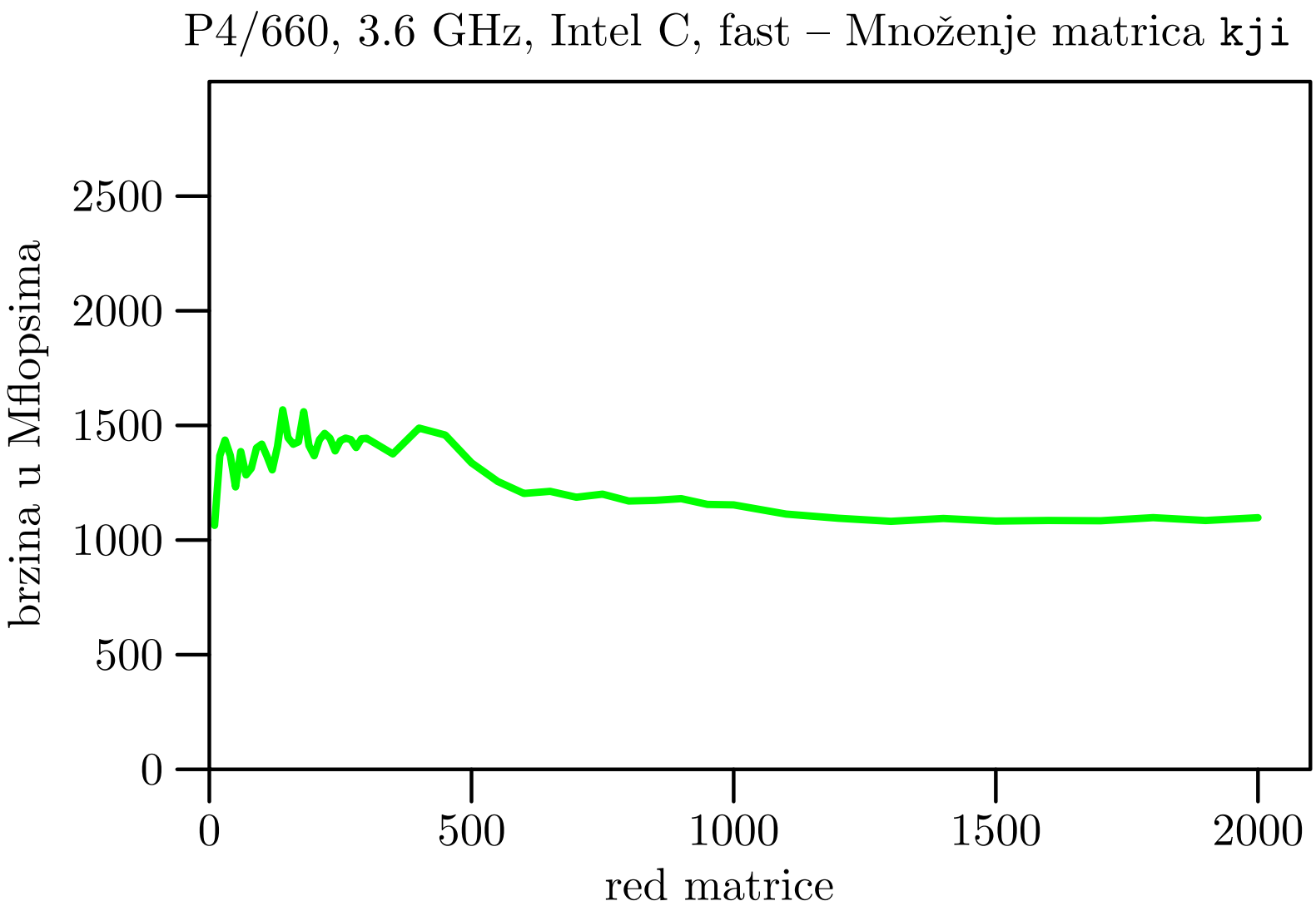
Alocirani vektor $M[i * n + j]$ — `jki`



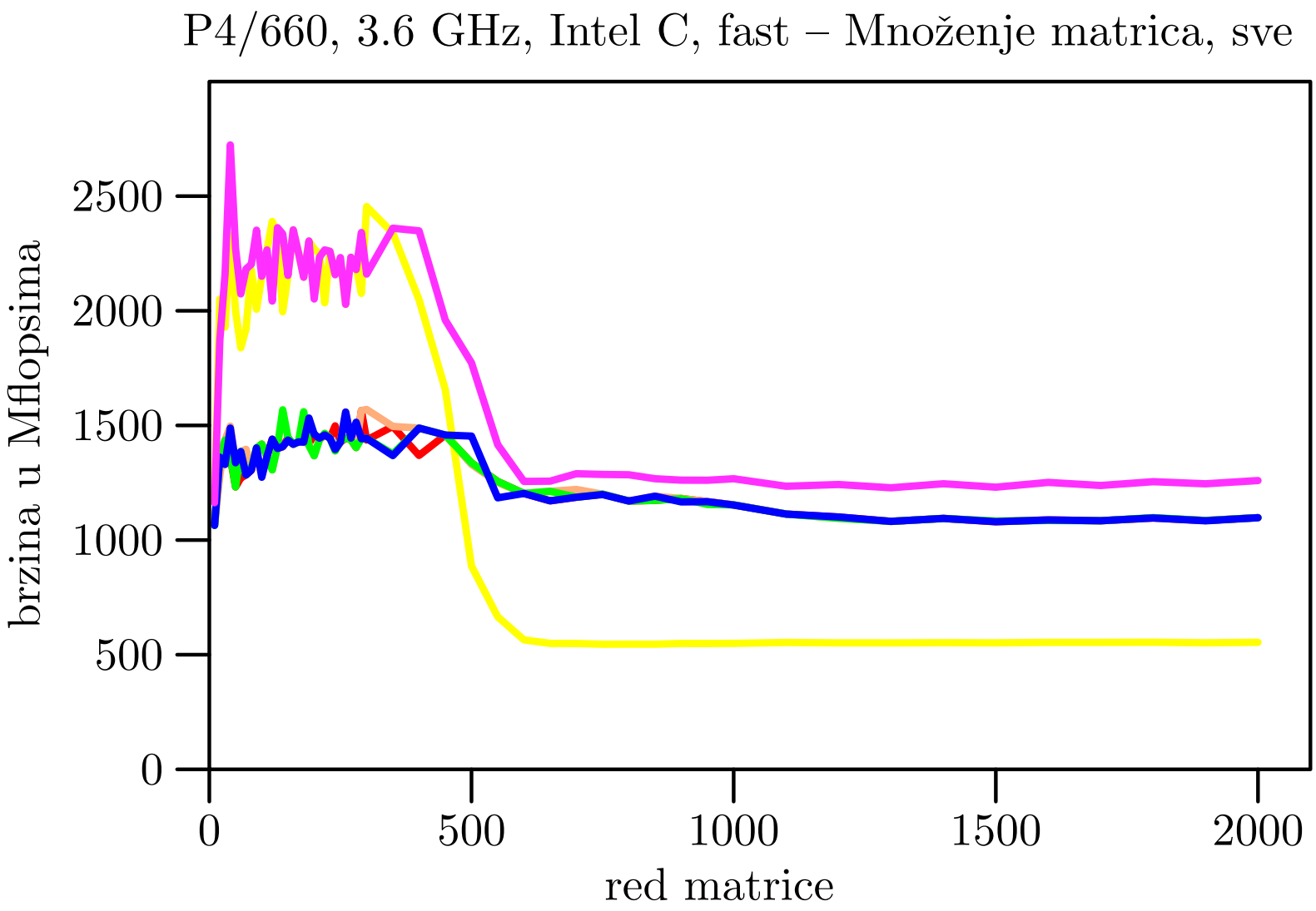
Alocirani vektor $M[i*n+j]$ — `kij`



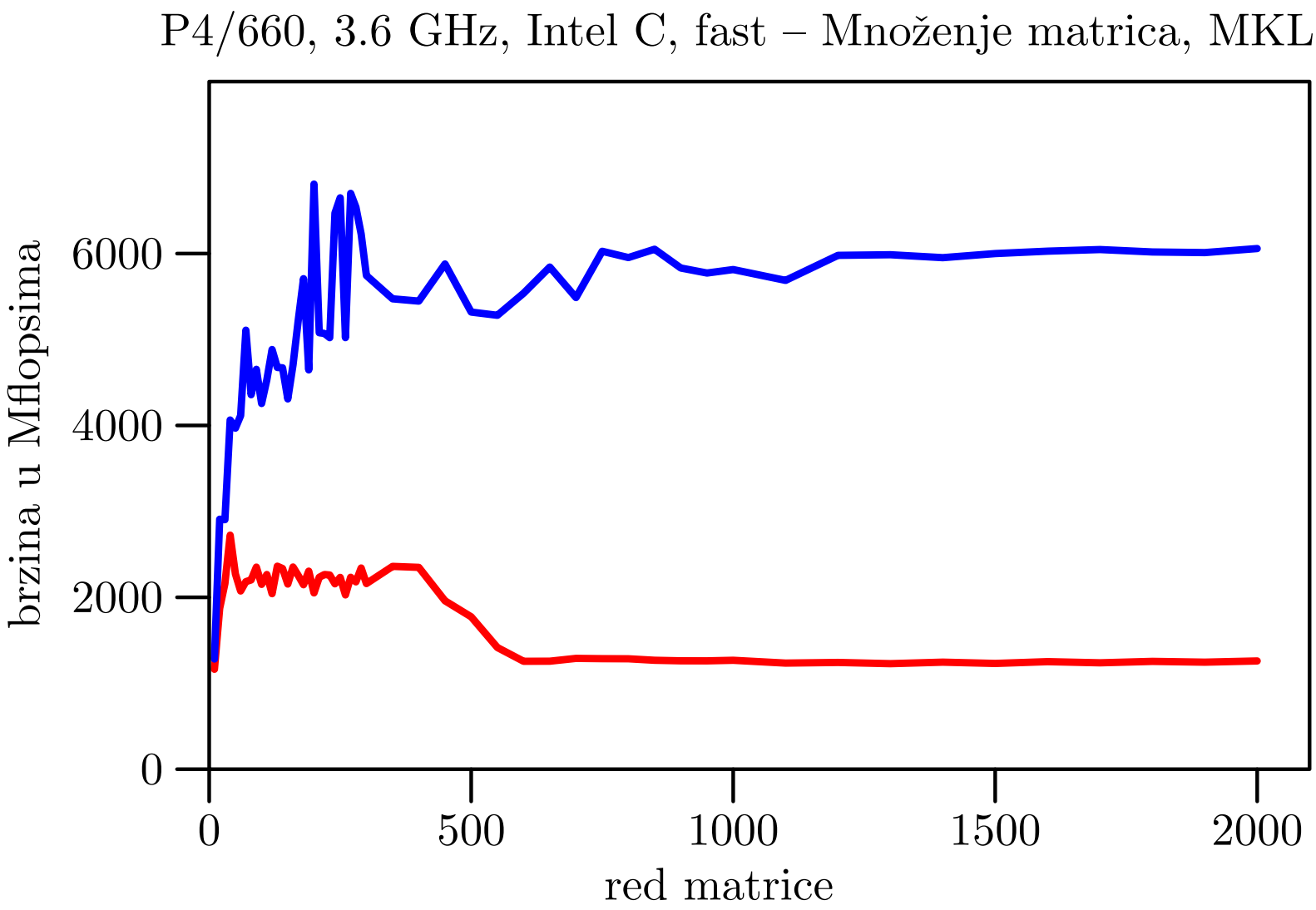
Alocirani vektor $M[i * n + j]$ — `kji`



Alocirani vektor $M[i * n + j]$ — sve zajedno



Alocirani vektor $M[i * n + j]$ — najbrži i MKL



Tablica brzina za velike n

Usporedba brzina — u Mflops, na P4/660:

- po petljama (uključivo i MKL),
- za sva četiri eksperimenta množenja matrica u C-u.

Petlja	normal matrice	fast matrice	fast "po recima"	fast "u bloku"
`ijk`	204.3	1270.4	1261.0	1097.5
`ikj`	814.6	1272.1	1245.7	1259.5
`jik`	226.4	1005.8	232.2	1097.5
`jki`	107.1	1006.9	107.8	1097.5
`kij`	521.1	553.5	554.1	554.1
`kji`	95.7	553.5	550.8	1097.5
MKL	5990.3	6058.3	6095.2	6058.3

Komentar rezultata

Kod množenja matrica, za razliku od zbrajanja,

- svaki ulazni podatak koristimo puno puta, (preciznije, točno n puta).

Zato brzina cache memorije može doći do izražaja, pa možemo dobiti

- bitno veće brzine nego kod zbrajanja.

Cache memorija je "glavni krivac" za:

- razlike u brzinama između raznih varijanti, i
- povećanu brzinu za male n-ove.

Ponavljanje eksperimenta ima neku ulogu samo za vrlo male redove n. Osim toga, za $n \geq 450$ nema ponavljanja.

Komentar rezultata (nastavak)

Brže su one varijante koje

- učestalije koriste iste podatke, dok su oni još u cacheu.

Konstruktivni dokaz: "Blokovskom" realizacijom algoritma

- za velike n možemo postići gotovo iste brzine kao i za male n (tj. spriječiti pad brzine).

Ovo, naravno, ide samo onda kad

- za velike n dobijemo pad brzine.

U protivnom, compiler se "već pobrinuo" da optimalno iskoristi cache.

Primjer za Intel C da to radi za normal, pa čak i za fast opciju.