Arhitektura računala 2 – Četvrta laboratorijska vježba

# 1 Priprema – pitanja za vježbu

## 1. Čemu služi do { ... } while(0); u definiciji makroa?

Takva konstrukcija se koristi u definiciji makroa kada makro pozivamo kao da je funkcija. To je jedina konstrukcija koja osigurava da se makro ponaša kao funkcija bez obzira gdje se poziva, pogotovo u situaciji kada se poziva unutar if-petlje bez vitičastih zagrada.

## 2. Čemu služi makro CL\_CHECK?

CL\_CHECK provjerava povratnu vrijednost OpenCL izraza koji mu je predan i za nas automatski nastavlja program u slučaju uspjeha, ili prekida program i ispisuje grešku u slučaju pogreške.

## 3. U kojem trenutku se pokreće prevođenje jezgrene procedure u OpenCL-u?

Prevođenje jezgrene procedure pokreće se pozivom funkcije clBuildProgram.

## 4. U kojem trenutku se pokreće izvođenje jezgrene procedure u OpenCL-u?

Nakon što je jezgrena procedura zajedno s poslom dodana u red čekanja pozivom funkcije clEnqueueNDRangeKernel, pozvat će se čim ostali poslovi dovrše i naš posao dođe na red.

## 5. Kako OpenCL prevoditelj “zna” treba li jezgru (jezgrenu proceduru, engl. kernel function) prevesti za Intelov ili AMD-ov GPU?

Kontekst koji se predaje pozivu clCreateProgramWithSource nas traži da predamo napravu s kojom želimo raditi.

## 6. Čemu služi poziv funkcije get\_global\_id? Izvodi li se ta funkcija na CPU-u ili na GPU-u? O čemu ovisi maksimalna povratna vrijednost?

get\_global\_id poziva se na GPU-u kako bi se dohvatio trenutni indeks elementa na kojem paralelni program radi. U slučaju višedimenzionalnog rada (work\_dim > 1), parametrom se može dohvatiti indeks na određenoj razini ugniježđenosti.

## 7. Kojim funkcijama ostvarujemo prijenos podataka iz radne memorije u memoriju GPU-a i obratno?

¸ Prijenos podataka iz radne memorije u memoriju GPU-a ostvarujemo funkcijom clEnqueueWriteBuffer, a obratno funkcijom clEnqueueReadBuffer.

## 8. Koje funkcije zauzimaju i otpuštaju memoriju GPU-a?

Funkcija clCreateBuffer zauzima memoriju GPU-a, a funkcija clReleaseMemObject ju otpušta.

## 9. Koji parametar određuje koliko puta će se izvršiti jezgrena procedura?

Parametri global\_work\_size i work\_dim određuju koliko puta će se izvršiti jezgrena procedura, na način da work\_dim zadaje broj dimenzija (ugniježđenih brojača) za posao koji treba odraditi, a global\_work\_size je niz od work\_dim cijelih brojeva koji predstavlja dimenzije (krajnje vrijednosti brojača).

## 10. Kako se provodi prebacivanje parametara iz programa u C-u u jezgrenu proceduru?

Parametre prebacujemo iz C-a u jezgrenu proceduru pomoću funkcije clSetKernelArg.

## 11. Na koji način zadajemo pokretanje jezgrene procedure?

Pokretanje jezgrene procedure zadajemo pozivom funkcije clEnqueueNDRangeKernel.

# 2 Zadatci

## 1. Napišite naivnu implementaciju matričnog množenja u C-u. Neka dimenzije ulaznih matrica budu i , pri čemu su , i parametri.

void mat\_mul(float\* R, float\* A, float\* B, int n, int m, int k) {

    for (int i = 0; i < n; i++) {

        for (int j = 0; j < k; j++) {

            float sum = 0;

            for (int l = 0; l < m; l++) {

                sum += A[i \* m + l] \* B[l \* k + j];

            }

            R[i \* k + j] = sum;

        }

    }

}

## 2. Isprobajte vašu funkciju na malim matricama i ručno provjerite točnost rezultata.

R[2x2] = {

    22 28

    49 64

}

B[3x2] = {

    1 2

    3 4

    5 6

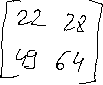
}

A[2x3] = {

    1 2 3

    4 5 6

}



## 3. Napišite izvedbu nove funkcije za množenje kvadratnih matrica koja izvođenje provodi na grafičkom procesoru (GPU) upotrebom odgovarajuće jezgrine funkcije. Provjerite točnost vaše izvedbe usporedbom s rezultatima prethodno razvijene izvedbe u C-u.

\_\_kernel void mat\_mul(\_\_global float\* R, \_\_global float\* A, \_\_global float\* B, int m, int k)

{

    int i = get\_global\_id(0);

    int j = get\_global\_id(1);

    float sum = 0;

    for (int l = 0; l < m; l++) {

        sum += A[i \* m + l] \* B[l \* k + j];

    }

    R[i \* k + j] = sum;

}

CPU\_S[5x8] = {

    924 952 980 1008 1036 1064 1092 1120

    2149 2226 2303 2380 2457 2534 2611 2688

    3374 3500 3626 3752 3878 4004 4130 4256

    4599 4774 4949 5124 5299 5474 5649 5824

    5824 6048 6272 6496 6720 6944 7168 7392

}

GPU\_S[5x8] = {

    924 952 980 1008 1036 1064 1092 1120

    2149 2226 2303 2380 2457 2534 2611 2688

    3374 3500 3626 3752 3878 4004 4130 4256

    4599 4774 4949 5124 5299 5474 5649 5824

    5824 6048 6272 6496 6720 6944 7168 7392

}

## 4. Iscrtajte grafove vremena izvođenja obje izvedbe na slučajno inicijaliziranim kvadratnim matricama 32-bitnih brojeva s pomičnim zarezom uz . Mjereno vrijeme treba uključivati prijenos matrica iz RAM-a u GPU memoriju i natrag, ali ne i prevođenje jezgrene procedure.

## 5. Profilirajte vrijeme izvođenja izvedbe za grafički procesor po odsječcima: prijenos RAM GPU, računanje rezultata, prijenos GPU RAM.

## 6. Procijenite brzinu prijenosa RAM GPU i GPU RAM u GB/s te ostvarenu brzinu računanja na GPU-u (TFLOPS). Usporedite dobivene veličine s brzinom prijenosa RAM RAM te ostvarenom brzinom računanja na CPU-u.

Benchmarking

RAM -> GPU took 3.588 s for 20.00 GB (5.575 GB/s)

GPU -> RAM took 3.275 s for 20.00 GB (6.107 GB/s)

Prijenos RAM RAM ispitivali smo u sklopu zadatka 2b na drugoj laboratorijskoj vježbi, a brzina prijenosa iznosila je .

Za svaki od elemenata rezultantne matrice, naša jezgrena procedura će izvršiti floating-point operacija množenja i floating-point operacija zbrajanja. Tako za slučaj množenja kvadratnih matrica s proizvoljnom veličinom , ukupan broj floating-point operacija možemo izračunati ovako:

Broj floating-point operacija u sekundi možemo onda dobiti dijeljenjem očekivanog broja floating-point operacija s vremenom koje je računalu trebalo da izračuna umnožak matrica.