

# Osnove grafičkih procesnih jedinica

uvod u ne-grafičke primjene GPU-ova

Siniša Šegvić  
UniZg-FER

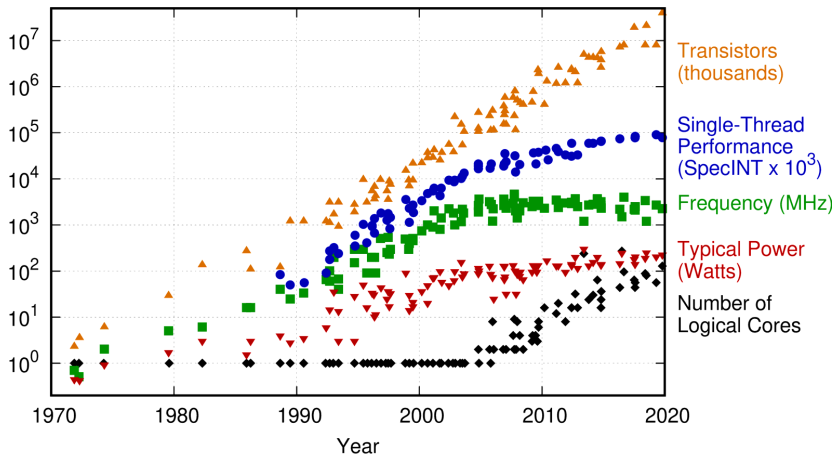
# PLAN

- usporedba s CPU-ovima
- organizacija GPU-ova: dretve, grupe, blokovi
- izražavanje GPU programa
- osnove OpenCL-a

# UVOD : SVIJET SE MIJENJA

## Performansa skalarnih procesora ulazi u zasićenje

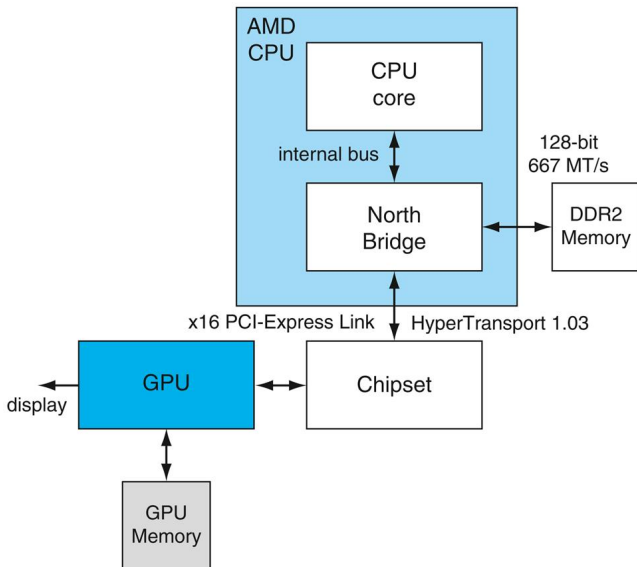
48 Years of Microprocessor Trend Data



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten  
New plot and data collected for 2010-2019 by K. Rupp

[rupp19github]

# GPU vs CPU : POZICIJA



[patterson20book]

# GPU vs CPU: VRŠNA PERFORMANSA (vp), FLOPS

CPU i9 10900K (Comet lake, 14nm, 206 mm<sup>2</sup>, 2020):

- 10 jezgara, 5.3 Ghz turbo
- svaka jezgra može izdati  $2 \times \text{AVX-512}$  u svakom taktu
  - SIMD: jedna instrukcija obrađuje više parova podataka
  - AVX-512:  $512/32=16$  FMA operacija u fp32 preciznosti
- $vp = 10 * 5.3G * 16 * 2 * 2 = 3.4 \text{ TFLOPS}$

GPU A100 (Ampere, 7nm, 826 mm<sup>2</sup>, 2020, 40 GB RAM):

- 108 multiprocesora (SMP), 1.4 GHz
- SMP: 64 procesora (SP, CUDA core)
- $vp = 108 * 1.4G * 64 * 2 = 19.4 \text{ TFLOPS}$

# GPU vs CPU: VRŠNA PERFORMANSA (2)

## Memorijski sustav (CPU):

- širina sabirnice (JEDEC DDR): 64 bit
- memorijska propusnost (DDR5 - 6000 MHz, 4 kanala): 192 GB/s

## Memorijski sustav (GPU):

- širina sabirnice(A100): 5120 bit
- memorijska propusnost: 1.5 TB/s

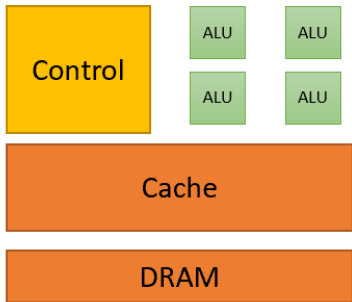
## Vršna performansa je odokativan pokazatelj:

- GPU-ovi dolaze bliže vršnoj performansi od CPU-ova
  - CPU-ovi obavljaju raznovrsne zadatke s puno uvjetnih grananja
- za paralelne zadatke, GPU-ovi su  $50\times$  brži
- za zadatke koji se ne mogu paralelizirati, GPU-ovi su neupotrebljivi

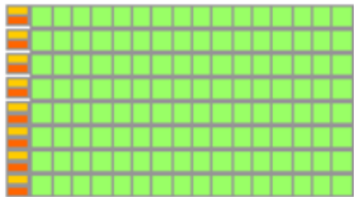
# GPU vs CPU : RASPODJELA RESURSA

Različita alokacija tranzistora:

- CPU - prvenstveno na priručne memorije i upravljanje
- GPU - prvenstveno na računanje



CPU



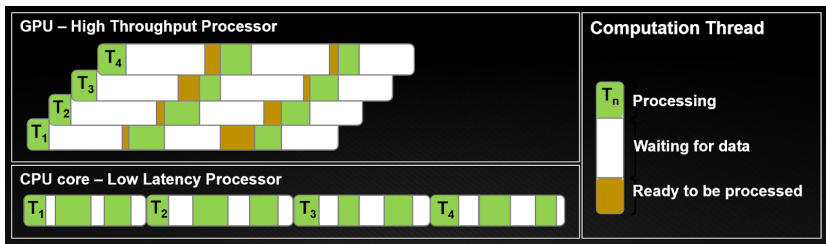
GPU

# GPU vs CPU: PRISTUP LATENCIJI

Umanjivanje problema vezanih uz hazarde:

- CPU - skraćuje latenciju sofisticiranim cachevima
- GPU - održava propusnost sklopovskom višedretvenošću (SIMT)
  - ideja je slična razvijanju petlje (preplitanje nezavisniog koda)
  - za one koji žele naučiti malo više: SIMD < SIMT < SMT

<https://yosefk.com/blog/simd-simt-smt-parallelism-in-nvidia-gpus.html>



[gupta20nvidia]

GPU uspijeva zaposliti procesore unatoč dužem čekanju na podatke



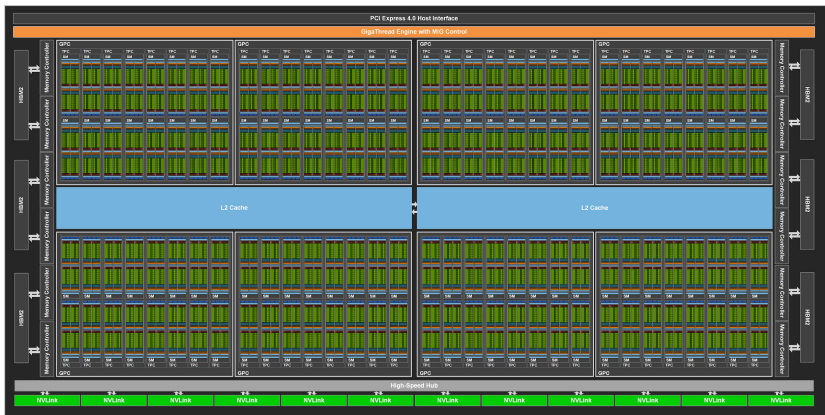
# GPU vs CPU : SAŽETAK

| Feature                                          | Multicore with SIMD | GPU             |
|--------------------------------------------------|---------------------|-----------------|
| SIMD processors                                  | 8 to 32             | 15 to 128       |
| SIMD lanes/processor                             | 2 to 4              | 8 to 16         |
| Multithreading hardware support for SIMD threads | 2 to 4              | 16 to 32        |
| Largest cache size                               | 48 MiB              | 6 MiB           |
| Size of memory address                           | 64-bit              | 64-bit          |
| Size of main memory                              | 64 GiB to 1024 GiB  | 4 GiB to 16 GiB |
| Memory protection at level of page               | Yes                 | Yes             |
| Demand paging                                    | Yes                 | No              |
| Cache coherent                                   | Yes                 | No              |

| i9-10900k | A100      |
|-----------|-----------|
| 10        | 108       |
| 16        | 64        |
| 2         | 32        |
| 20 MiB    | 40 MiB    |
| 64        | 5120      |
| 128 GiB   | 40-80 GiB |

[patterson20book]

# GPU ORGANIZACIJA : CJELOKUPNI SUSTAV (A100)



[nvidia]

- 128 (108) multiprocesora (ugrubo, odgovaraju CPU jezgrama)
- svaki multiprocesor ima 64 jednostavna procesora (CUDA core)
- svi procesori multiprocesora izvršavaju istu instrukciju (SIMD-like)

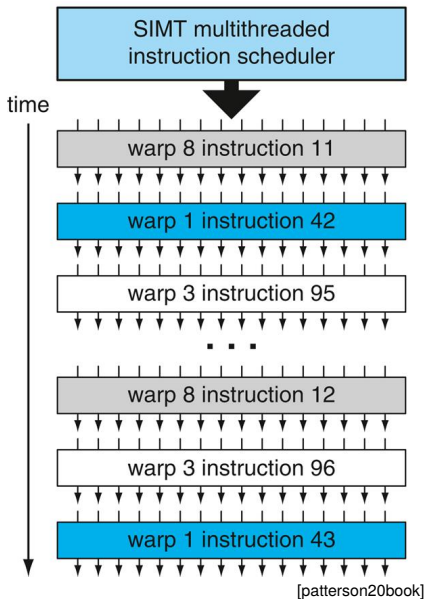
# GPU ORGANIZACIJA : MULTIPROCESOR



[nvidia]

- procesori (SP) multiprocссора (SMP) izvode različite dretve
- grupa (warp): skup dretvi koje se istovremeno izvršavaju na istom SMP-u
- dretve grupe su sinkrone: svaki SP izvršava istu instrukciju (dijeljeni PC!)
- grananje je dozvoljeno ali jako usporava izvođenje
- latencija se ublažava paralelnim izvođenjem **više grupa**

# GPU ORGANIZACIJA : SIMT



- SIMT: single instruction, multiple threads
- multiprocesor mijenja grupu nakon svakog takta
- **blok dretvi** (thread block): sve grupe dodijeljene istom SMP-u
- preklapanje grupa dretvi provodi se sklopovski
- srodno vektorskim (SIMD) i višedretventim računalima (SMT)
- možda najinovativniji koncept modernih GPU-ova

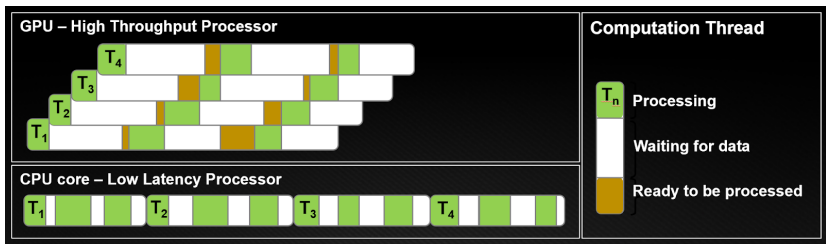
# GPU ORGANIZACIJA : PRISTUP LATENCIJI

Svaki procesor je protočan: ako imamo dovoljno grupa, možemo zamaskirati sporu priručnu memoriju i hazarde.

Zbog toga procesori najčešće uspijevaju raditi nešto korisno

Može biti korisno kad:

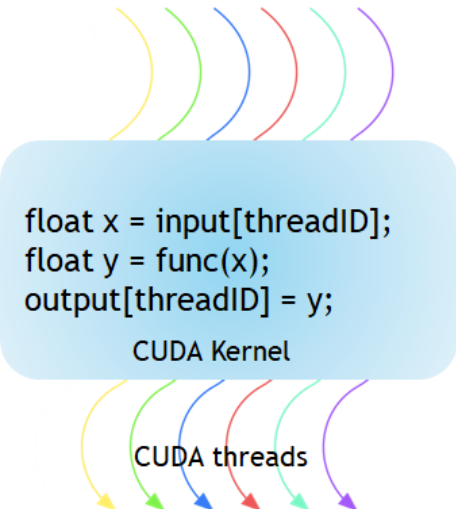
- na raspolaganju imamo puno grupa dretvi (tj. problem je paralelan)
- dretve ne koriste grananje



[gupta20nvidia]

# PROGRAMIRANJE : PROBLEMI

Pogodni problemi obrađuju mnogo podataka na sličan način:



```
float x = input[threadID];  
float y = func(x);  
output[threadID] = y;
```

CUDA Kernel

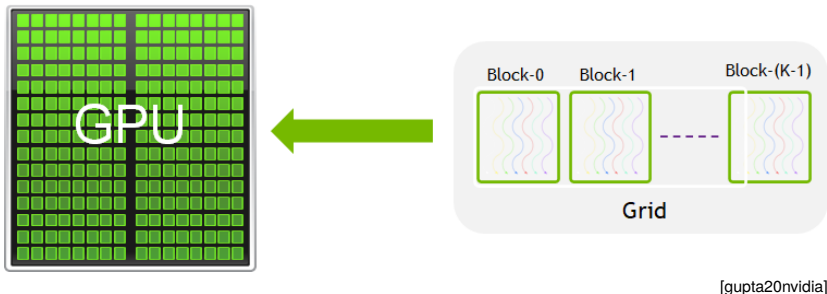
CUDA threads

[gupta20nvidia]

# PROGRAMIRANJE : STRUKTURA

Programi za GPU izražavamo u obliku polja (grid) nezavisnih dretvi.

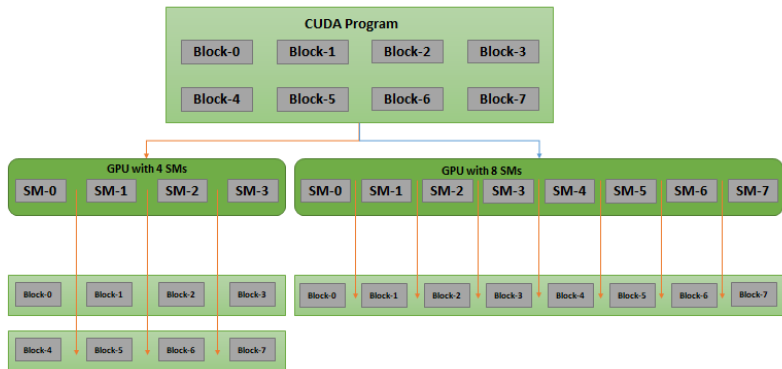
Programski okviri (npr. CUDA, OpenCL) transparentno grupiraju dretve u blokove te ih raspoređuju multiprocesorima



Npr. ako jezgru za obradu floatova konfiguriramo tako da ima blokove od 256 dretvi, oni će na A100 biti raspoređeni u 4 grupe.

## PROGRAMIRANJE: STRUKTURA (2)

Prevedeni program može se pokretati na sklopovlju različite snage:



[gupta20nvidia]

Ipak, najbolju performansu postizemo prevođenjem za ciljani uređaj

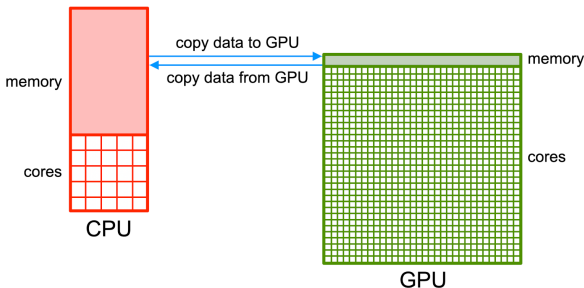
- korak dalje: ciljana optimizacija (TensorRT)



# PROGRAMIRANJE: IZVORNI KOD

Tipično, programi za GPU izvršavaju se barem djelomično na CPU-u

- CPU: učitavanje ulaza i pohranjivanje rezultata
- heterogeno računarstvo!



```
data = open("input.dat");  
copyToGPU(data);  
matrix_inverse(data.gpu);  
copyFromGPU(data);  
write(data, "output.dat");
```

```
# read the data on the CPU  
# copy the data to the GPU  
# perform a matrix operation on the GPU  
# copy the resulting output to the CPU  
# write the output to file on the CPU
```

# OPENCL: UVOD

OpenCL i CUDA su dominantni okviri za programiranje na GPU-ovima

Jezgra: osnovna jedinica koda za GPU (OpenCL, CUDA)

- sve dretve bloka su instance iste jezgre
- tipično odgovara jednoj funkciji

Prednosti OpenCL-a:

- podržava Intelove i AMD-ove GPU-ove (CUDA je NVidijin proizvod)
- može se izvršavati i na CPU-ovima

Instalacija na arch Linuxu:

```
$ sudo pacman -S intel-compute-runtime ocl-icd opencl-headers
```

## OPENCL : MINIMALNA JEZGRA

Ova jezgra izražava množenje jednog elementa cjelobrojnog polja zasebnom dretvom:

```
__kernel void simple_demo(  
    __global int *src,  
    __global int *dst,  
    int factor)  
{  
    int i = get_global_id(0);  
    dst[i] = src[i] * factor;  
}
```

Izvorni kod jezgre prevodimo pozivom funkcije `clBuildProgram`:

- na taj način osiguravamo just-in-time prevođenje za ciljani uređaj

Objekt koji enkapsulira jezgru dobivamo funkcijom `clCreateKernel`:

- simboličko ime jezgre (`simple_demo`) zadajemo argumentom

# OPENCL: IZVRŠAVANJE JEZGRE

Izvršavanje jezgre iniciramo funkcijom `clEnqueueNDRangeKernel`:

- rezultat poziva: stvaranje polja dretvi (grid)
- svaka dretva - jedan prolazak kroz jegrenu funkciju

```
clEnqueueNDRangeKernel(queue, kernel, 1,  
    NULL, global_work_size, NULL, 0, NULL, NULL)
```

# OPENCL : IZVRŠAVANJE JEZGRE

Najvažniji parametri funkcije `clEnqueueNDRangeKernel` su redom:

- `command_queue` - asinkroni red zadataka za odabrani uređaj
- `kernel` - jezgra koju želimo pokrenuti
- `work_dim` - broj dimenzija polja dretvi koje će instancirati jezgra
- `global_work_offset` - početni indeksi polja dretvi (ako NULL, onda se počinje od indeksa 0)
- `global_work_size` - polje od `work_dim` elemenata, određuje dimenzije polja dretvi
- `local_work_size` - polje od `work_dim` elemenata, određuje dimenzije blokova dretvi (ako NULL, blokove dimensionira OpenCL)

<https://man.openc1.org/clEnqueueNDRangeKernel.html>

## OPENCL : VJEŽBA

Četvrta laboratorijska vježba traži dvije naivne izvedbe množenja kvadratnih matrica:

- jednu u C-u za CPU,
- drugu u OpenCL-u za GPU.

Zadatak je izmjeriti brzinu izvođenja dviju izvedbi, kao i vrijeme prijenosa podataka s CPU-a na GPU.

Upute sadrže više pitanja koja omogućavaju samotestiranje: nemojte ih preskočiti!