Porazdeljeni sistemi

7. Računanje na grafičnih procesnih enotah II

Predavatelj: izr. prof. Uroš Lotrič
Asistent: Davor Sluga

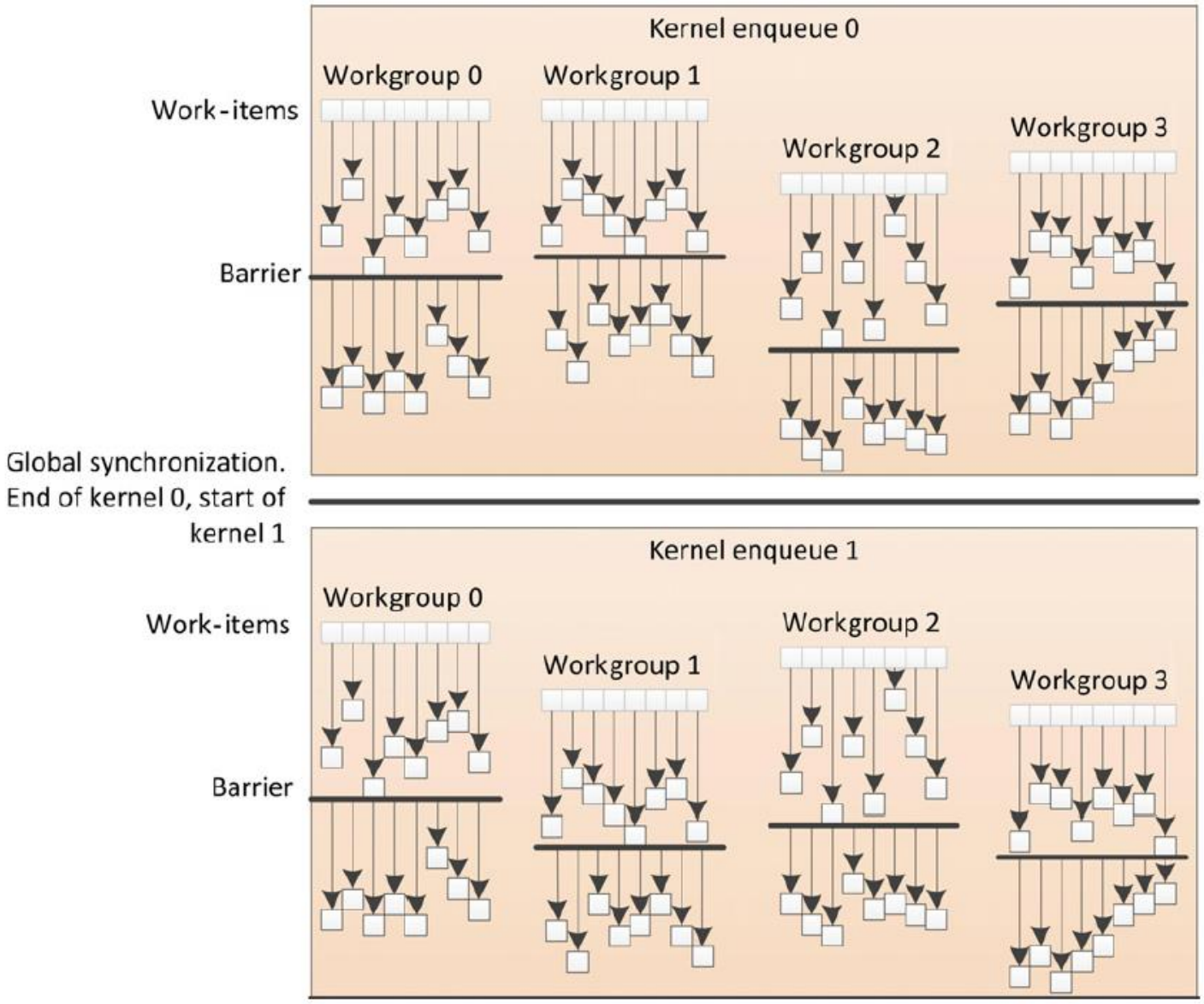
OpenCL aplikacija: povzetek

- Izberemo platformo
- Izberemo naprave
- Definiramo kontekst
- Pripravimo ukazno vrsto za napravo
- Alociramo pomnilnik na napravi
- Skopiramo podatke iz gostitelja na napravo
- Pripravimo program in ga prevedemo
- Pripravimo ščepec z argumenti in ga zaženemo
- Skopiramo rezultate iz naprave na gostitelja
- Pospravimo za sabo

Organizacija niti

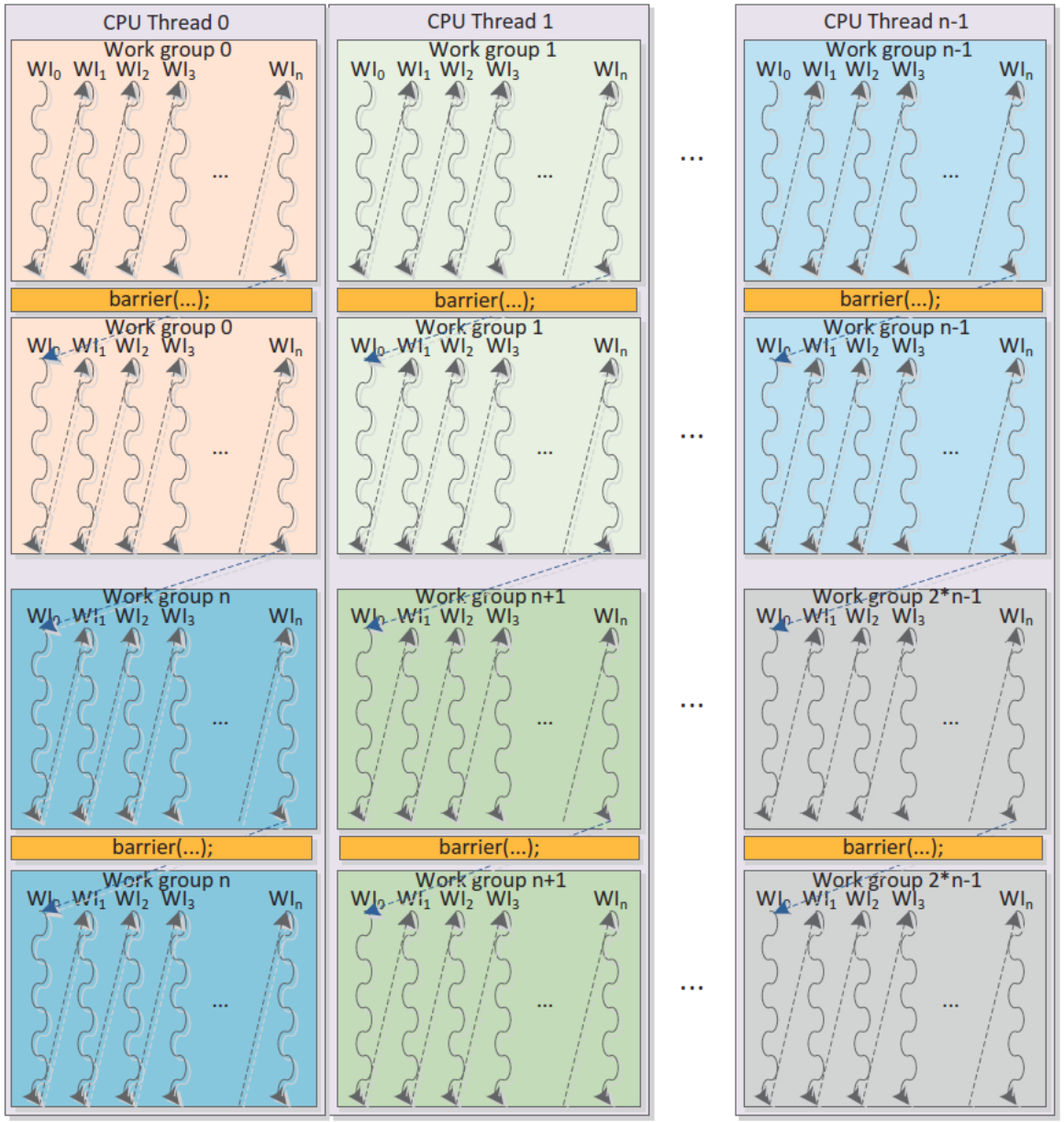
- Programer se sam odloči, koliko niti bo uporabil za reševanje določenega problema – programer sam definira izvajalno okolje
- Dobro je, da imamo niti čim več, saj tako
  - olajšamo delo razvrščevalniku, ki ima za izvajanje vedno na voljo več snopov niti
  - pohitrimo izvajanje programa
- Omejitve:
  - število niti v bloku
  - število blokov v mreži
  - Število niti na multiprocesorju

Izvajanje niti



Izvajanje niti

- Niti na CPU



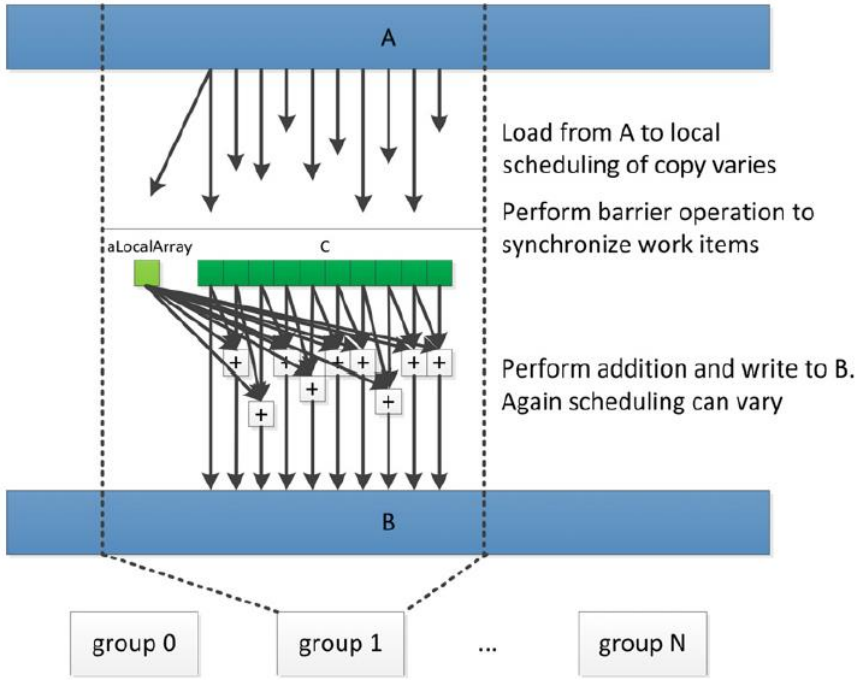
Komunikacija med nitmi v bloku

- En blok niti se izvaja na eni računski enoti (Compute unit)
  - Niti v istem bloku torej lahko komunicirajo preko spremenljivk v lokalnem pomnilniku
  - Vse niti lahko pišejo v spremenljivke in berejo iz spremenljivk v lokalnem pomnilniku
  - Niti na računski enoti lahko sinhroniziramo
    - Prepreko postavimo z ukazom barrier(), ki mu navedemo ali naj se sinhronizacija izvede po pisanju
      - v lokalni pomnilnik: barrier(CLK_LOCAL_MEM_FENCE) ali
      - globalni pomnilnik: barrier(CLK_GLOBAL_MEM_FENCE)
    - Sinhronizacija na lokalnem pomnilniku je hitrejša
    - Paziti moramo, da prepreko postavimo za vse niti v bloku

Komunikacija med nitmi v bloku

- Primer

```
__kernel void localAccess(
    __global float* A,
    __global float* B,
    __local float* C )
{
    __local float aLocalArray[1];

    if( get_local_id(0) == 0 ) {
       aLocalArray[0] = A[0];
    }
    C[get_local_id(0)] = A[get_global_id(0)];

    barrier( CLK_LOCAL_MEM_FENCE );

    float neighborSum = C[get_local_id(0)] + aLocalArray[0];

    if( get_local_id(0) > 0 )
       neighborSum = neighborSum + C[get_local_id(0)-1];

    B[get_global_id(0)] = neighborSum;
}
```



OpenCL

- Primeri:
  - Računanje skalarnega produkta
  - Množenje matrik

Zgled: skalarni produkt

- Pri skalarnem produktu moramo najprej zmnožiti vse istoležne elemente v vektorju, nato pa zmnožke sešteti
  - Množenje istoležnih elementov lahko izvedemo na enak način, kot smo to spoznali pri seštevanju vektorjev:
    - Vsaka nit zmnoži po en element vektorja
    - Če je elementov veliko, lahko tudi več
  - Kako izvesti seštevanje?
    - Lahko na CPE, vendar jo je škoda obremenjevati za take stvari.
    - Lahko pa na GPE. Kako?

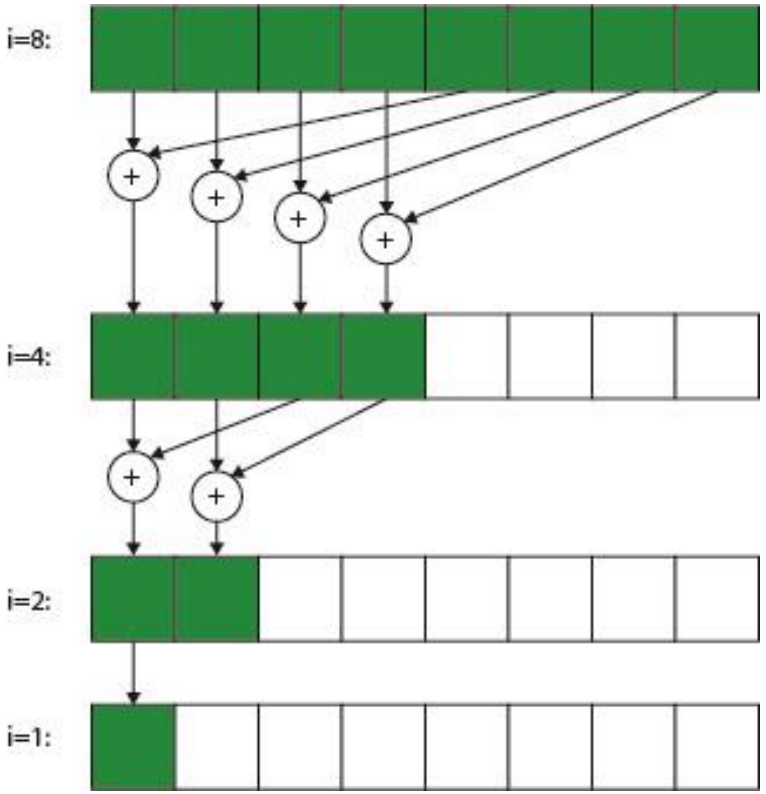
Zgled: skalarni produkt

- Če želimo zmnožke sešteti na napravi GPE, mora posamezna nit znati prebrati zmnožke, ki so jih izračunale druge niti → niti morajo med seboj komunicirati
- Za medsebojno komunikacijo uporabimo lokalni pomnilnik
  - Niti v istem bloku lahko dostopajo do podatkov v lokalnem pomnilniku
- IDEJA:
  - Naj vsaka nit računa in sešteva svoje delne produkte. Ko zaključi, naj svoj produkt vpiše v element polja, ki je vidno vsem nitim
  - To polje mora imeti toliko elementov, kolikor je niti v bloku
  - To polje se mora nahajati v lokalnem pomnilniku na računski enoti, da bodo lahko vse niti v bloku dostopale do njega

Zgled: skalarni produkt

- Seštevanje elementov polja v lokalnem pomnilniku
  - To bi lahko počela ena sama nit, ostale niti bi čakale
  - Se da bolje?

- Redukcija
  - Kaj, če vsaka nit sešteje svoj in še en delni rezultat?
  - V tem primeru za seštevanje *N* delnih produktov potrebujemo $\log_2(N)$ korakov



Zgled: skalarni produkt

- Redukcija
  - Po vsaki iteraciji se nam število produktov, ki jih moramo sešteti, razpolovi
  - V vsaki iteraciji se razpolovi število niti, ki sešteva delne produkte
  - Po vsaki iteraciji potrebujemo pregrado!

- Pregrada
  - Niti se izvajajo neodvisno ena od druge, zato ne moremo vedeti v kakšnem vrstnem redu se bodo delni skalarni produkti vpisovali v skupni pomnilnik
  - Delne skalarne produkte lahko seštejemo šele potem, ko vse niti zaključijo računanje
  - To dosežemo s pregrado
    - Sinhronizira vse niti v bloku
    - Ustavi izvajanje vseh niti dokler vse ne pridejo do pregrade!!!

Zgled: skalarni produkt

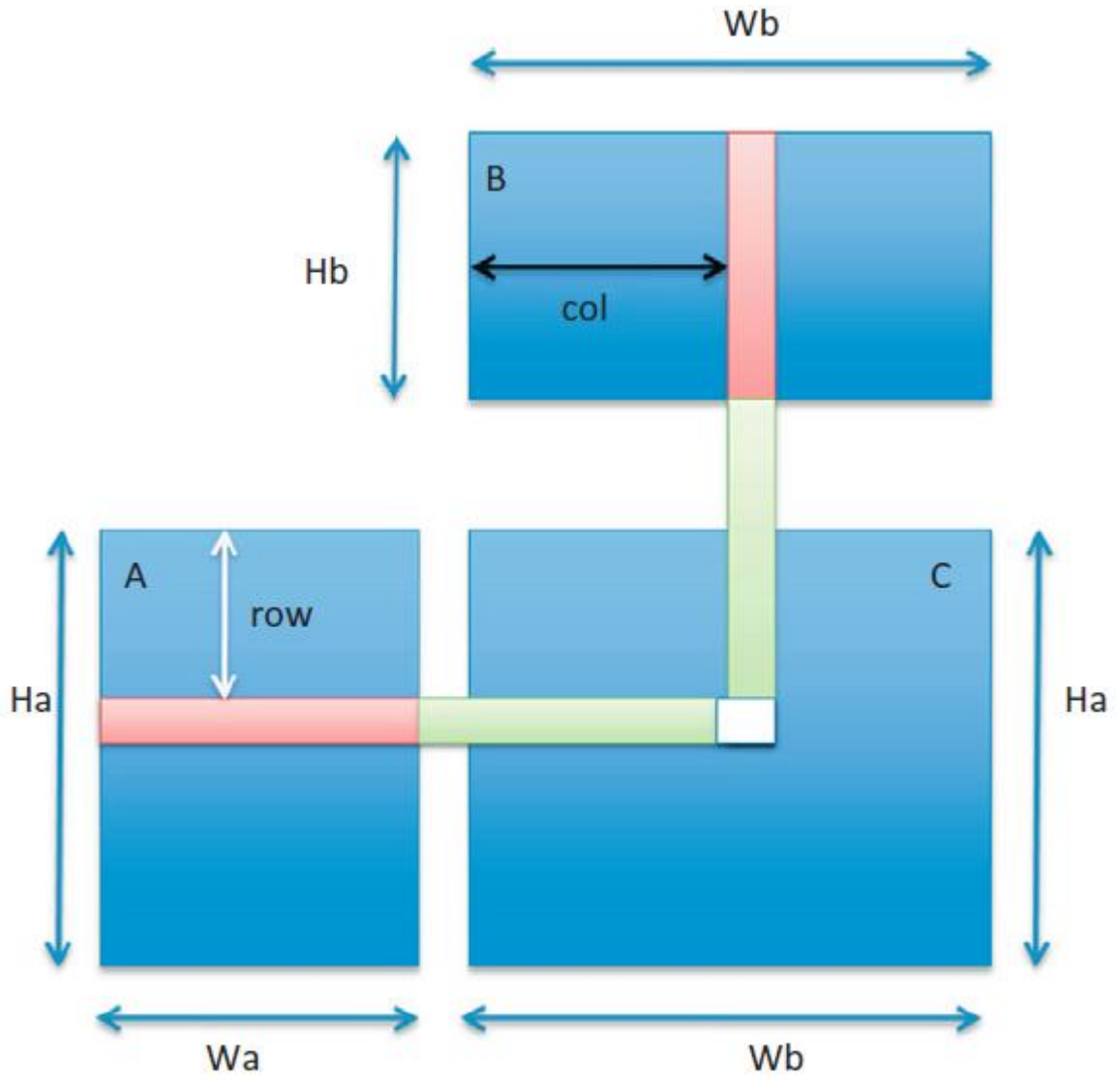
- Redukcija: nevarnost
  - Ali bi lahko pred pregrado počakale samo tiste niti, ki v posamezni iteraciji seštevajo delne produkte?
  - Ne bo šlo, saj barrier() vstavi pregrado za vse niti!
    - Niti, ki ne seštevajo, nikoli ne bi prišle do pregrade, zato bi se izvajanje na napravi ustavilo brez sporočila o napaki
    - Niti, ki seštevajo, bi prišle do pregrade in tam čakale ostale niti!!!

```
while( idxStep > 0 )
{
    if(lid < idxStep)
    {
        partial[lid] += partial[lid+idxStep];
        barrier( CLK_LOCAL_MEM_FENCE );
    }

    idxStep /= 2;
}
```

Zgled: skalarni produkt

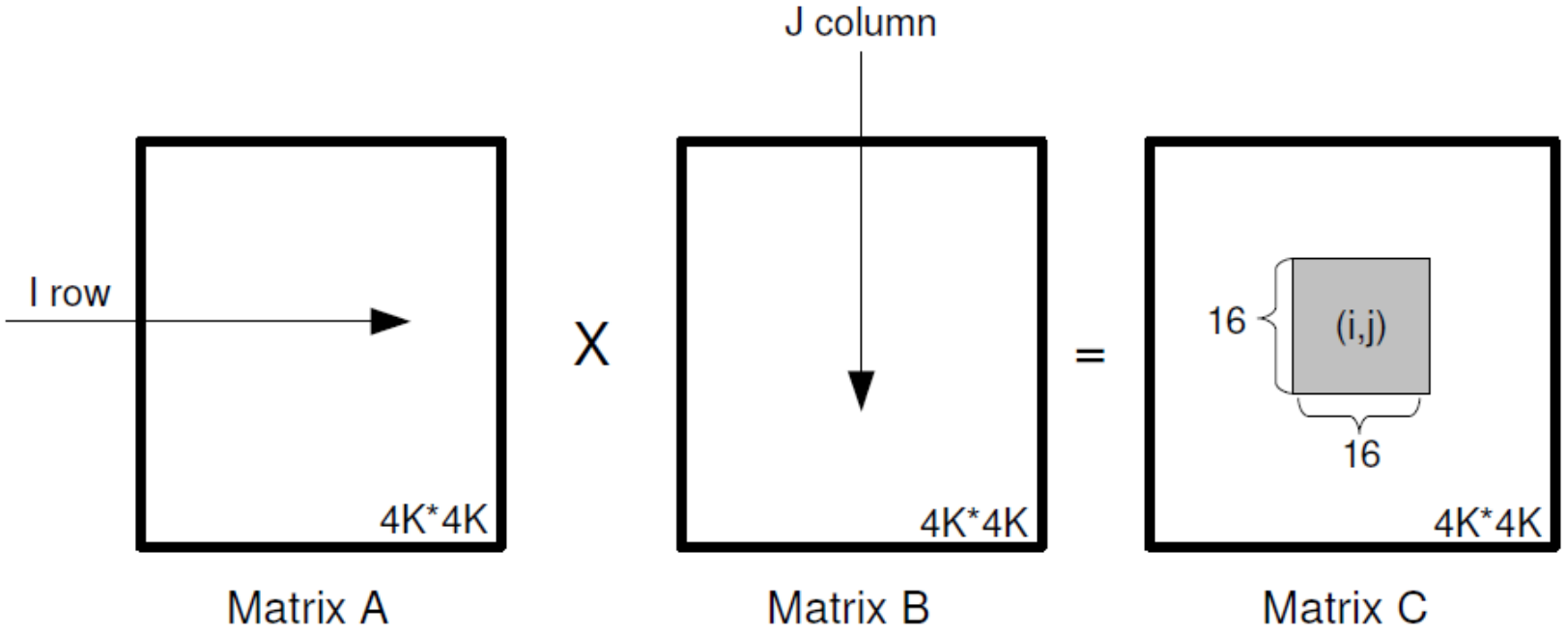
- Redukcija
  - Niti iz različnih blokov med seboj ne morejo komunicirati preko skupnega pomnilnika
  - Komunikacija med nitmi iz različnih blokov je možna le preko globalnega pomnilnika na napravi
  - Zato moramo delne skalarne produkte iz vsakega bloka prepisati v globalni pomnilnik v polje

Zgled: množenje matrik



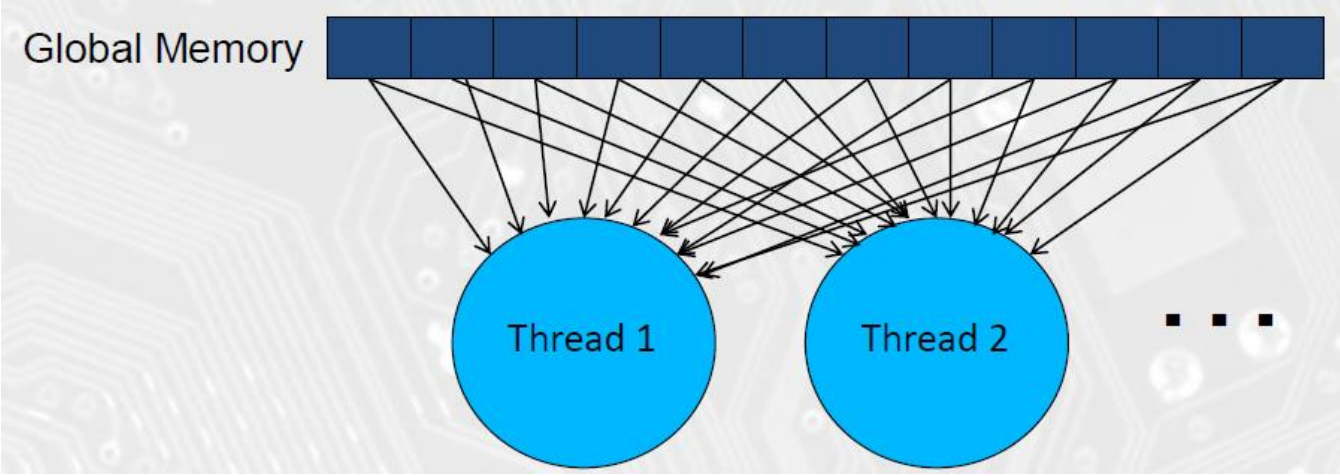
Zgled: množenje matrik

- Enostavna rešitev



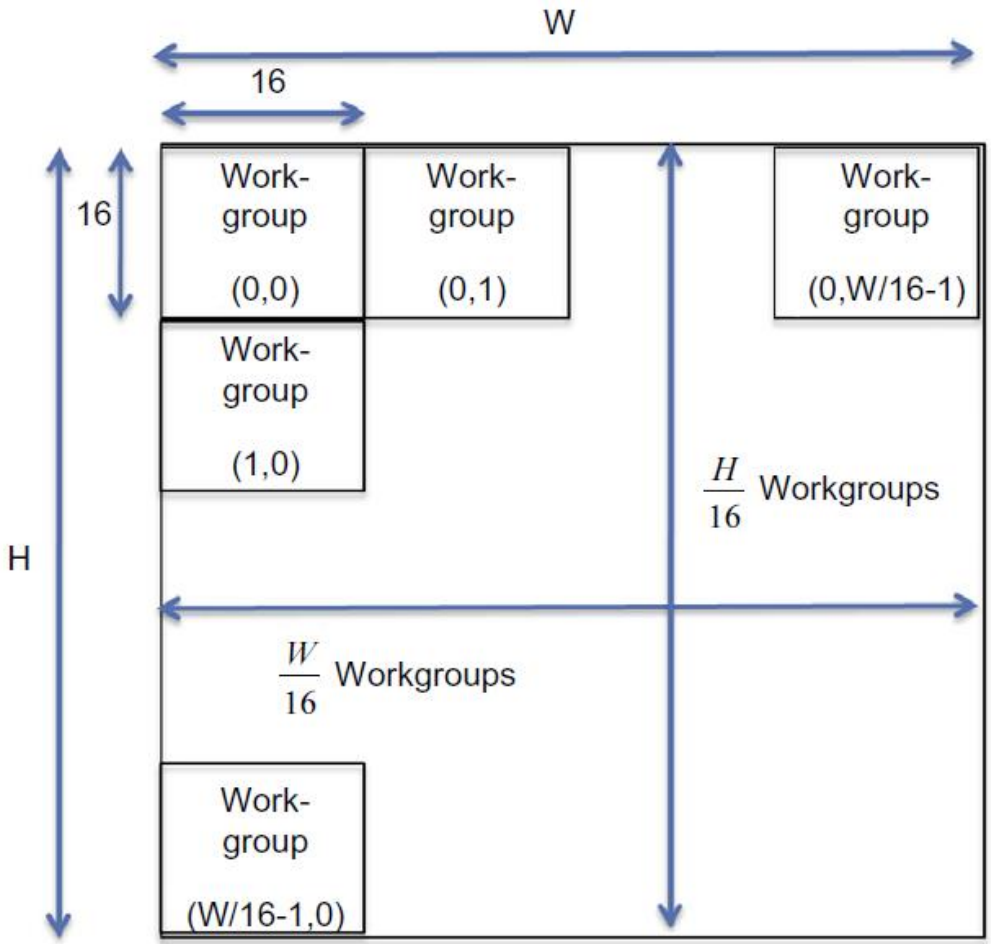
Zgled: množenje matrik

- Enostavna rešitev
  - Niti berejo vrednosti matrik iz globalnega pomnilnika
  - Neporavnan dostop do glavnega pomnilnika
  - Dostopni časi niti do glavnega pomnilnika so različni



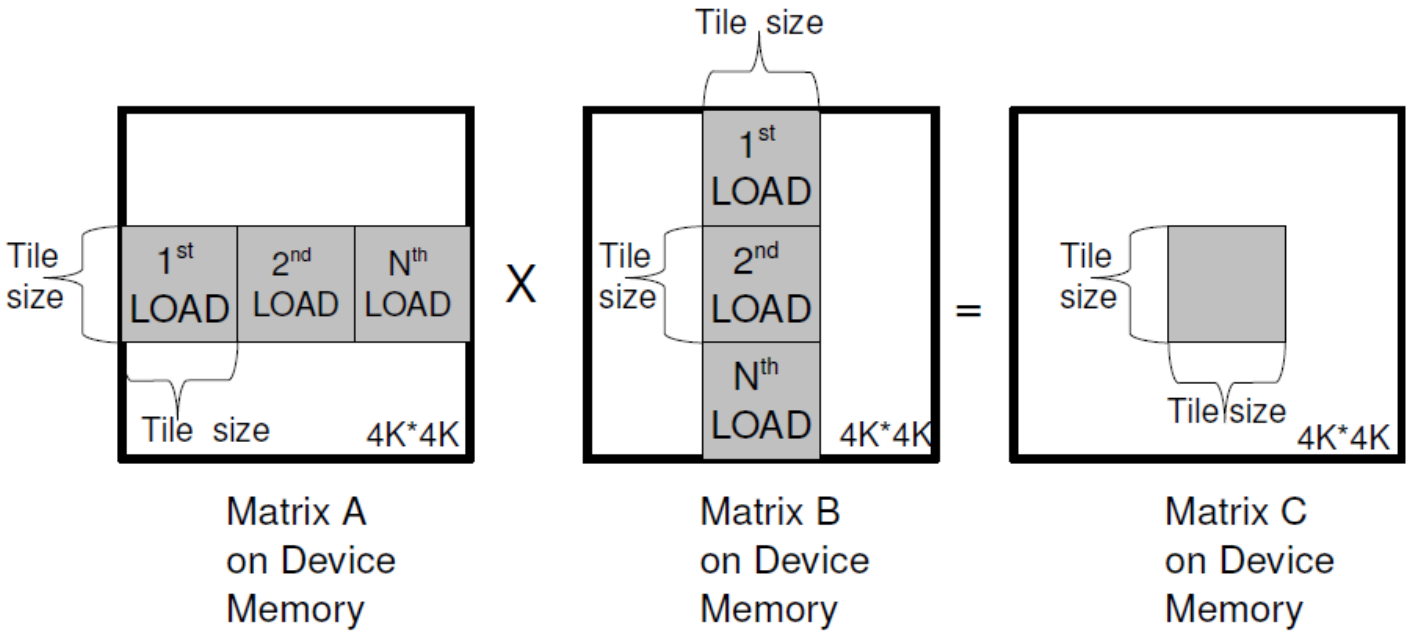
Zgled: množenje matrik

- Ploščice
  - Matriko razdelimo na ploščice



Zgled: množenje matrik

- Ploščice
  - Računanje izvedemo za vsako ploščico matrike C posebej
  - Naložimo po eno ploščico iz matrike A in iz matrike B in izračunamo prispevke za matriko C
  - Postopek ponavljamo, dokler ne obdelamo vseh ploščic



Zgled: množenje matrik

- Ploščice
  - Vsako ploščico najprej skopiramo v lokalni pomnilnik
  - Kopirajo vse niti hkrati, zato je prenos maksimalno poravnan in usklajen
  - Niti hitreje dostopajo do podatkov v lokalnem pomnilniku
  - Izvajanje je zato hitrejše