Paralelno računarstvo

Računarske vežbe

Letnji semestar 2022/23.
Studijski program: Informacioni inženjering

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

CUDA programiranje 4

Sadržaj

- 1D konvolucija
- 2D konvolucija
- Osnovni kernel za 2D konvoluciju
- Paralelna redukcija i njene optimizacije

Konvolucija

Konvolucija

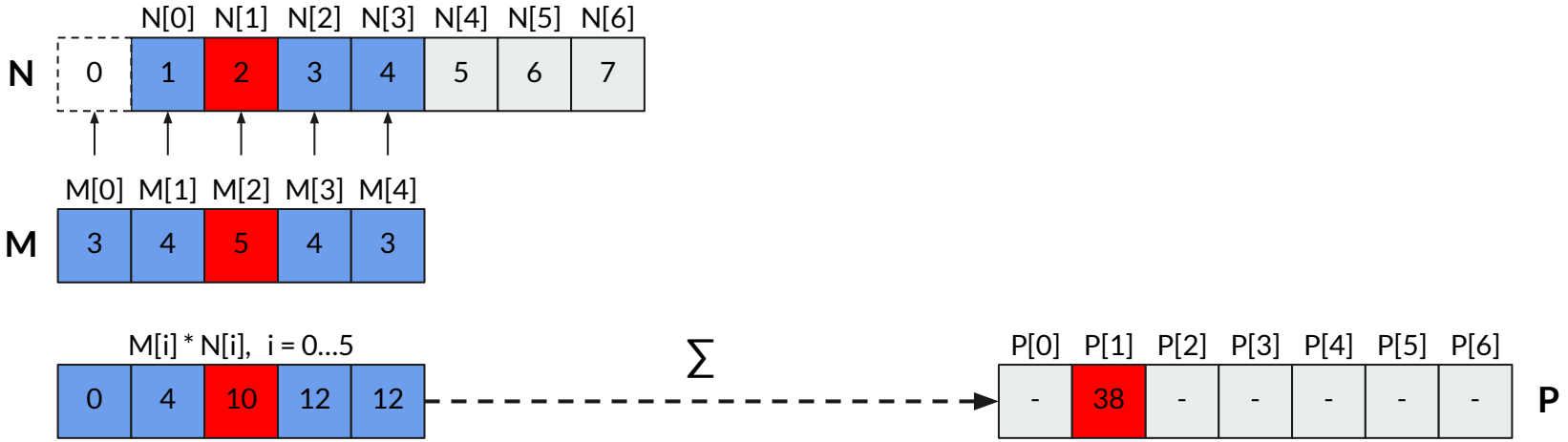
- Često korišćena u obradi audio signala (1D konvolucija) i slike (2D konvolucija) za zamućenje, izdvajanje ivica, itd.

Definicija konvolucije

- Nizovna operacija gde se elemeni izlaznog niza dobijaju kao težinske sume kolekcija susednih elemenata ulaznog niza
- Težine su predstavljene posebnim nizom koji se zove *konvolucioni kernel* (i nema veze sa CUDA kernelima!)
  - U nastavku ćemo konvolucioni kernel zvati konvolucionom maskom
  - ImageBlur vežba je bila specijalni slučaj konvolucije

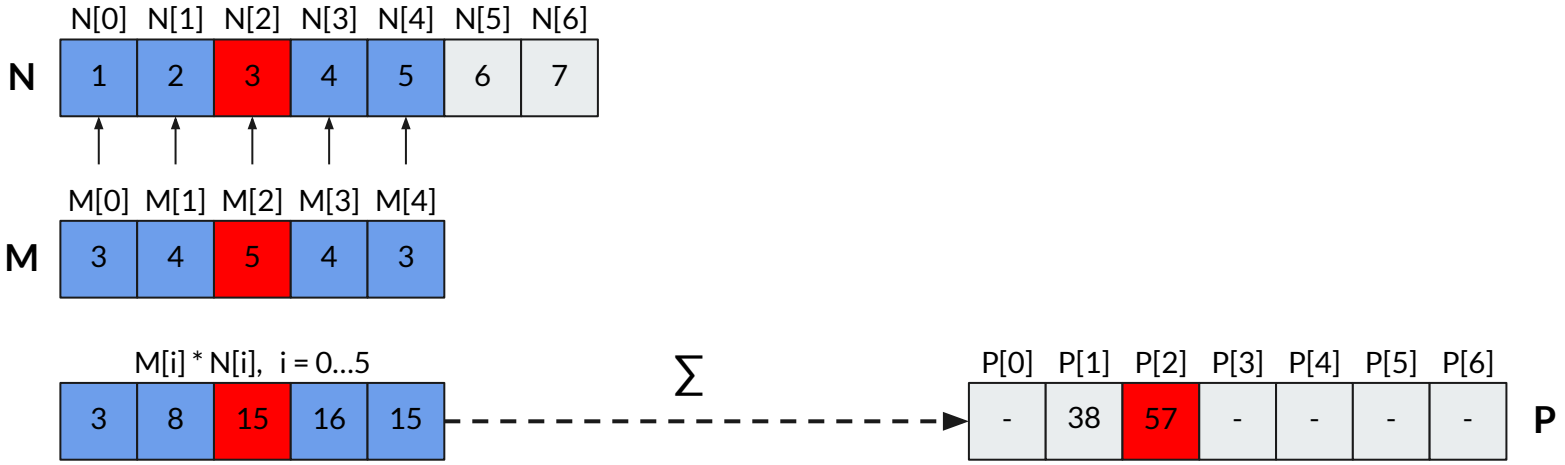
1D konvolucija

Primer: 1D konvolucija



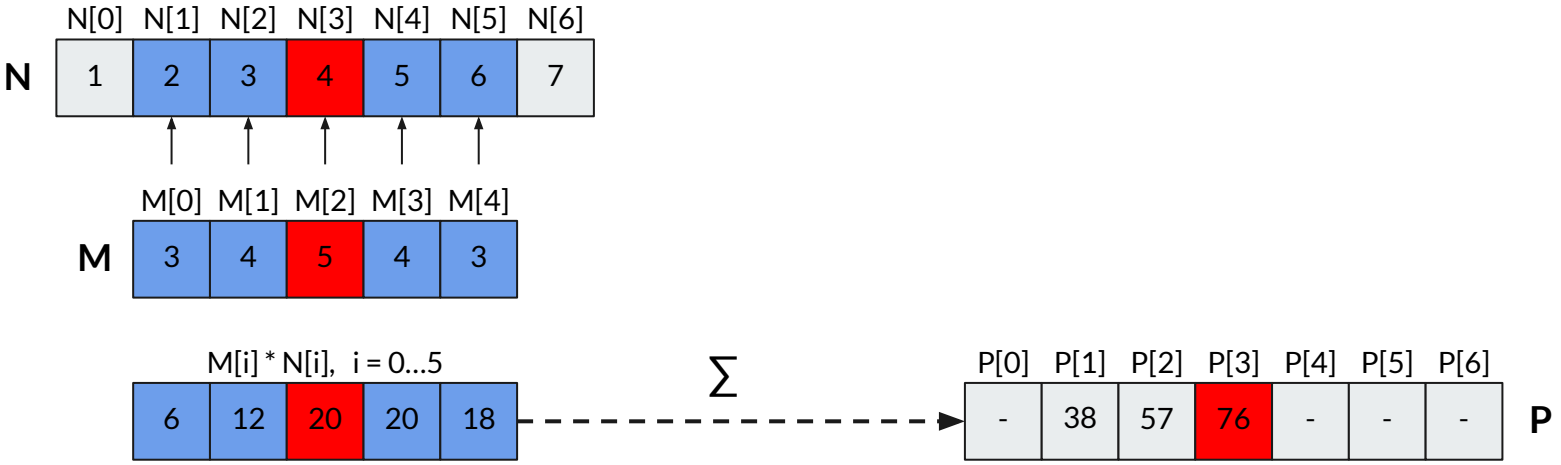
- **N** — ulazni niz, **M** — konvoluciona maska, **P** — rezultat konvolucije
- **P[1] = 0*M[0] + N[0]*M[1] + N[1]*M[2] + N[2]*M[3] + N[3]*M[4]**
- Ovde se element maske koji izlazi iz opsega niza množi nulom, ali može se primeniti i neka druga strategija, recimo replikacija elementa N[0]

Primer: 1D konvolucija



- P[2] = N[0]*M[0] + N[1]*M[1] + N[2]*M[2] + N[3]*M[3] + N[4]*M[4]

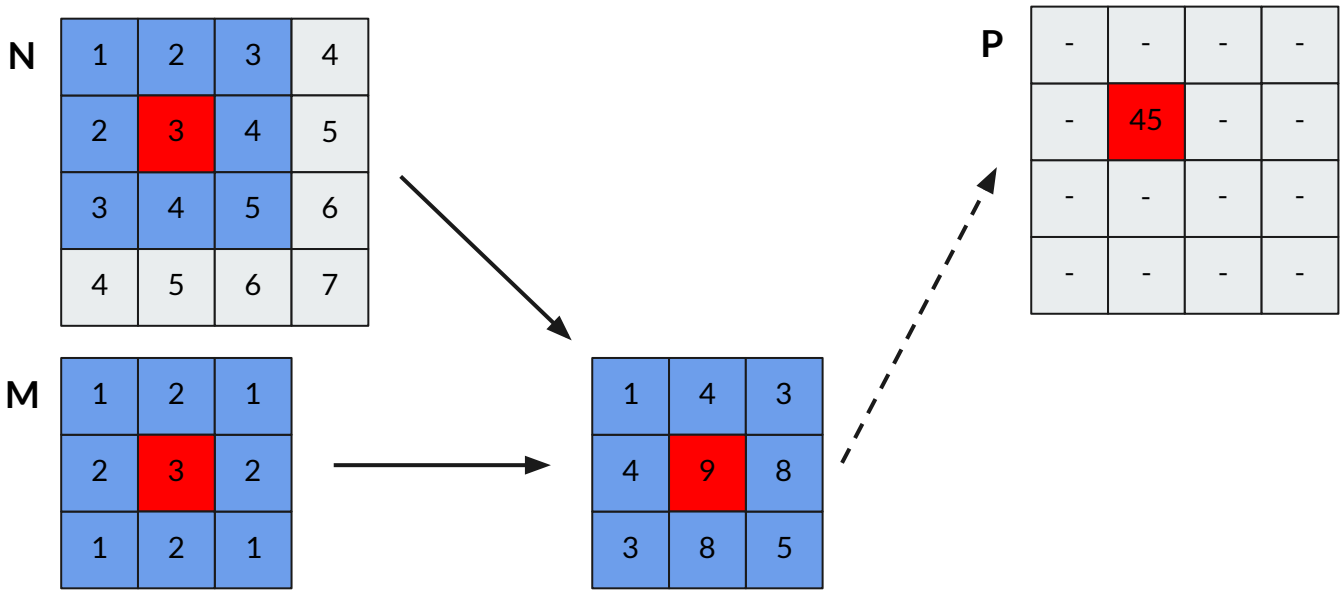
Primer: 1D konvolucija



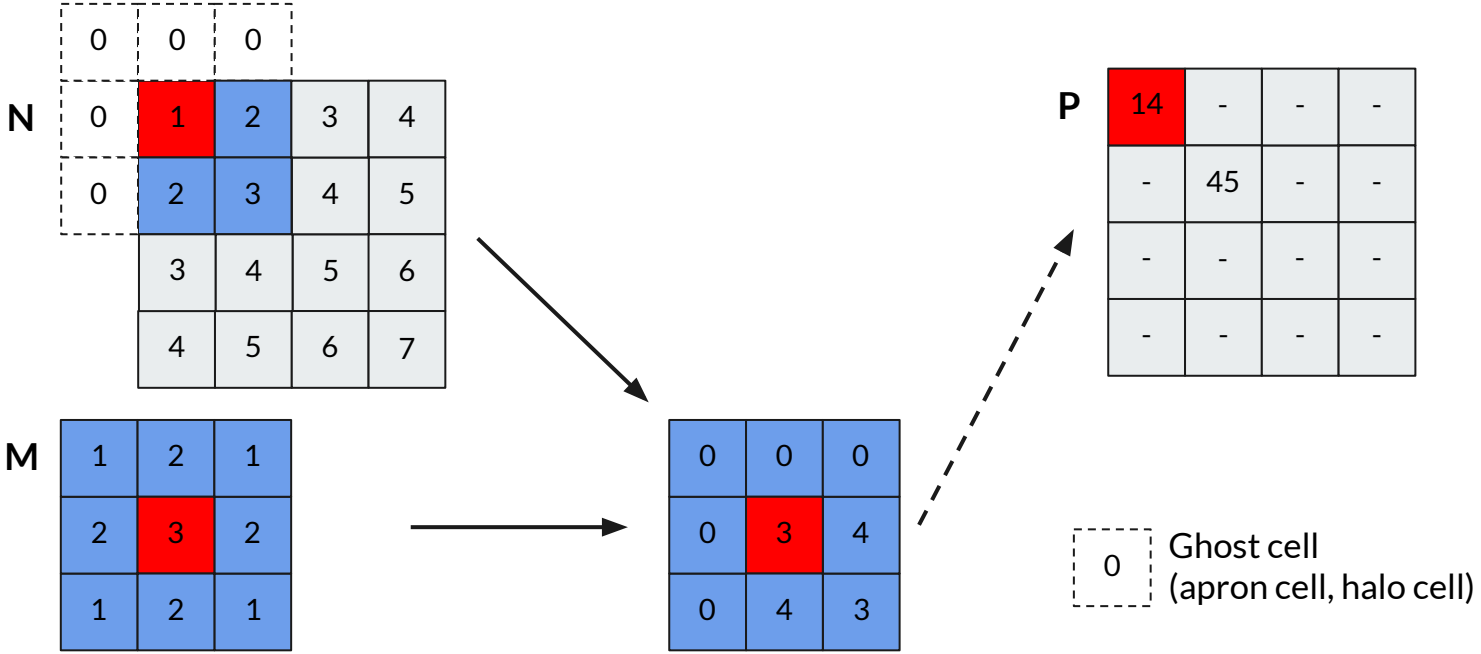
- P[3] = N[1]*M[0] + N[2]*M[1] + N[3]*M[2] + N[4]*M[3] + N[5]*M[4]

2D konvolucija

Primer: 2D konvolucija



Primer: 2D konvolucija



Realni primeri 2D konvolucije

1. Box filter

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Pre konvolucije

Posle konvolucije

Realni primeri 2D konvolucije

2.

Pre konvolucije

0	0	0
0	0	1
0	0	0

Realni primeri 2D konvolucije

2. Pomeranje piksela na levo

Pre konvolucije

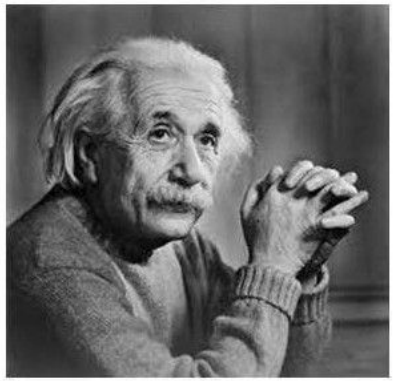
0	0	0
0	0	1
0	0	0

Posle konvolucije

Realni primeri 2D konvolucije

3. Sobel operator

Pre konvolucije

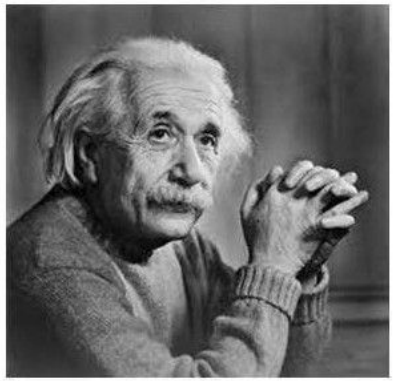


1	0	-1
2	0	-2
1	0	-1

Realni primeri 2D konvolucije

3. Sobel operator — detekcija vertikalnih ivica

Pre konvolucije



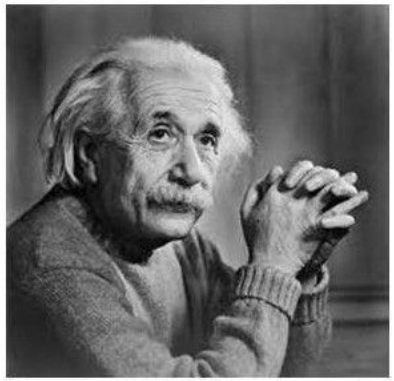
1	0	-1
2	0	-2
1	0	-1

Posle konvolucije

Realni primeri 2D konvolucije

3. Sobel operator — detekcija horizontalnih ivica

Pre konvolucije



1	2	1
0	0	0
-1	-2	-1

Posle konvolucije

Zadatak 1

- Konvolucija matrice, prvi zadatak u Colab svesci

Paralelna redukcija

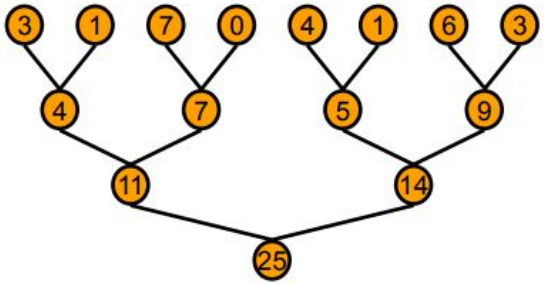
Paralelna redukcija

- Redukcija — obrada niza elemenata koja kao rezultat daje jednu vrednost
- Da li je nešto slično već viđeno u *OpenMP*-u?

Paralelna redukcija u CUDA-i

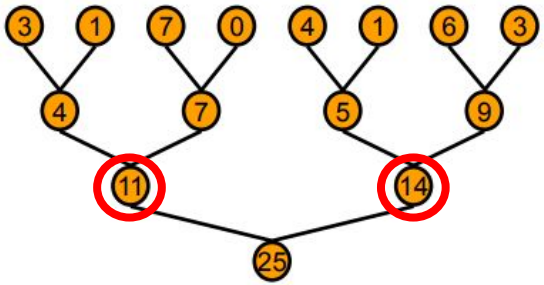
- Odličan primer upotrebe optimizacija o kojima je bilo reči do sada
- Naredni slajdovi su bazirani na slajdovima Marka Harisa o optimizaciji, dostupnim na zvaničnom NVIDIA sajtu: https://developer.download.nvidia.com/assets/cuda/files/reduction.pdf
- Slike su preuzete sa pomenutih slajdova

Paralelna redukcija — ilustracija ideje



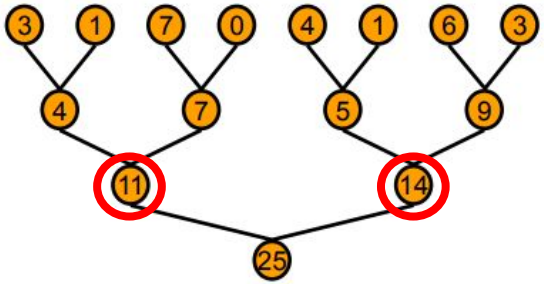
- U jednom koraku, pojedinačna nit obrađuje dva elementa niza
- U narednom koraku, pojedinačna nit obrađuje dva rezultata obrade iz prethodnog koraka
- Ilustracija prikazuje kako se niz globalno treba obraditi, ali se može posmatrati i kao parcijalni rezultat koji proizvodi svaki blok niti

Paralelna redukcija — ilustracija ideje



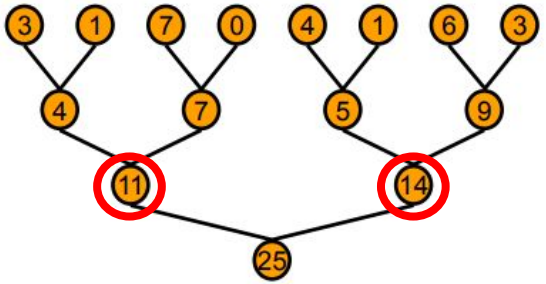
- Smatrajmo da je na vrhu slike početni niz, i da smo pokrenuli dva bloka niti, od kojih svaki obrađuje po jednu polovinu početnog niza
- U tom slučaju, crveno uokvireni elementi jesu rezultati obrade ovih blokova
- Da bi neka nit mogla da sumira rezultate ovih blokova, ona mora da sačeka trenutak u kome su sigurno oba bloka obradila svoje rezultate
- Potrebna nam je **sinhronizacija na nivou mreže**

Paralelna redukcija — ilustracija ideje



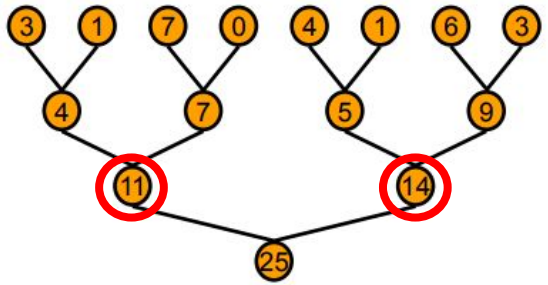
- **Problem** — kako sinhronizovati niti?
- **Rešenje:**

Paralelna redukcija — ilustracija ideje



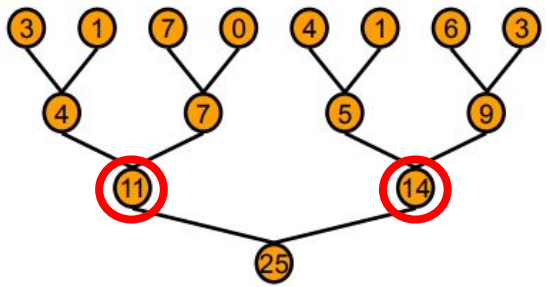
- **Problem** — kako sinhronizovati niti?
- **Rešenje:**
  - __syncthreads?

Paralelna redukcija — ilustracija ideje



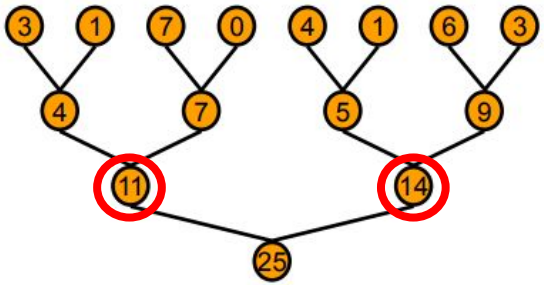
- **Problem** — kako sinhronizovati niti?
- **Rešenje:**
  - __syncthreads?
  - Ova funkcija vrši **sinhronizaciju na nivou bloka**

Paralelna redukcija — ilustracija ideje



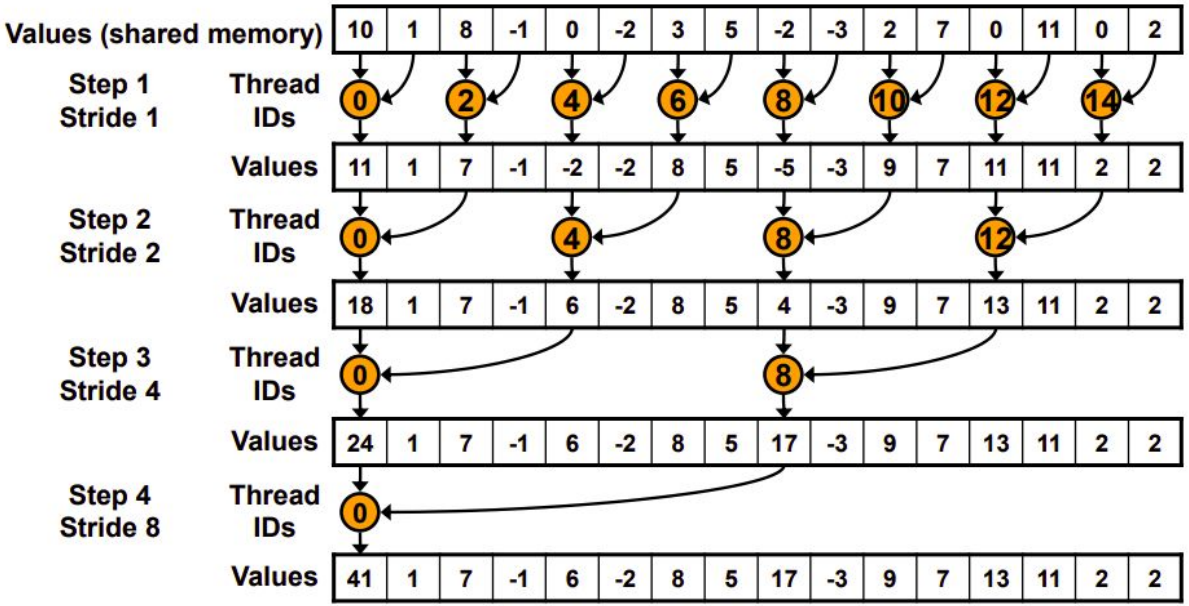
- **Problem** — kako sinhronizovati niti?
- **Rešenje:**
  - Pozvati prvo kernel za više blokova, od kojih svaki generiše rezultat i smešta ga u neki niz
  - Kraj izvršavanja kernela nam je garancija da su svi blokovi završili svoju obradu
  - Pozvati **isti kernel za samo jedan blok**, a kao ulaz proslediti niz parcijalnih rezultata generisanih u prvom pozivu kernela

Paralelna redukcija — ilustracija ideje



- Predloženo rešenje može da koristi isti kernel, koji prima niz kao ulaz i daje niz kao izlaz
- Razlika u dva poziva kernela je ta što se kao rezultat drugog poziva dobija niz od jednog elementa

Paralelna redukcija — prvo rešenje



Paralelna redukcija — prvo rešenje

```
__global__ void reduce0(int *g_idata, int *g_odata) {
        extern __shared__ int sdata[];

        // each thread loads one element from global to shared mem
        unsigned int tid = threadIdx.x;
        unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
        sdata[tid] = g_idata[i];
        __syncthreads();

        // do reduction in shared mem
        for(unsigned int s=1; s < blockDim.x; s *= 2) {
                if (tid % (2*s) == 0) {
                        sdata[tid] += sdata[tid + s];
                }
                __syncthreads();
        }

        // write result for this block to global mem
        if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

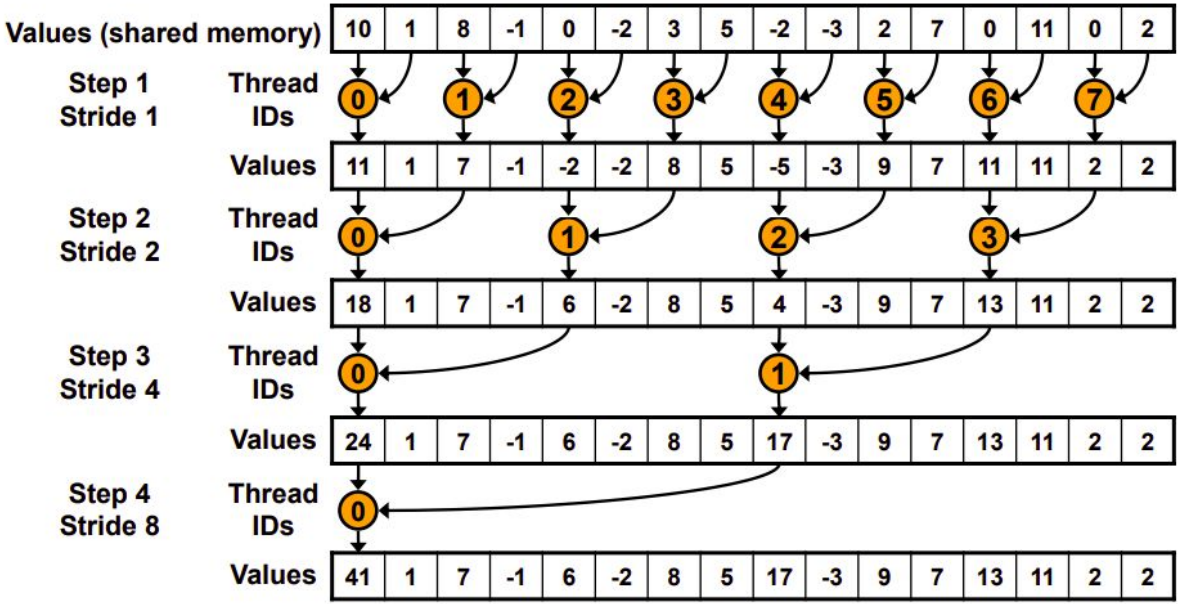
Paralelna redukcija — prvo rešenje

- Svaka nit učitala je iz globalne memorije element koji odgovara njenom globalnom indeksu i upisala ga u deljenu memoriju na poziciju koja odgovara njenom indeksu u bloku
- Nit sa indeksom 0 iz svakog bloka upisuje rezultat obrade tog bloka u deljenu memoriju
- Svaki blok obrađuje svoj deo niza u nekoliko faza (za deo niza veličine 2^n, potrebno je n faza), sve dok ne svede svoj niz na jedan element.
- Problem – divergentnost niti

Paralelna redukcija — prvo rešenje i problem divergentnih niti

- Grananja u CUDA-i mogu da loše utiču na performanse
- Setimo se na kratko i vrlo površno warp-ova — skup niti iz jednog bloka koje se bukvalno izvršavaju u isto vreme
- Ako je kôd takav da neke niti iz warp-a idu u then granu if naredbe, a druge niti iz istog warp-a idu u else granu, CUDA ovo pretvara u dva uzastopna izvršenja — prvo, u kome deo niti izvršava then granu, a ostatak niti efektivno ne radi ništa, i drugo, u kome drugi deo niti izvršava else granu, a ostatak niti je nezaposlen
- Ako već moramo da imamo grananja, najbolje ih je organizovati tako da uzastopne niti idu u istu granu — npr. za blok od 64 niti ako je threadIdx.x < 32, onda idi u then granu, a u suprotnom u else; ovako, veće su šanse da niti iz istog warp-a idu u istu granu

Paralelna redukcija — drugo rešenje



Paralelna redukcija — drugo rešenje

Just replace divergent branch in inner loop:

```
for (unsigned int s=1; s < blockDim.x; s *= 2) {
    if (tid % (2*s) == 0) {
        sdata[tid] += sdata[tid + s];
    }
    __syncthreads();
}
```

With strided index and non-divergent branch:

```
for (unsigned int s=1; s < blockDim.x; s *= 2) {
    int index = 2 * s * tid;

    if (index < blockDim.x) {
        sdata[index] += sdata[index + s];
    }
    __syncthreads();
}
```

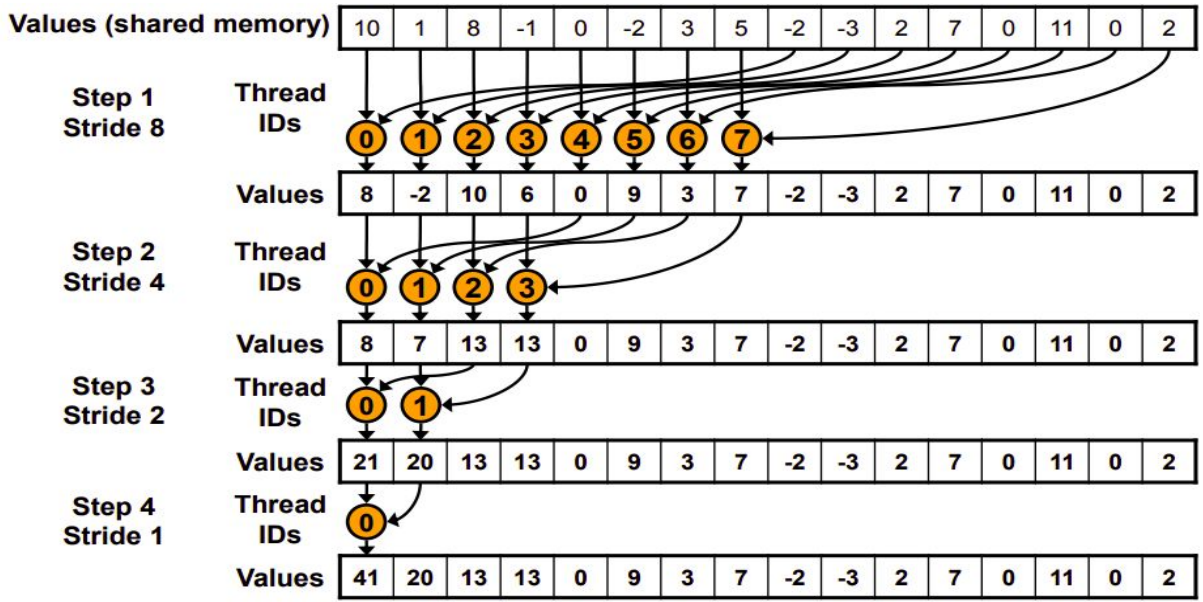
Paralelna redukcija — drugo rešenje

- Vidimo da više ne postoji operand ‘%’; izbaciti ga je bio dobar potez, s obzirom na to da je veoma spor
- Problem divergencije rešen je uvođenjem indeksa koji se menja u zavisnosti od trenutne iteracije, odnosno faze
- Ovakav indeks dovodi do toga da su uvek aktivne sukcesivne niti, a broj aktivnih niti se, kao i u prethodnom rešenju, smanjuje u svakom koraku
- Na primer, u drugom koraku na ilustraciji aktivne su niti 0, 1, 2 i 3, umesto niti 0, 4, 8, 12 koje nisu sukcesivne, kao što je bio slučaj sa prvim rešenjem

Paralelna redukcija — drugo rešenje

- Drugo rešenje ima problem sa konfliktima pri pristupu memorijskim bankama
- Ovo nismo pominjali na kursu i neće biti vremena za detaljnu obradu; jednostavno ćemo prihvatiti da bi u ovom primeru bilo performantnije da, osim što su susedne niti aktivne, da pristupaju i susednim indeksima u nizu u deljenoj memoriji
- **Obratiti pažnju** — Iako je i ovde dobar pristup sukcesivnim elementima, to nije zbog poravnatog pristupa; poravnat pristup se odnosi na globalnu memoriju!

Paralelna redukcija — treće rešenje



Paralelna redukcija — treće rešenje

Just replace strided indexing in inner loop:

```
for (unsigned int s=1; s < blockDim.x; s *= 2)  {
   int index = 2 * s * tid;

   if (index < blockDim.x) {
      sdata[index] += sdata[index + s];
   }
   __syncthreads();
}
```

With reversed loop and threadID-based indexing:

```
for (unsigned int s=blockDim.x/2; s>0; s>>=1) {
   if (tid < s) {
      sdata[tid] += sdata[tid + s];
   }
   __syncthreads();
}
```

Paralelna redukcija — treće rešenje

- Vidimo da je sada parametar s (stride, korak) na početku postavljen na polovinu bloka, pa se smanjuje
- Ako ima 8 niti, a 16 elementa, nit 0 u prvom koraku pristupa elementima na indeksima 0 i 8, nit 1 elementima 1 i 9…
- U drugom koraku upola manje niti je aktivno, pa nit 0 pristupa indeksima 0 i 4, nit 1 indeksaim 1 i 5...

Paralelna redukcija — problem sa trećim rešenjem

- Ovaj problem provlači se od samog početka implementacije naše redukcije
- Polovina niti iz svakog bloka će već u prvoj iteraciji, odnosno fazi obrade, biti neaktivno što se tiče glavne obrade niza
- Svrha polovine ukupnog broja niti jeste samo da učita podatke u deljenu memoriju, nakon čega odmah postaju neaktivne

Paralelna redukcija — četvrto rešenje

Halve the number of blocks, and replace single load:

```
// each thread loads one element from global to shared mem
unsigned int tid = threadIdx.x;
unsigned int i = blockIdx.x*blockDim.x + threadIdx.x;
sdata[tid] = g_idata[i];
__syncthreads();
```

With two loads and first add of the reduction:

```
// perform first level of reduction,
// reading from global memory, writing to shared memory
unsigned int tid = threadIdx.x;
unsigned int i = blockIdx.x*(blockDim.x*2) + threadIdx.x;
sdata[tid] = g_idata[i] + g_idata[i+blockDim.x];
__syncthreads();
```

Paralelna redukcija — četvrto rešenje

- Prepolovimo broj blokova koji se pokreću u startu
- Svaka nit će sada čitati po dva elementa iz glavne memorije, odmah ih sabirati i smeštati u deljenu memoriju

Paralelna redukcija — konačan kernel

```
__global__ void reduce0(int *g_idata, int *g_odata) {
        extern __shared__ int sdata[];

        // each thread loads one element from global to shared mem
        unsigned int tid = threadIdx.x;
        unsigned int i = blockIdx.x * (blockDim.x * 2) +  threadIdx.x;
        sdata[tid] = g_idata[i] + g_data[i + blockDim.x];
        __syncthreads();

        // do reduction in shared mem
        for(unsigned int s = blockDim.x / 2; s > 0; s >>= 1) {
                if (tid < s) {
                        sdata[tid] += sdata[tid + s];
                }
                __syncthreads();
        }

        // write result for this block to global mem
        if (tid == 0) g_odata[blockIdx.x] = sdata[0];
}
```

Paralelna redukcija — poziv kernela

```
...
int blockSize = 256
int gridSize = N / blockSize / 2;
int shMemSize = blockSize;
reduce<<<gridSize, blockSize, shMemSize>>>(data_in, data_intermediate);

blockSize = gridSize / 2;
gridSize = 1;
shMemSize = blockSize;
reduce<<<gridSize, blockSize, shMemSize>>>(data_intermediate, data_out);
...
```

- Pri drugom pozivu, duplo manji broj niti od broja elemenata ulaznog niza postiže se smanjivanjem veličine bloka na pola