Računarske vežbe
Letnji semestar 2023/24.

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
KATEDRA ZA PRIMENJENE RAČUNARSKE NAUKE

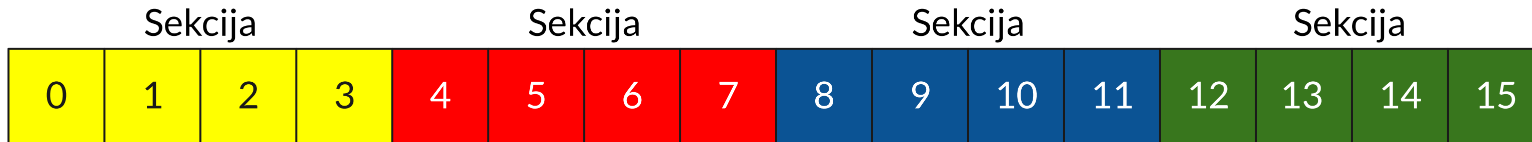
CUDA programiranje 3

Sadržaj

- Optimizacija pristupa globalnoj memoriji
  - Poravnat pristup
  - Transponovanje matrica
- Atomične operacije
- Dinamički alocirana memorija
- Računski šabloni — zadaci
  - Histogram

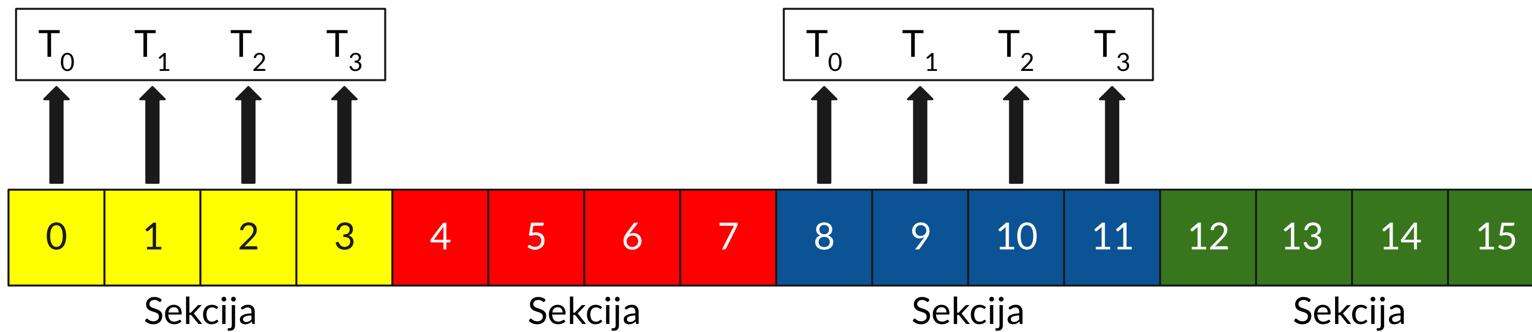
Optimizacija pristupa glavnoj memoriji

Pristup *CUDA* globalnoj memoriji



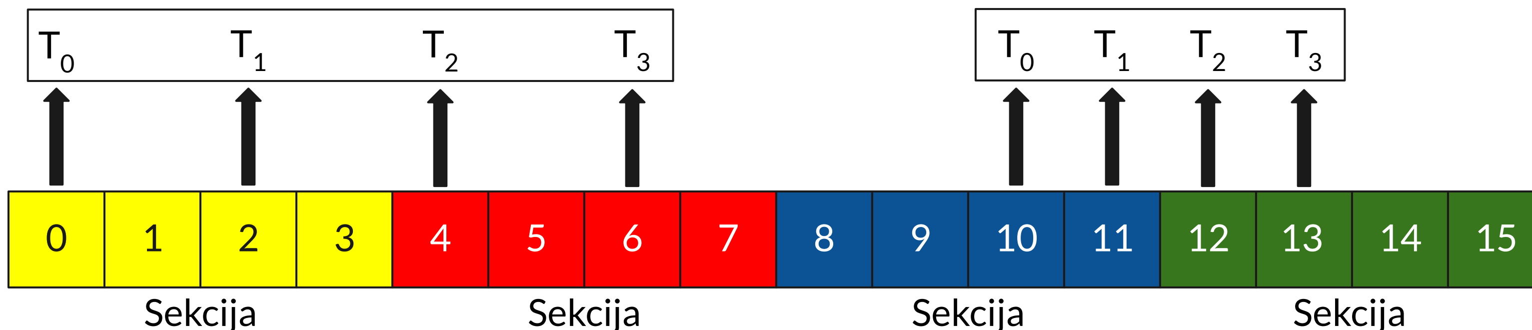
- *CUDA* globalna memorija podeljena je u sekcije.
- Pristup jednom podatku neće učitati samo taj podatak, već više podataka iz iste sekcije, u vidu višebajtne transakcije.
- Zarad ilustracije, možemo smatrati da jedna transakcije učitava sve podatke iz sekcije (iako su sekcije u uređajima najverovatnije veće od transakcije).
- Ilustracija prikazuje sekcije veličine 4 jedinice (npr. jedinice od po 4 bajta, iako današnji *CUDA* uređaji imaju veće sekcije).

Poravnat pristup globalnoj memoriji (engl. *Coalesced access*)



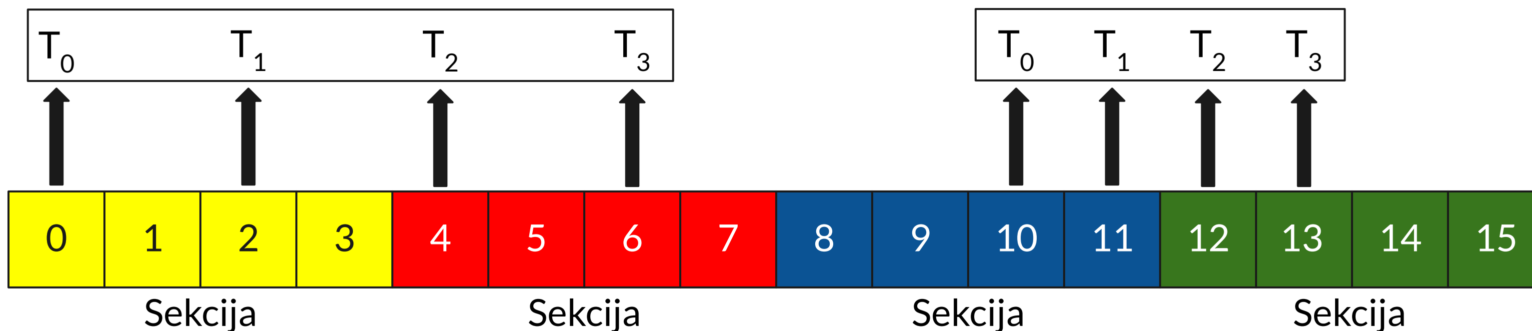
- Kada sve niti iz iste osnove (engl. *warp*) izvršavaju instrukciju učitavanja podataka, ako sve lokacije kojima se pristupa pripadaju istoj sekciji, pristup je **poravnat**.

Neporavnat pristup globalnoj memoriji



- Kada se lokacije kojima pristupaju niti iz iste osnove nalaze u različitim sekcijama, pristup **nije poravnat**.
- Potrebno je poslati više zahteva za učitavanje podataka, a niti neke od učitanih vrednosti neće ni koristiti.

Neporavnat pristup globalnoj memoriji



- Na gornjoj slici, kod prvog skupa niti problem je postojanje razmaka između uzastopnih niti (engl. *stride*).
- Kod drugog skupa niti problem je udaljenost od početka sekcije (engl. *offset*).

Analiza neporavnatog pristupa na primeru kernela

CUDA

Analiza neporavnatog pristupa na primeru kernela

- Primeri koda i grafikoni koji slede preuzeti su sa https://developer.nvidia.com/blog/how-access-global-memory-efficiently-cuda-c-kernels/ (pristupljeno 15.04.2024.)

CUDA

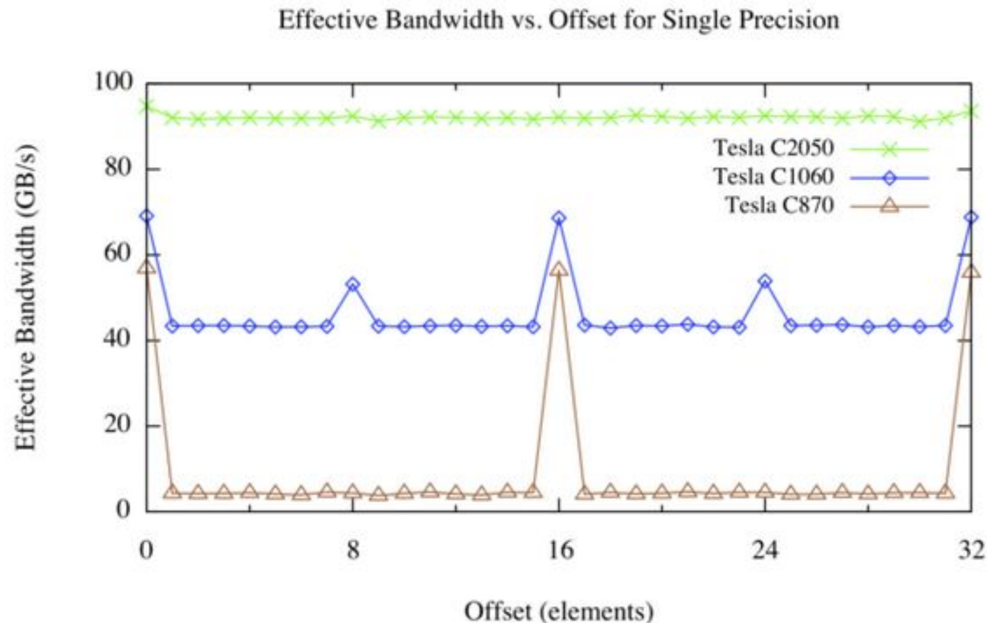
Analiza neporavnatog pristupa na primeru kernela

```
/**
 * kernel sa problemom offset-a
 * kernel se pokreće više puta, svaki put sa sve većim “s”
*/
__global__ void offset(T* a, int s)
{
	int i = blockDim.x * blockIdx.x + threadIdx.x + s;
	a[i] = a[i] + 1;
}
```

Analiza neporavnatog pristupa na primeru kernela

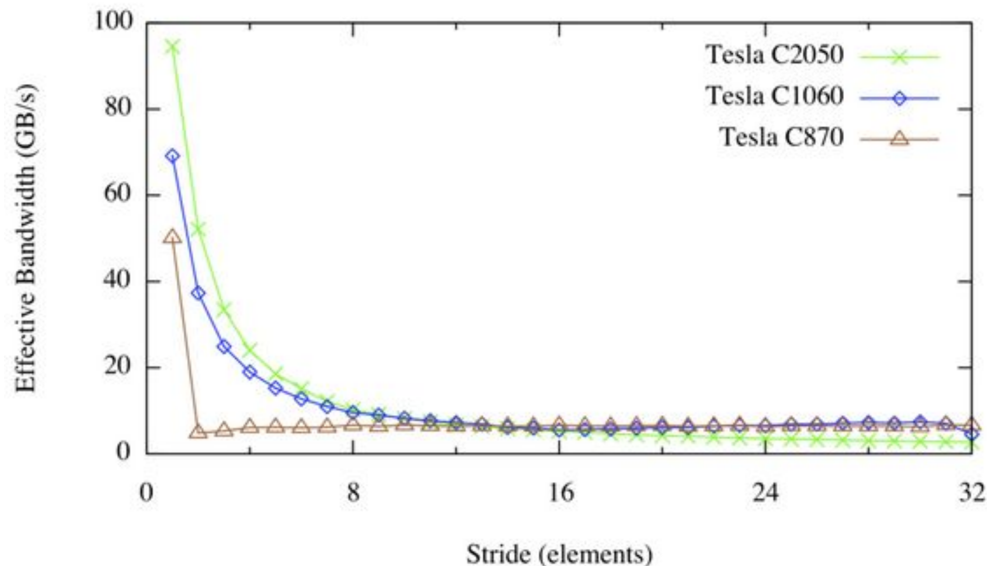
```
/**
 * kernel sa problemom stride-a
 * kernel se pokreće više puta, svaki put sa sve većim “s”
*/
__global__ void stride(T* a, int s)
{
        int i = (blockDim.x * blockIdx.x + threadIdx.x) * s;
        a[i] = a[i] + 1;
}
```

Analiza neporavnatog pristupa na primeru kernela



- Na starijim arhitekturama (C870), ako pristup ne kreće od početka 16-bajtne sekcije, svaka nit učitava zasebno 32 bajta.
- Na nešto novijim (C1060) postoji sličan problem, ali je pristup optimizovaniji.
- Najnovije arhitekture (C2050) imaju mogućnost keširanja 128-bajtnih linija, pa je problem gotovo neprimetan.

Analiza neporavnatog pristupa na primeru kernela



- Na starijim arhitekturama (C1060), ukoliko postoji makar jedan “preskočen” indeks između uzastopnih niti, performanse drastično opadaju.
- Na novijim arhitekturama (C1060, C2050) pad performansi je nešto blaži, ali postojanje razmaka između niti u svakom slučaju predstavlja problem.
- Ne postoji način da se pristupi podacima koji su fizički mnogo udaljeni sabiju u jedan pristup.

- *Stride* je mnogo veći problem od *offset*-a za današnje arhitekture.
- Zato je potrebno veću pažnju obratiti na rešavanje problema *stride*-a.
- Pristup globalnoj memoriji za niti iz iste osnove je poravnat, ako je indeks pristupa nizu oblika:
  - `arr[(izraz koji ne zavisi od threadIdx.x) + threadIdx.x]`
- Niti iz višedimenzionalnih blokova se ređaju sukcesivno po *x* komponenti indeksa.
- Ponekad je problem takav da, ukoliko se radi isključivo sa globalnom memorijom, veliki *stride* je neizbežan.
  - Sa ovakim problemima se možemo boriti deljenom memorijom.

Optimizacija kernela za transponovanje matrica

Transponovanje matrica — poravnata čitanja

```
__global__ void transpose_read_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
	int  j = threadIdx.x + blockIdx.x * blockDim.x;
	int i = threadIdx.y + blockIdx.y * blockDim.y;
	int from_idx = i * N + j;
	if ((i < M) && (j < N)) {
		int to_idx = j * M + i;
		output_matrix[to_idx] = input_matrix[from_idx];
	}
}
```

Uzastopne niti imaće uzastupne vrednosti ovog indeksa, jer varira najpre po x komponenti indeksa

Transponovanje matrica — poravnata čitanja

```
__global__ void transpose_read_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
	int  j = threadIdx.x + blockIdx.x * blockDim.x;
	int i = threadIdx.y + blockIdx.y * blockDim.y;
	int from_idx = i * N + j;
	if ((i < M) && (j < N)) {
		int to_idx = j * M + i;
		output_matrix[to_idx] = input_matrix[from_idx];
	}
}
```

Vrednosti ovog indeksa za uzastopne niti razlikovaće se za *M*

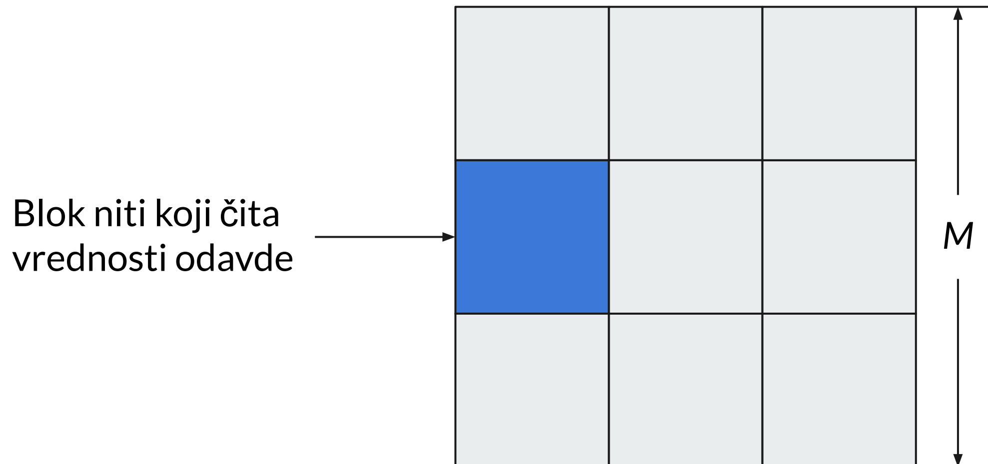
Transponovanje matrica — poravnata čitanja

```
__global__ void transpose_read_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
        int  j = threadIdx.x + blockIdx.x * blockDim.x;
        int i = threadIdx.y + blockIdx.y * blockDim.y;
        int from_idx = i * N + j;
        if ((i < M) && (j < N)) {
                int to_idx = j * M + i;
                output_matrix[to_idx] = input_matrix[from_idx];
        }
}
```

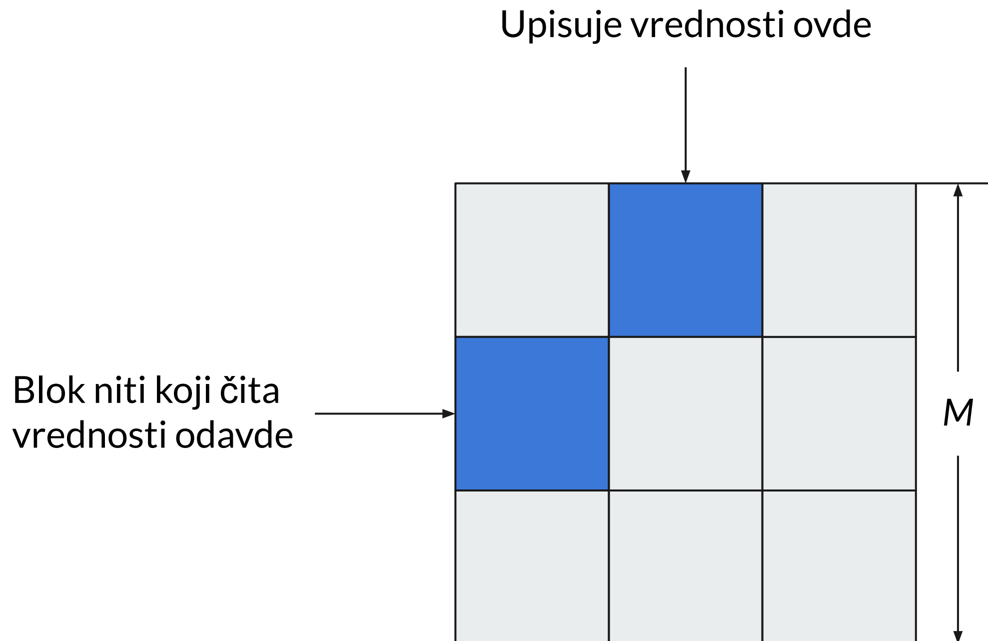
Uzastopne niti pristupaju elementima ovog niza koji su udaljeni za M mesta

Uzastopne niti pristupaju uzastopnim elementima ovog niza

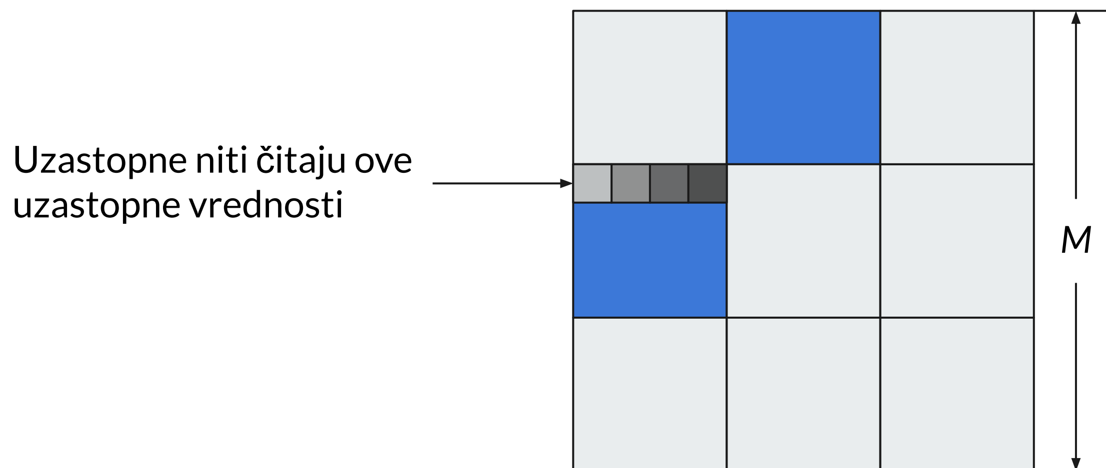
Transponovanje matrica — poravnata čitanja



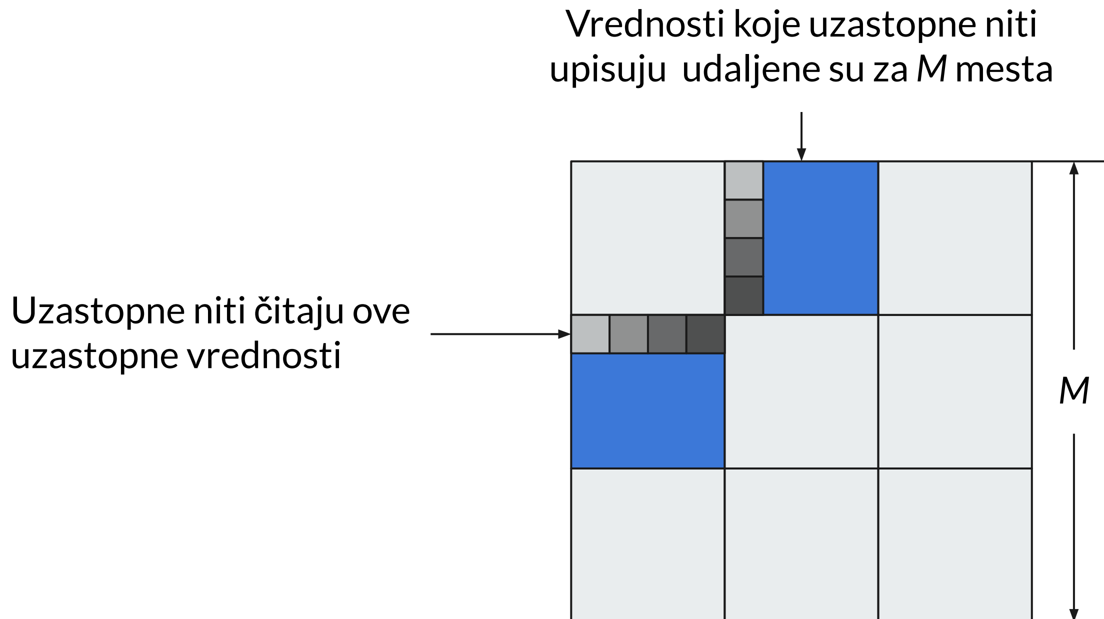
Transponovanje matrica — poravnata čitanja



Transponovanje matrica — poravnata čitanja



Transponovanje matrica — poravnata čitanja



Transponovanje matrica — poravnati upisi

```
__global__ void transpose_write_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
        int  j = threadIdx.x + blockIdx.x * blockDim.x;
        int i = threadIdx.y + blockIdx.y * blockDim.y;
        int to_idx = i * M + j;
        if ((i < N) && (j < M)) {
                int from_idx = j * N + i;
                output_matrix[to_idx] = input_matrix[from_idx];
        }
}
```

Uzastopne niti imaće uzastupne vrednosti ovog indeksa, jer varira najpre po x komponenti indeksa

CUDA

Transponovanje matrica — poravnati upisi

```
__global__ void transpose_write_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
	int  j = threadIdx.x + blockIdx.x * blockDim.x;
	int i = threadIdx.y + blockIdx.y * blockDim.y;
	int to_idx = i * M + j;
	if ((i < N) && (j < M)) {
		int from_idx = j * N + i;
		output_matrix[to_idx] = input_matrix[from_idx];
	}
}
```

Vrednosti ovog indeksa za uzastopne niti razlikovaće se za *N*

Transponovanje matrica — poravnati upisi

```
__global__ void transpose_write_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N)
{
        int  j = threadIdx.x + blockIdx.x * blockDim.x;
        int i = threadIdx.y + blockIdx.y * blockDim.y;
        int to_idx = i * M + j;
        if ((i < N) && (j < M)) {
                int from_idx = j * N + i;
                output_matrix[to_idx] = input_matrix[from_idx];
        }
}
```

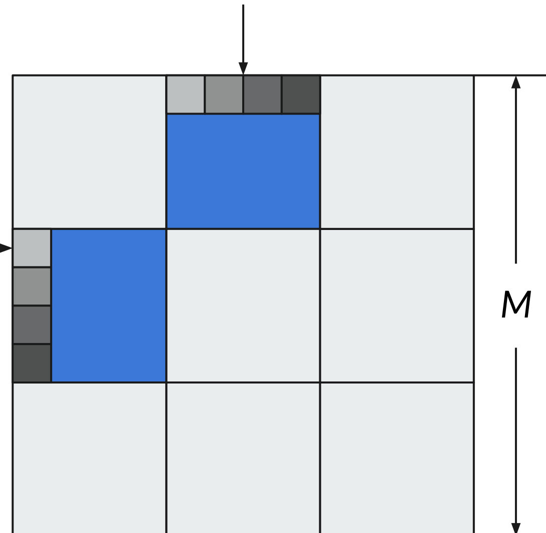
Uzastopne niti pristupaju uzastopnim elementima ovog niza

Uzastopne niti pristupaju elementima ovog niza koji su udaljeni za N mesta

Transponovanje matrica — poravnati upisi

Uzastopne niti upisuju
vrednosti na uzastopna mesta

Uzastopne niti čitaju vrednosti
udaljene za *M* mesta



CUDA

Transponovanje matrica — poravnati upisi i čitanja

```
__global__ void transpose_read_write_coalesced(T* output_matrix,  T const* input_matrix, size_t M, size_t N) {
        __shared__ T buffer[BLOCK_SIZE][BLOCK_SIZE];
        int j = threadIdx.x + blockIdx.x * blockDim.x,   i = threadIdx.y + blockIdx.y * blockDim.y;
        int matrix_from_idx = i * N + j;

        if ((i < M) && (j < N)) {
                buffer[threadIdx.x][threadIdx.y] = input_matrix[matrix_from_idx];
        }
        __syncthreads();

        int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

        if ((transp_i < N) && (transp_j < M)) {
                int to_idx = transp_i * M + transp_j;
                output_matrix[to_idx] = buffer[threadIdx.y][threadIdx.x];
        }
}
```

Ovaj pristup je poravnat jer uzastopne niti imaju uzastopne vrednosti indeksa.

CUDA

Transponovanje matrica — poravnati upisi i čitanja

```
__global__ void transpose_read_write_coalesced(T* output_matrix,  T const* input_matrix, size_t M, size_t N) {
        __shared__ T buffer[BLOCK_SIZE][BLOCK_SIZE];
        int j = threadIdx.x + blockIdx.x * blockDim.x,  i = threadIdx.y + blockIdx.y * blockDim.y;
        int matrix_from_idx = i * N + j;

        if ((i < M) && (j < N)) {
                buffer[threadIdx.x][threadIdx.y] = input_matrix[matrix_from_idx];
        }
        __syncthreads();

        int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

        if ((transp_i < N) && (transp_j < M)) {
                int to_idx = transp_i * M + transp_j;
                output_matrix[to_idx] = buffer[threadIdx.y][threadIdx.x];
        }
}
```

Zamenom komponente indeksa bloka vrši se transponovanje pozicije čitavog bloka. Sada se *i* određuje na osnovu *x* komponente, a *j* na osnovu *y* komponente.

Transponovanje matrica — poravnati upisi i čitanja

```
__global__ void transpose_read_write_coalesced(T* output_matrix, T const* input_matrix, size_t M, size_t N) {
        __shared__ T buffer[BLOCK_SIZE][BLOCK_SIZE];
        int j = threadIdx.x + blockIdx.x * blockDim.x,   i = threadIdx.y + blockIdx.y * blockDim.y;
        int matrix_from_idx = i * N + j;

        if ((i < M) && (j < N)) {
                buffer[threadIdx.x][threadIdx.y] = input_matrix[matrix_from_idx];
        }
        __syncthreads();

        int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

        if ((transp_i < N) && (transp_j < M)) {
                int to_idx = transp_i * M + transp_j;
                output_matrix[to_idx] = buffer[threadIdx.y][threadIdx.x];
        }
}
```

Komponenta indeksa niti ostaje ista, pa uzastopne niti i dalje rade sa uzastopnim elementima.

Transponovanje matrica — poravnati upisi i čitanja

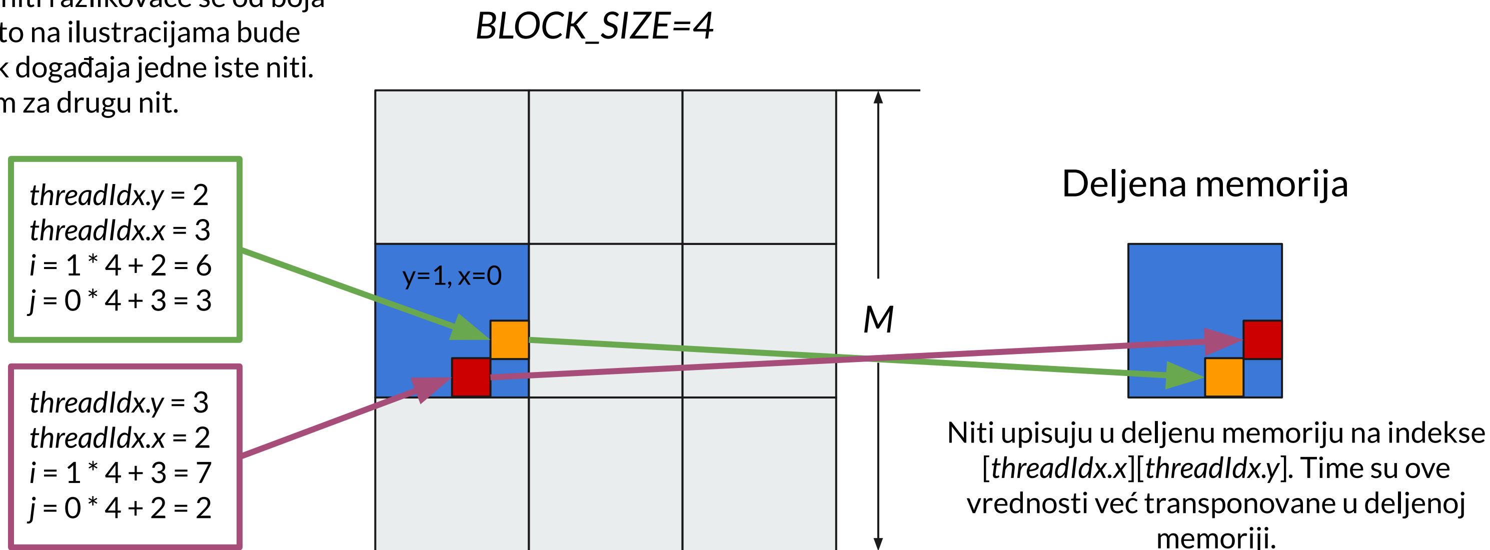
```
__global__ void transpose_read_write_coalesced(T* output_matrix,  T const* input_matrix, size_t M, size_t N) {
        __shared__ T buffer[BLOCK_SIZE][BLOCK_SIZE];
        int j = threadIdx.x + blockIdx.x * blockDim.x,   i = threadIdx.y + blockIdx.y * blockDim.y;
        int matrix_from_idx = i * N + j;

        if ((i < M) && (j < N)) {
                buffer[threadIdx.x][threadIdx.y] = input_matrix[matrix_from_idx];
        }
        __syncthreads();

        int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

        if ((transp_i < N) && (transp_j < M)) {
                int to_idx = transp_i * M + transp_j;
                output_matrix[to_idx] = buffer[threadIdx.y][threadIdx.x];
        }
}
```

Zamenom indeksa deljene memorije vrši se transponovanje unutar prethodno transponovanog bloka.

Transponovanje matrica — poravnati upisi i čitanja

- Posmatrajmo čitav proces iz ugla dve niti iz bloka *blockIdx.y=1*, *blockIdx.x=0* (naznačeno i unutar samog bloka na slici).

Zarad jasnoće, boje samih niti razlikovaće se od boja vrednosti iz matrice. Sve što na ilustracijama bude zelene boje predstavlja tok događaja jedne iste niti. Isti je slučaj i sa ljubičastom za drugu nit.



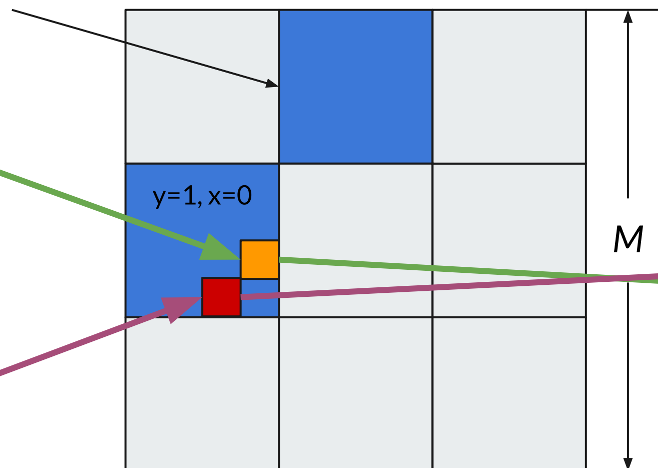
Transponovanje matrica — poravnati upisi i čitanja

int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

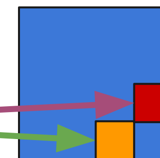
Označen deo promenljive *transp_i* određuje vrstu bloka koji gađamo i jednaka je *blockIdx.x*, odnosno 0.

threadIdx.y = 2
threadIdx.x = 3
i = 1 * 4 + 2 = 6
j = 0 * 4 + 3 = 3

threadIdx.y = 3
threadIdx.x = 2
i = 1 * 4 + 3 = 7
j = 0 * 4 + 2 = 2



Deljena memorija



Transponovanje matrica — poravnati upisi i čitanja

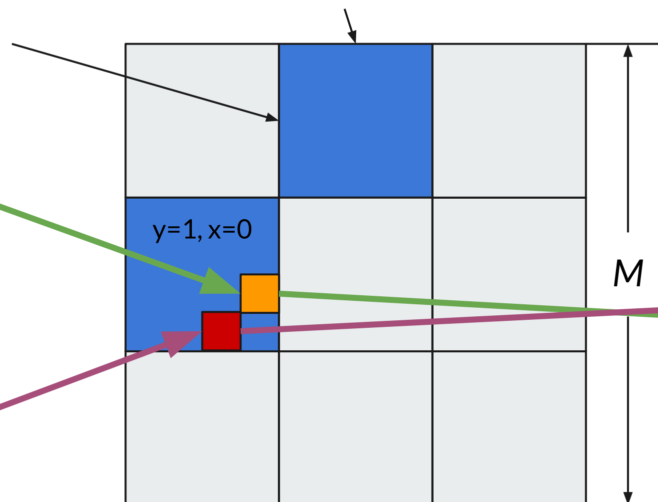
int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

Označeni deo promenljive *transp_j* određuje kolonu bloka koji gađamo i jednaka je *blockIdx.y*, odnsno 1.

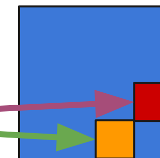
Označen deo promenljive *transp_i* određuje vrstu bloka koji gađamo i jednaka je *blockIdx.x*, odnosno 0.

threadIdx.y = 2
threadIdx.x = 3
i = 1 * 4 + 2 = 6
j = 0 * 4 + 3 = 3

threadIdx.y = 3
threadIdx.x = 2
i = 1 * 4 + 3 = 7
j = 0 * 4 + 2 = 2



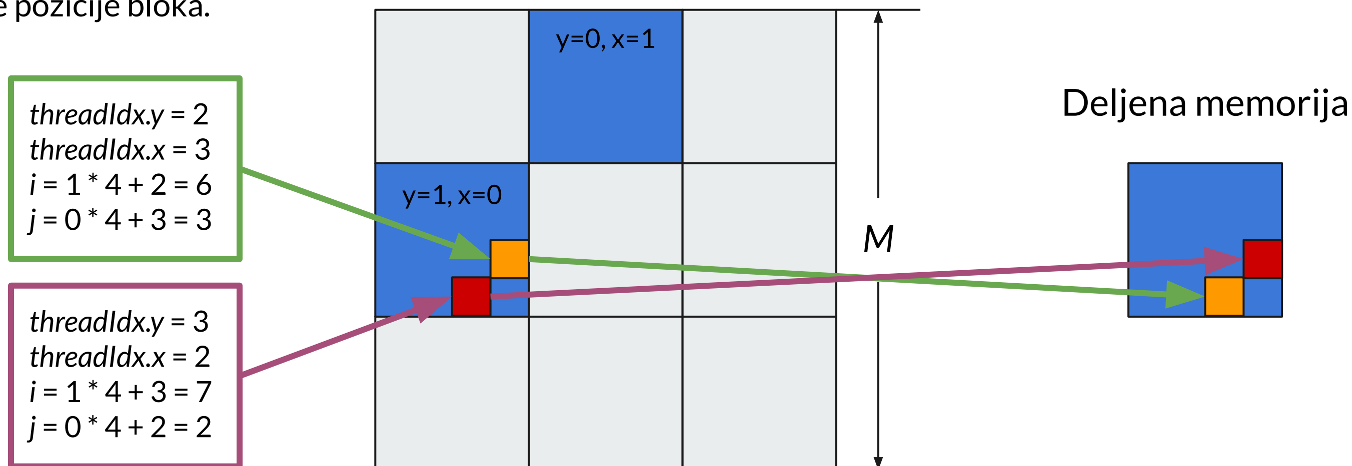
Deljena memorija



Transponovanje matrica — poravnati upisi i čitanja

int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

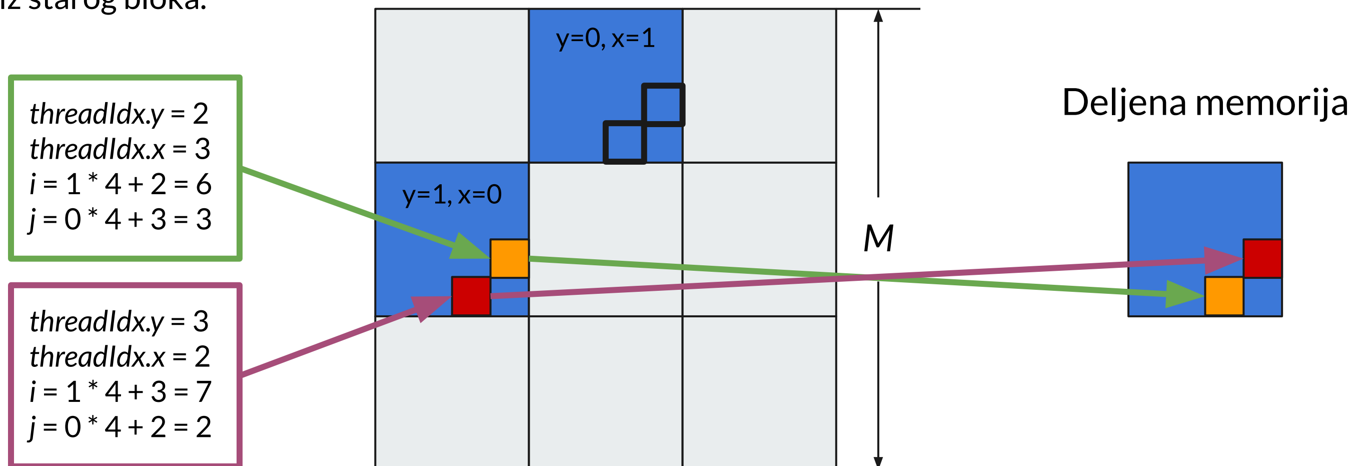
Blok u koji naša nit upisuje je na indeksima [0, 1], a čitala je iz bloka sa indeksima [1, 0]. Time dobijamo transponovanje pozicije bloka.



Transponovanje matrica — poravnati upisi i čitanja

int transp_j = threadIdx.x + blockIdx.y * blockDim.y, transp_i = threadIdx.y + blockIdx.x * blockDim.x;

Indeksi elemenata na koje niti upisuju vrednost unutar novog bloka su isti kao i indeksi sa kojih čitaju iz starog bloka.

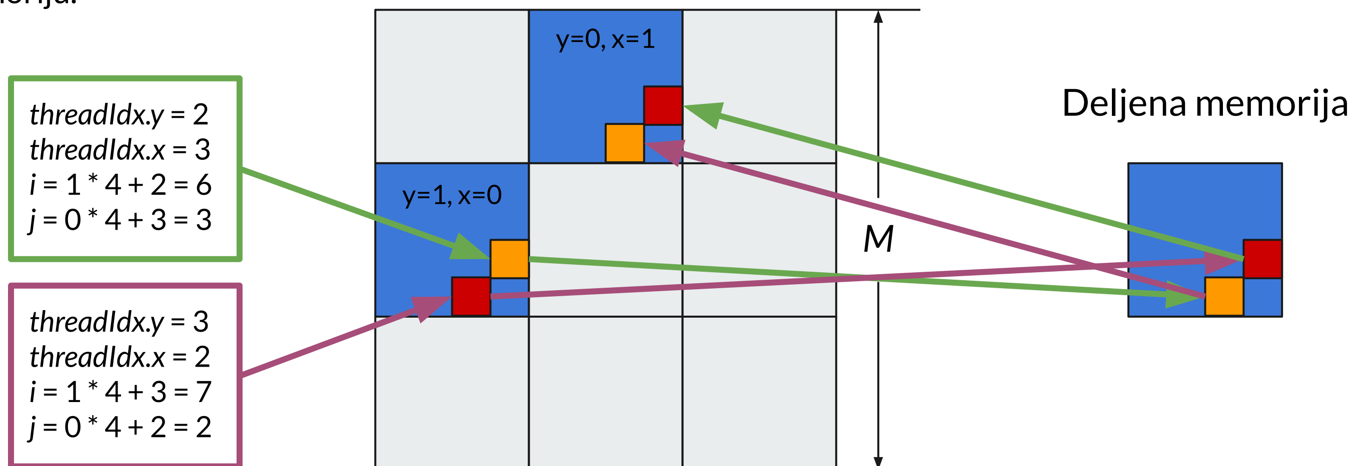


Transponovanje matrica — poravnati upisi i čitanja

buffer[threadIdx.y][threadIdx.x];

Nit čita sa indeksa iz deljene memorije koji su transponovani u odnosu na indekse na koje je upisala u deljenu memoriju.

Primetite da nit koja je iz glavne matrice iz početnog bloka čitala sa indeksa 2, 3 upisuje na iste te indekse novog bloka. Transponovanje indeksa u deljenoj memoriji dovodi do toga da ta nit upiše narandžastu vrednost u deljenu memoriju, a pročita iz nje crvenu. Isti je slučaj i sa drugom niti, kao i sa svima ostalima.



Zadatak

- Analizirati pristup globalnoj memoriji u kernelima oba zadatka za množenje matrica sa prošlog časa.
  - Pristup kojim matricama (A, B, M, N) je poravnat?
  - Zašto?

Atomične funkcije

Atomične funkcije

- Podržane nad 32-bitnim i 64-bitnim vrednostima u deljenoj i globalnoj memoriji
- Primeri:
  - atomicAdd( )
  - atomicSub( )
  - atomicAnd( )
  - ...
- https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#atomic-functions

Dinamički alocirana deljena memorija

Dinamički alocirana deljena memorija

- Ključna reč *extern* ispred *__shared__* dekoratora.
- Prazne uglaste zagrade nakon poziva polja.
- Zahteva treći parametar u konfiguraciji pri pozivu kernela — veličinu polja u bajtovima

extern __shared__ unsigned int arr[]; —unutar kernela
*myKernel<<<gridDim, blokcDim, n * sizeof(unsigned int)>>>(...)* — na domaćinu

Zadaci — histogram

Zadaci

- Pogledati zadatke sa histogramima sa ACS-a
- Uraditi ih koristeći postavke sa *Google Colab*-a

Obrada nizova većih od broja niti

- Ukoliko je neka dimenzija niza veća od ukupnog broja niti koje su na raspolaganju po toj dimenziji, neke niti će obraditi više elemenata.
- Obrada se vrši u petlji, a indeks niti se ažurira sve dok ne izađe iz opsega niza.

```
for (int i = globalThreadId; i < arrDim; i += blockDim.x * gridDim.x) {...}
```