OpenCL tutorijal

Sadržaj:

Uvod u OpenCL

Heterogeni sistemi

Moderne računarske platforme sadrže:

- Jedan ili više CPU
- Jedan ili više GPU
- Kombinaciju CPU i GPU
- Namenske DSP
- Ubrzivače
- FPGA

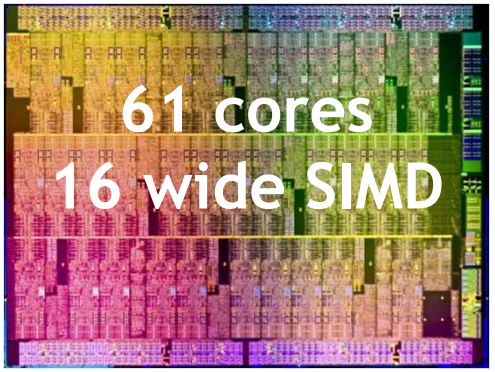
Na primer Qualcomm® Snapdragon 810 MSM8994:

- 4 x Cortex-A57 (2 GHz)
- 4 x Cortex-A53 (1.5 GHz)
- Adreno 420 (128 pipelines)

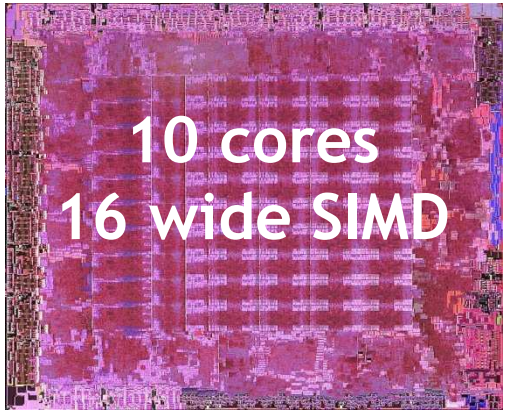
OpenCL omogućuje razvoj prenosivih (portabilnih) programa sposobnih da koriste sve raspoložive resurse koji su dostupni na konkretnom heteregenom sistemu.

Trendovi i izazovi

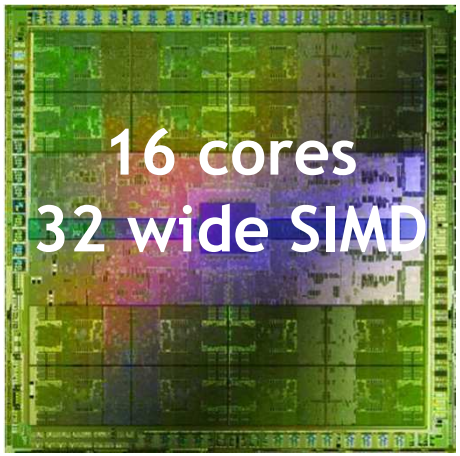
Težnja ka (moguće heterogenim) više-jezgarnim (*multi-core*) i mnogo-jezgarnim (*many-core*) arhitekturama.



Intel® Xeon Phi™ coprocessor



ATI™ RV770

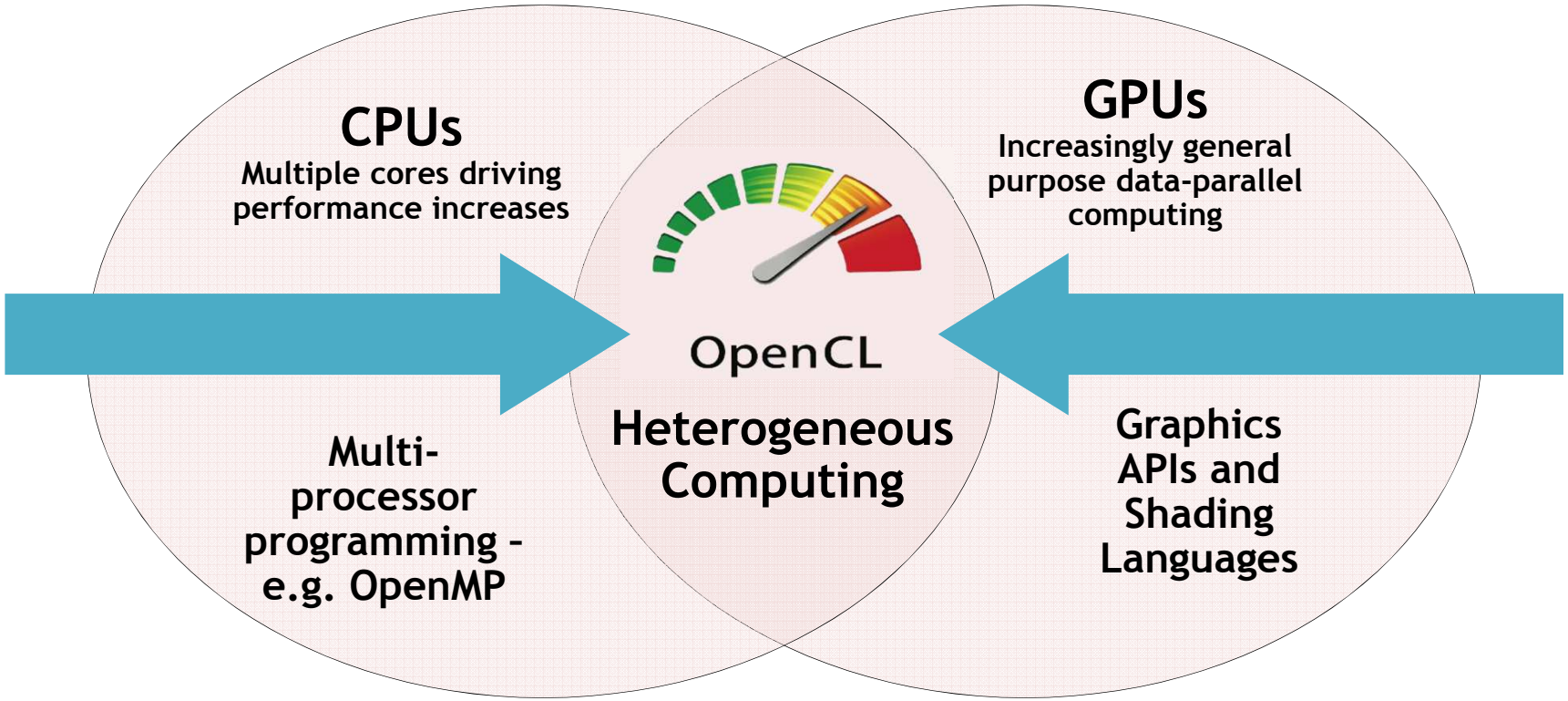


NVIDIA® Tesla® C2090

Izazovi heterogenih više-procesorskih arhitektura:

Kako i na koji način razvijati softver za takve sisteme?

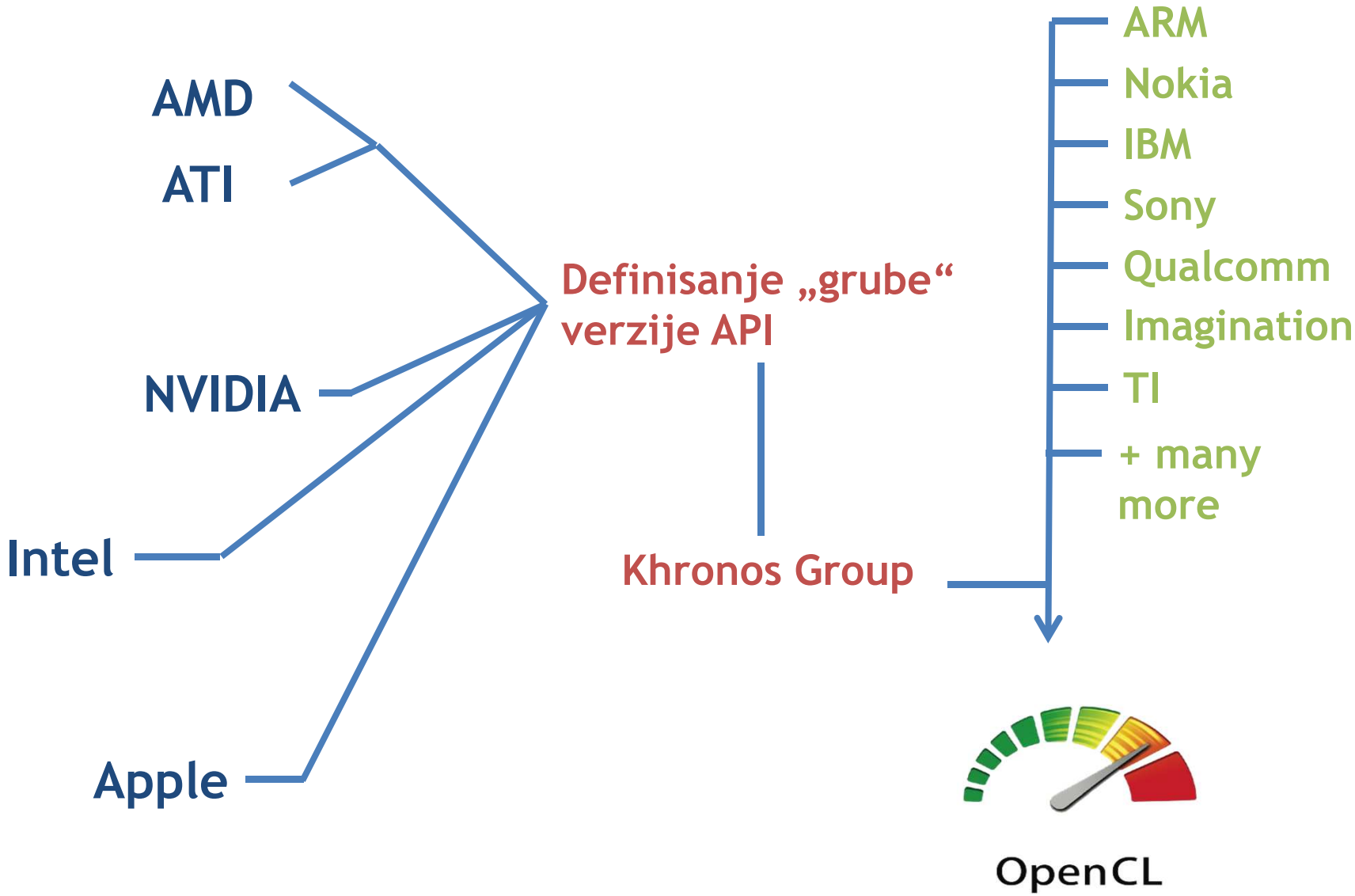
Industrijski standardi za programiranje heterogenih sistema



OpenCL – Open Computing Language

Otvoren, besplatan standard za pisanje prenosivh programa za paralelnu obradu podataka na heterogenim platformama koje uključuju CPU, GPU i druge tipove procesora.

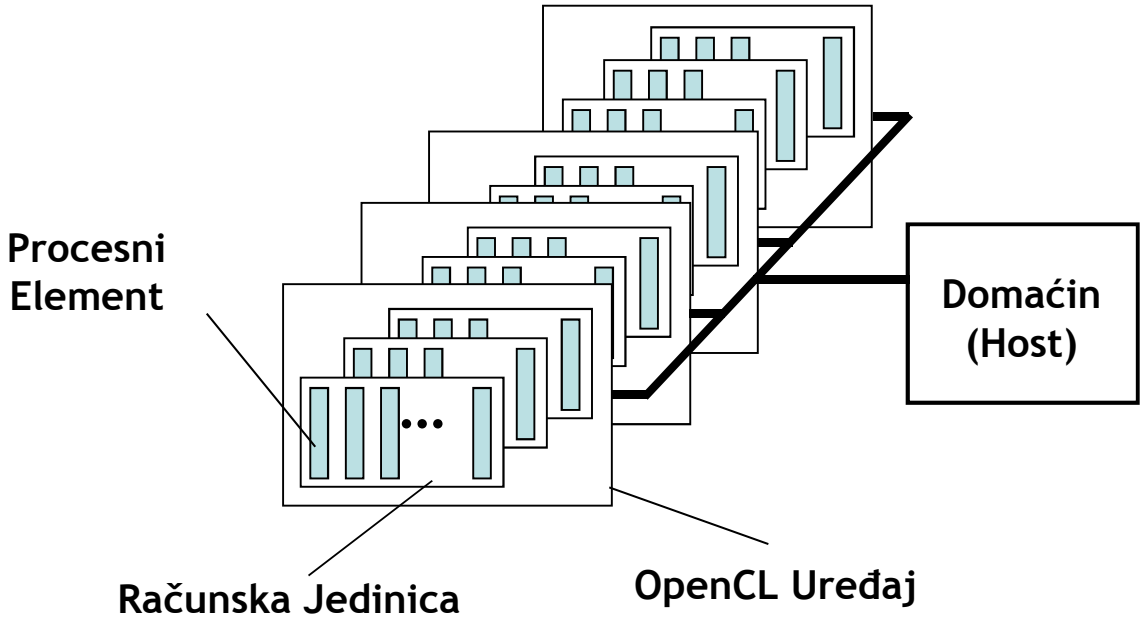
Početak OpenCL standarda



OpenCL Radna grupa

- Uključuje različite industrijske činioce:
  - Proizvođače procesora i računarske opreme
  - Kompanije za razvoj programskih komponenti
- Zahvaljujući članovima Radne grupe koji su podržali njegov razvoj OpenCL je postao značajan i široko korišćen standard.

Model OpenCL platforme



- Jedan *Domaćin* sadrži jedan ili više *OpenCL Uređaja*
  - Svaki OpenCL Uređaj sadrži jednu ili više *Računskih Jedinica*
    - Svaka Računska Jedinica podeljena je na jedan ili više *Procesnih Elemenata*
- Memorija je podeljena na *memoriju domaćina* i *memoriju uređaja*.

Primer OpenCL platforme (dvo-jezgarni CPU i dva GPU)

CPU:

- Tretira se kao OpenCL uređaj
  - Jedna RJ po jezgru
  - Jedan PE po RJ
  - ili ako su PE mapirani na SIMD trake, *n* PE po RJ, gde je *n* SIMD širina
- Napomena:
  - CPU se ujedno smatra i svojim Domaćinom

GPU:

- Svaki GPU predstavlja poseban OpenCL uređaj
- Kroz OpenCL moguće je konkuretno koristiti CPU i sve GPU uređaje

RJ = Računarska Jedinica; PE = Procesni Element

Primer OpenCL platforme (GPU AMD Radeon 6970)

GPU:

- Tretira se kao OpenCL Uređaj
  - 24 SIMD jezgra (24 RJ)
  - Svako SIMD jezgro sadrži 16 SIMD traka (16 PE)
  - Svaka SIMD traka izvršava veoma dugačku instrukcionu reč (*Very Large Instruction Word*) sa četiri operacije (4 OP)
  - Istovremeno se izvrše 1536 operacije (24 RJ x 16 PE x 4 OP)

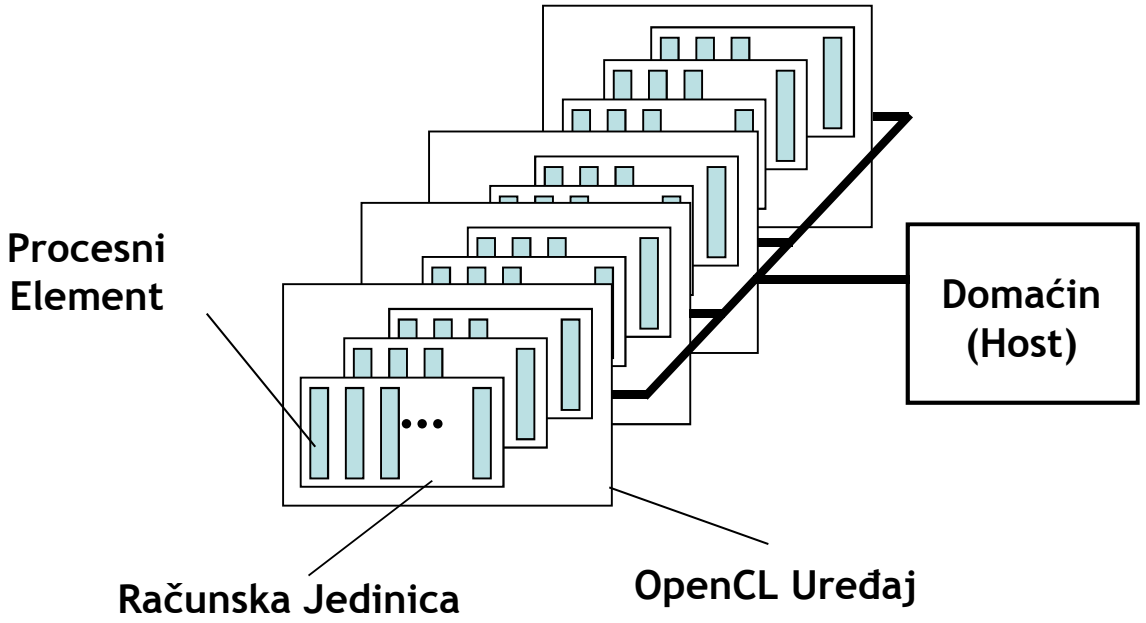
RJ = Računarska Jedinica; PE = Procesni Element

Naša plaforma?

- Šta čini našu platformu:
  - CPU?
  - GPU?
  - Ubrzivač?
  - Ostalo?

OpenCL koncepti

Model OpenCL platforme



- Jedan *Domaćin* sadrži jedan ili više *OpenCL Uređaja*
  - Svaki OpenCL Uređaj sadrži jednu ili više *Računskih Jedinica*
    - Svaka Računska Jedinica je podeljena na jedan ili više *Procesnih Elemenata*
- Memorija je podeljena na *memoriju domaćina* i *memoriju uređaja*.

Osnovna ideja iza OpenCL

- Podelom iteracionog prostora petlju zameniti sa mnogo funkcija jezgra (*kernel*) koje se paralelno izvršavaju.
  - Obradu slike rezolucije 1024x1024 razložiti na jednu funkciju jezgra po pikselu: 1024x1024=1,048,576 funkcija jezgra

Standardna petlja

```
void
mul(const int n,
    const float *a,
    const float *b,
          float *c)
{
  int i;
  for (i = 0; i < n; i++)
    c[i] = a[i] * b[i];
}
```

Paralelna OpenCL obrada

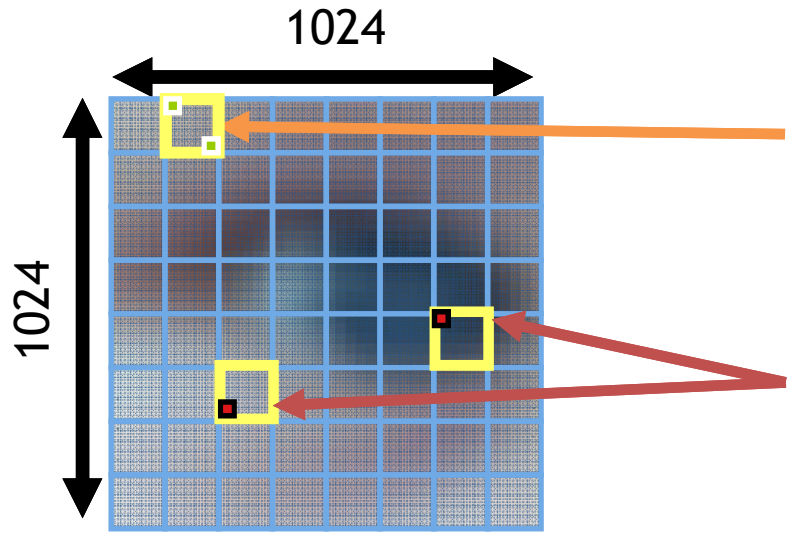
```
__kernel void
mul(__global const float *a,
    __global const float *b,
    __global       float *c)
{
  int id = get_global_id(0);
  c[id] = a[id] * b[id];
}
// mnogo instanci funkcija jezgra,
// nazvane radne stavke (work-items),
// se paralelno izvršavaju
```

Organizacija konkurentnog izvršenja

- Funkcija jezgra (kernel funkcija) definiše operacije nad podacima
- Osnovna jedinica konkurentnog izvršenja naziva se **radna stavka (*work-item*)**
  - Radna stavka izvršava funkciju jezgra
- Radne stavke se logički mogu organizovati u **radna grupe (*work-group*)**

N-dimenzioni opseg radnih-stavki

- Globalne dimenzije:
  - 1024x1024 (veličina celog prostora problema)
- Lokalne dimenzije:
  - 64x64 (veličina radne grupe, izvršavaju se zajedno)



Sinhronizacija radnih-stavki je moguća samo unutar iste radne-grupe (barijerna sinhronizacija)

Unutar funkcije jezgra nije moguće ostvariti sinhronizaciju radnih-grupa

- Definišemo dimenzije koje „najviše“ odgovaraju konkretnom problemu koji rešavamo

OpenCL *N*-dimenzioni opseg (NDO)

- Problem koji rešavamao trebao bi da ima svojstvo **dimenzionalnosti**:
  - Izvrši funkciju jezgra nad svim elementima niza (1D)
  - Izvrši funkciju jezgra nad svim tačkama matrice (2D)
- Način izvršenja jezgara opisuje se kroz specifikaciju globalnog NDO i lokalnog NDO
- **Globalni NDO** predstavlja ukupan opseg (prostor) problema (npr. veličina niza ili dimenzije matrice)
- **Lokalni NDO** predstavlja veličinu radne grupe

OpenCL *N*-dimenzioni opseg (NDO)

- Globalni i lokalni NDO se definisu kao **jedno-**, **dvo-** ili **tro-dimenzioni** prostor indeksa radnih stavki preko kojih se preslikavaju funkcije jezgara na ulazne i/ili izlazne podatke
- Primeri:
  - Opisivanje niza dužine 1024 elementa

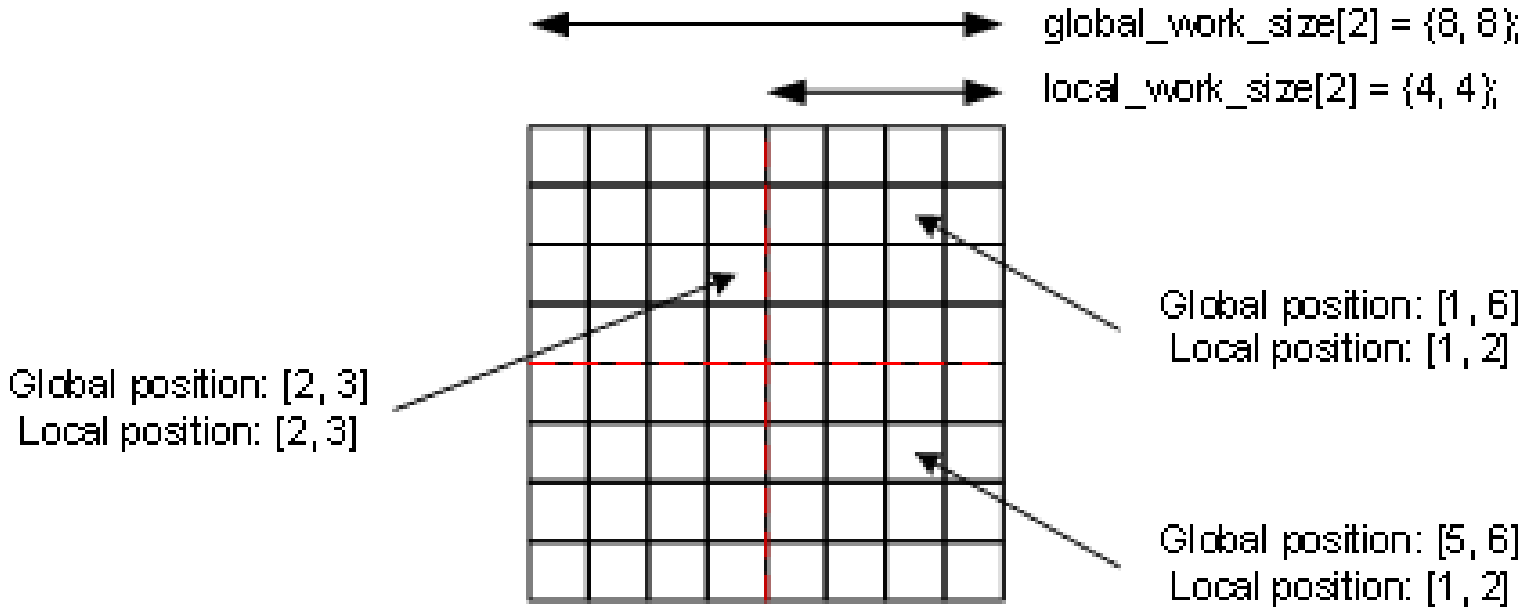
```
size_t n_dim = 1;
size_t global_work_size[1] = {1024};
size_t local_work_size[1]= {64}; // veličina radne stavke

size_t n_dim = 2;
size_t global_work_size[2] = {1024, 1};
size_t local_work_size[2]= {64, 1}; // veličina radne stavke
```

OpenCL *N*-dimenzioni opseg (NDO)

- Primeri:
  - Opisivanje matrice dimenzije 8x8

```
size_t n_dim = 2;
size_t global_work_size[2] = {8, 8};
size_t local_work_size[2]= {4, 4}; // veličina radne grupe
```



OpenCL intrizičke funkcije

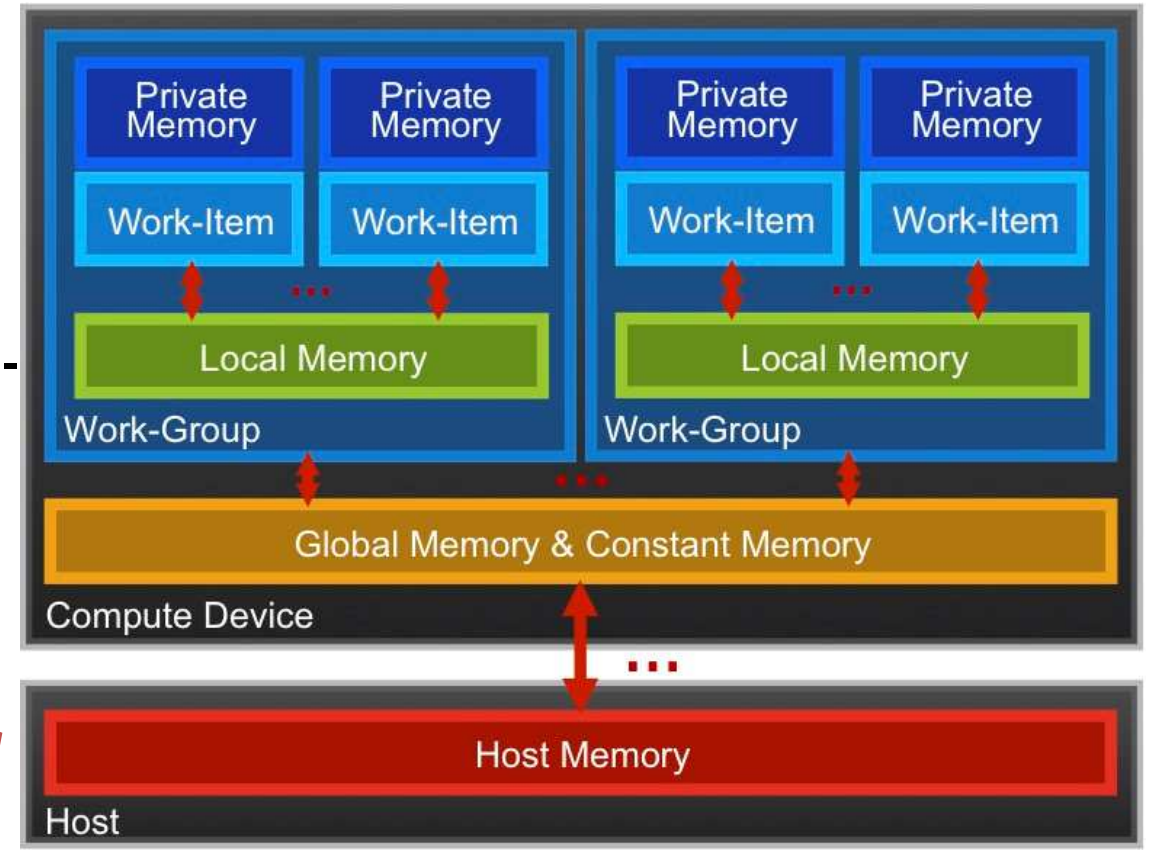
- OpenCL intrizičke funkcije se pozivaju unutar OpenCL jezgra i omogućuju da jezgro odnosno radna stavka indetifikuje sebe unutar globalnog i lokalnog prostora indeksa:
  - uint *get_work_dim* ()
  - size_t *get_global_size* (uint)
  - size_t *get_global_id* (uint)
  - size_t *get_local_size* (uint)
  - size_t *get_local_id* (uint)
  - size_t *get_num_groups* (uint)
  - size_t *get_group_id* (uint)
  - size_t *get_global_offset* (uint)

OpenCL *N*-dimenzioni opseg (NDO)

- Radne-stavke unutar radne-grupe mogu da pristupe istoj **lokalnoj(deljenoj)-memoriji** i mogu biti **sinhronizovane**
- Definisanje broja radnih-stavki unutar radne-grupe (skalabilnost)
- Ukoliko lokalni opseg (tj. opseg radne-grupe) nije definisan OpenCL okruženje će definisati veličinu (moguće neoptimalno)

Memorijski model OpenCL platforme

- ***Privatna memorija***
  - Dodeljena svakoj radnoj stavki
- ***Lokalna memorija***
  - Deljena unutar radne-grupe
- ***Globalna memorija***
  - Vidljiva svim radnim-grupama
- ***Memorija domaćina***
  - Memorija računara

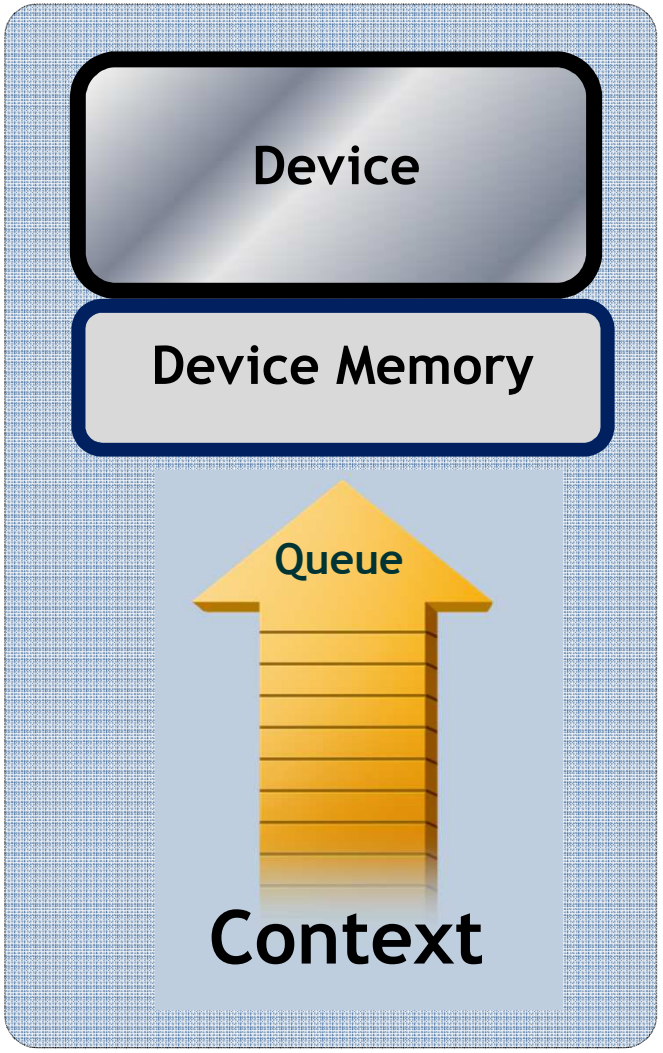


Rukovanje memorijom (*memory management*) je <u>eksplicitan</u>:
Korisnik ručno upravlja prenosom podataka:

domaćin → globalna → lokalna *i* nazad

Kontekst (*context*) i komandni red (*command-queue*)

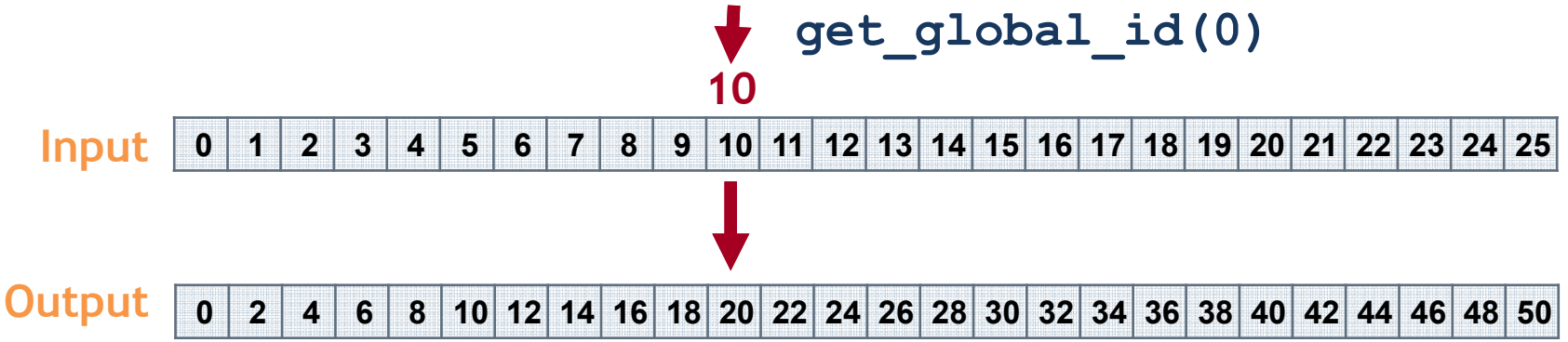
- ***Kontekst***:
  - Okruženje unutar kojeg se izvršava (i)funkcija jezgra i sinhronizacija, i (ii)nad kojim je definisano rukovanje memorijom.
- Kontekst uključuje:
  - Jedan ili više uređaja
  - Memoriju uređaja
  - Jedan ili više komandnih redova (*commnad-queue*)
- Sve ***komande*** za upravljanje uređajem (izvršavanje jezgra, sinhronizacija, naredbe za prenos memorije) su poslate kroz ***komandni red***.
- Unutar konteksta svaki ***komandni red*** opslužuje samo jedan uređaj.



Model izvršenja i funkcije jezgra (*kernels*)

- OpenCL model izvršenja definiše opseg problema i izvršava instancu funkcije jezgra (***kernel***) nad svakim delom tog opsega

```
__kernel void times_two(
    __global float* input,
    __global float* output)
{
    int i = get_global_id(0);
    output[i] = 2.0f * input[i];
}
```

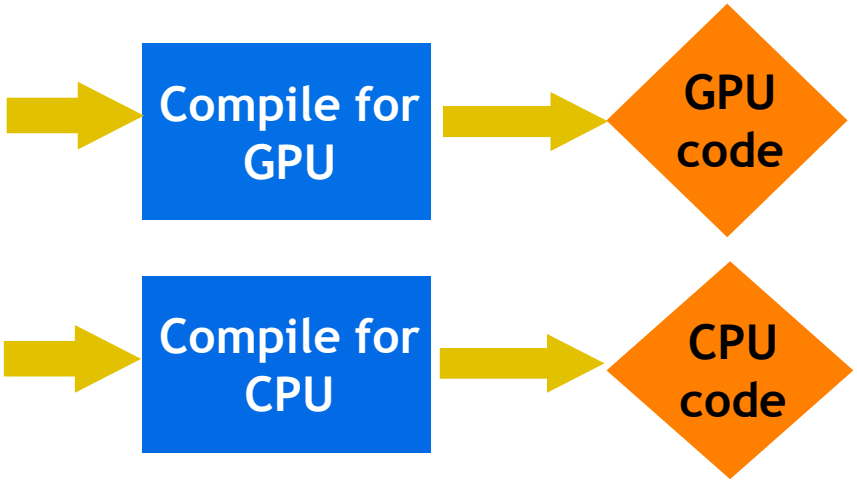


Pravljenje OpenCL programskog objekta

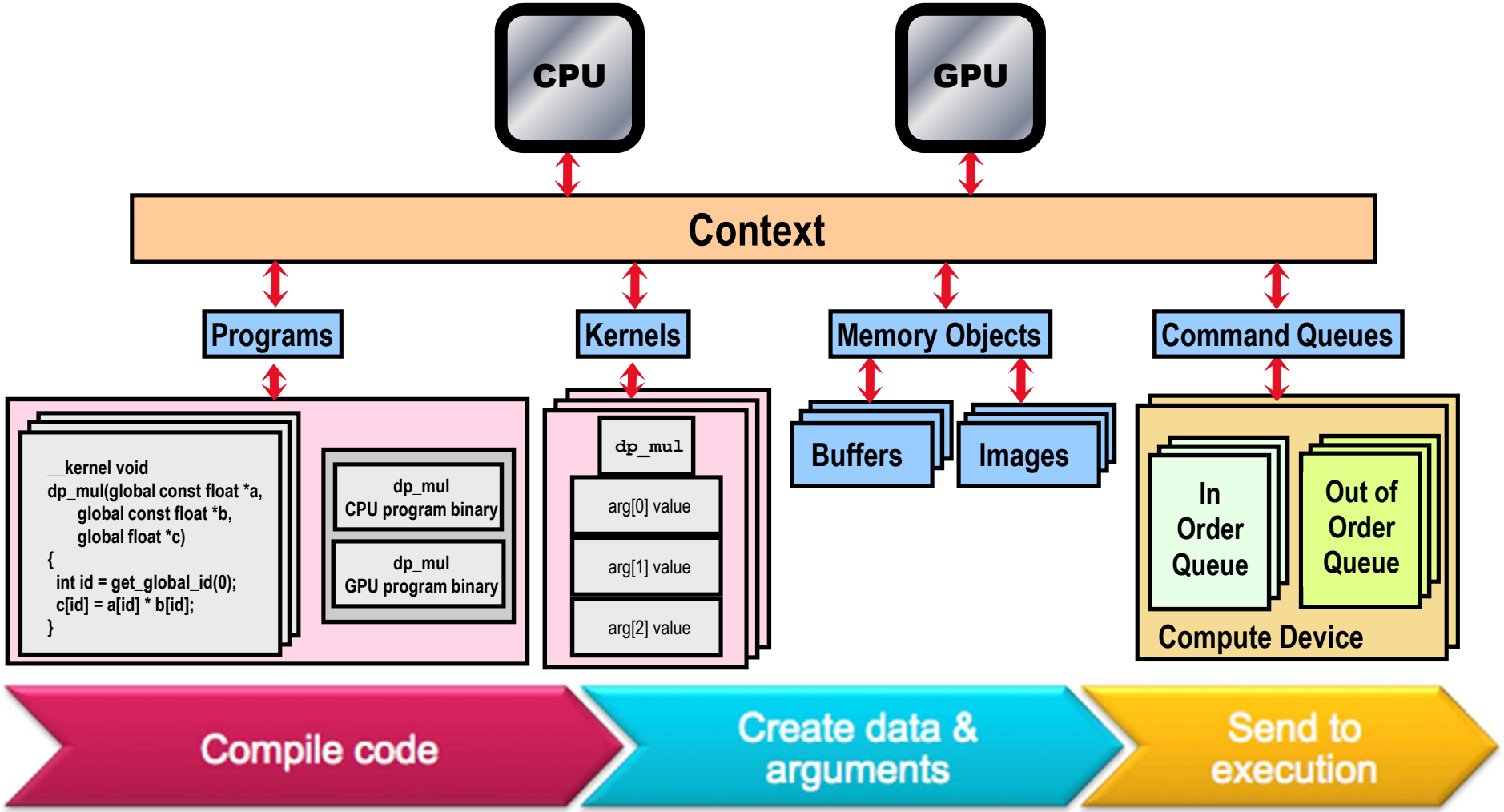
- **`Programski objekt`** enkapsulira:
  - Kontekst
  - Funkciju jezgra (binarni objekt ili izvorni kod)
  - Lista ciljanih uređaja za izvršavanje
  - Opcije
- OpenCL C\C++ API za pravljenje programskog objekta koristi sledeče funkcije:
  - **`clCreateProgramWithSource()`**
  - **`clCreateProgramWithBinary()`**

OpenCL koristi dinamičko prevođenje (*runtime compilation*) programskog koda jer u opštem slučaju ne znamo na kojoj će se ciljnoj platformi izvršavati program.

```
__kernel void
horizontal_reflect(read_only image2d_t src,
                   write_only image2d_t dst)
{
  int x = get_global_id(0);  // x-coord
  int y = get_global_id(1);  // y-coord
  int width = get_image_width(src);
  float4 src_val = read_imagef(src, sampler,
                               (int2)(width-1-x, y));
  write_imagef(dst, (int2)(x, y), src_val);
}
```



Osnovna platforma i OpenCL C\C++ API



Primer: Sabiranje vektora

Primer: Sabiranje vektora

- Sabiranje dva vektora je „hello world“ program paralelne obrade podataka

```
C[i] = A[i] + B[i] za i=0 to N-1
```

- U OpenCL rešenju definišu se dva dela
  - Deo jezgra (funkcija jezgra se izvršava na OpenCL uređaju - CPU, GPU, ubrzivač)
  - Deo domaćina (izvršava se na računaru - CPU)

Sabiranje vektora - Deo jezgra

```
__kernel void vectorAdd(
                        __global int *A,
                        __global int *B,
                        __global int *C)
{
    int idx = get_global_id(0);
    C[idx]  = A[idx] + B[idx];
    return;
}
```

Sabiranje vektora – Deo domaćina

- Izvršavanje programa domaćina (*host program*) omogućuje:
  - Pripremu okruženja za OpenCL program (koj se izvršava na uređaju)
  - Stvaranje i upravljanje funkcijama jezgra
- 5 osnovnih koraka u programu domaćina:
  1. Definisati *platformu* (uređaji + kontekst + redovi komandi)
  2. Napraviti i podesiti (prebaciti) *memorijske objekte*
  3. Napraviti i prevesti *OpenCL program* (dinamičko prevođenje)
  4. Definisati *funkcije jezgra* i prudružiti im odgovarajuće argumente
  5. Zadati *komande* (definisati broj radnih stavki i radnih grupa, izvršiti funkcije jezgra)

Sabiranje vektora - Implementacija

- Implementacioni koraci (*VectorAddition*):
  1) Zauzimanje memorije (inicijalizacija)
  2) Očitavanje dostupnih OpenCL platformi
  3) Odabir OpenCL pltaforme
  4) Očitavanje dostupnih OpenCL uređaja
  5) Odabir OpenCL uređaja
  6) Pravljenje konteksta
  7) Pravljenje komandnog reda
  8) Pravljenje OpenCL memorijskih objekata
  9) Prebacivanje OpenCL memorijskih objekata u OpenCL uređaje

Sabiranje vektora - Implementacija

- Implementacioni koraci (*VectorAddition*):
  10) Pravljenje i prevođenje OpenCL programskog objekta
  11) Pravljenje funkcija jezgra
  12) Pridruživanje argumenata funkcijama jezgra
  13) Definisanje broja radnih stavki i radnih grupa
  14) Izdavanje naredbi OpenCL uređajima
  15) Preuzimanje rezultata sa OpenCL uređaja
  16) Provera rezultata
  17) Oslobađanje memorije

Rukovanje greškama

- Izuzetno je važno proveravati povratne vrednosti OpenCL API

- Proveri povratnu vrednost funkcije i učitaj poruku o grešci:

```
if (err != CL_SUCCESS) {
 size_t len;
 char buffer[2048];
 clGetProgramBuildInfo(program, device_id,
  CL_PROGRAM_BUILD_LOG, sizeof(buffer), buffer, &len);
 cout << buffer << endl;
}
```

- Moguće je koristiti C++ try/catch mehanizam

Dodatni materijali

- Knjiga Paralelno programiranje, poglalje 3 (Paralelno programiranje sa OpenCL)

- OpenCL dokumentacija: vodiči za programiranje, priručnici, specifikacija (https://www.khronos.org/opencl/)

Zadatak

- Na osnovu primera za sabiranje vektora napisati OpenCL program za sabiranje dve matrice [A] = [A] + [B]
- Iskoristiti priloženi projekat *MatrixAddition*
- Koje korake možemo iskoristiti?
- Koje korake moramo da prepravimo?