OpenCL, I deo: Model paralelnog programiranja

- Platforma i uređaji
- Izvršno okruženje
- Model memorije
- OpenCL jezgro

Uvod (1/2)

- Postoje dve vrste arhitektura:
  - Homogene – npr. CPU sa više jezgara
  - Heterogene – sa mešovitim komponentama (CPU, GPU, FPGA)
- Aplikacije sa različitim radnim opterećenjima:
  - Sa intenzivnim upravljačkim operacijama
  - Sa intenzivnim operacijama nad podacima
  - Sa intenzivnim računanjima
- Heterogene arh. podržavaju sva opterećenja
  - OpenCL je okruženje za programiranje heterogenih arhitektura

Uvod (2/2)

- OpenCL obezbeđuje:
  - Jezik za programiranje uređaja (eng. device-side)
  - API na domaćinu (eng. host) za upravljanje sistemom
- OpenCL podržava korišćenje šablona:
  - Paralelizam zadataka
  - Geometrijska dekompozicija
- Na arh. sa CPU i GPU, moguće rasteretiti CPU:
  - tzv. JEZGRO OBRADE (eng. kernel) na GPU
  - CPU izvršava manje intenzivne upravljačke delove

OpenCL model paralelnog programiranja (1/3)

- Osobine OpenCL
  - Uopštenost: podržava značajno različite arhitekture – radi prenosivosti programa
  - Prilagodivost: svaka fizička platforma i dalje može da obezbedi visoku performansu
- OpenCL specifikacija se sastoji od četiri modela:
  - MODEL PLATFORME
  - MODEL IZVRŠENJA
  - MODEL MEMORIJE
  - MODEL PROGRAMIRANJA

OpenCL model paralelnog programiranja (2/3)

- MODEL PLATFORME:

  Jedan procesor (*domaćin*) koordinira izvršenje i više procesora izvršava OpenCL C kod (*uređaji*)

- MODEL IZVRŠENJA:
  - Definiše kako treba konfigurisati OpenCL okruženje na domaćinu i kako se jezgra izvršavaju na uređajima
- MODEL MEMORIJE:
  - Definiše apstraktnu hijerarhiju memorije koju koriste jezgra; liči na hijerarhije memorija tekućih GPU

OpenCL model paralelnog programiranja (3/3)

- MODEL PROGRAMIRANJA:
  - Definiše kako se model konkurencije preslikava na fizičke komponente sistema
- KONTEKST za rukovanje izvršenjem
  - Komande za prenos podataka domaćin-uređaji
  - Komande za izvršenje jezgra (obradu podataka)
- OpenCL model je sintaksno sličan standardnoj C funkciji, ključna razlika je u modelu izvršenja

Primer: sabiranje vektora

```
// Saberi elemente A i B i smesti ih u C.
// Svaki niz ima N elementa.
void vecadd(int *C, int* A, int *B, int N) {
    for(int i = 0; i < N; i++) {
        C[i] = A[i] + B[i];
    }
}
```

- Serijski kod u jeziku C

```
// Saberi elemente A i B i smesti ih u C.
// Svaki niz ima N elementa a CPU ima NP jezgara.
void vecadd(int *C, int* A, int *B, int N, int NP, int tid)
{
    int ept = N/NP; // elemenata po niti
    for(int i = tid*ept; i < (tid+1)*ept; i++) {
        C[i] = A[i] + B[i];
    }
}
```

- Paralelno sabiranje vektora u jeziku C sa POSIX/Win32 nitima

Radna-stavka

- Organizacija paralelne operacije nad podacima:
  - RADNA-STAVKA = jedinica konkurentnog izvršenja
  - Svaka radna-stavka izvršava telo funkcije jezgra
  - Biblioteka preslikava iteracije na radne-stavke
- Ovo liči na paralelne **for** petlje u Cilk i TBB
- Postoje intrinističke funkcije pomoću kojih radna-stavka može sebe da identifikuje
  - get_global_id(0): pozicija tekuće radne-stavke, tj. odgovarajuća vrednosti brojača petlje

Primer: sabiranje vektora u OpenCL

- Paralelno sabiranje vektora u jeziku OpenCL C:

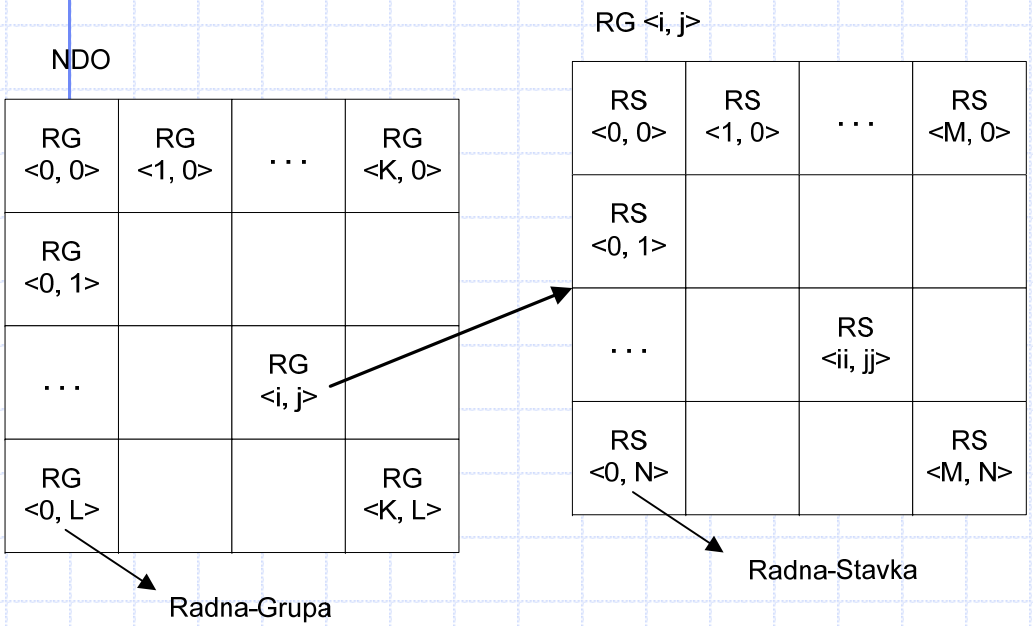
```
// Saberi elemente A i B i smesti ih u C.
// Biće napravljeno N radnih-stavki da izvršava ovo jezgro.
__kernel
void vecadd(__global int *C, __global int* A, __global int *B) {
    int tid = get_global_id(0); // OpenCL intrinistička funkcija
    C[tid] = A[tid] + B[tid];
}
```

- Radna-stavka je jedinica fine granularnosti
- Biblioteka može pokrenuti ogroman broj radnih-stavki
- Skalabilnost?
  - Garantuje je hijerarhijski model konkurentnosti

N-Dimenzioni Opseg (NDO) radnih-stavki

- U cilju opisa načina izvršenja jezgra, zadaje se br. potrebnih radnih-stavki u obliku NDO
- NDO je 1, 2, ili 3-dimenzioni prostor indeksa, koji se najčešće preslikava na ul. ili izl. podatke
- Dimenzije NDO se zadaju kao nizovi sa N elem., gde je N br. dimenzija prostora radnih-stavki
  - Primer: NDO za obradu vektora sa 1024 elementa
    size_t indexSpaceSize[3] = {1024, 1, 1};

Radna-grupa (1/2)

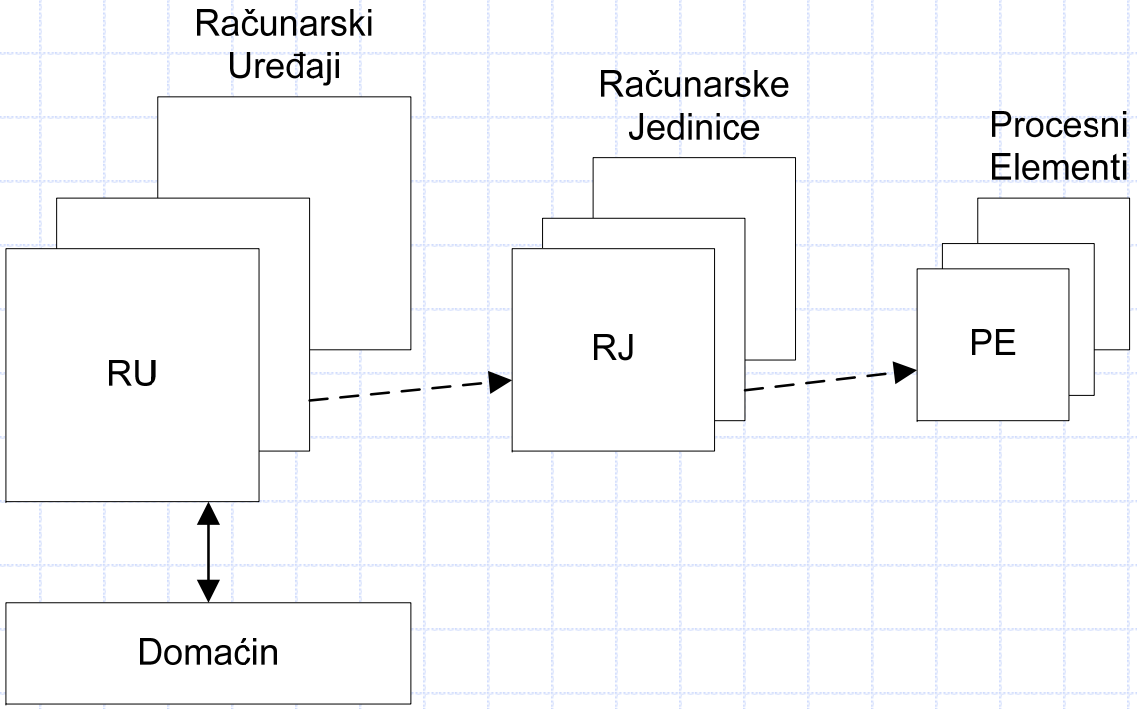


- Skalabilnost: podelom datog NDO na manje RADNE-GRUPE iste veličine
  - Prostor indeksa sa *N* dimenzija zahteva da radne-grupe budu takođe sa tih *N* dimenzija
  - Npr. 2-dimenzioni prostor indeksa zahteva 2-dimenzione radne-grupe

Radna-grupa (2/2)

- Radne-stavke unutar iste radne-grupe:
  - Mogu koristiti barijerne operacije radi sinhronizacije
  - Dele zajednički memorijski adresni prostor
- Pošto su veličine radnih-grupa fiksne, komunika-cija unutar radne-grupe ne mora da se skalira
- Za dati primer veličina radne-grupe može biti size_t workGroupSize[3] = {64, 1, 1};
  - Napomena: veličine prostora indeksa moraju biti deljive bez ostatka sa veličinama radnih-grupa po svakoj dimenziji.

Model platforme



- Analogija sa GPU
- AMD Radeon 6970 GPU ima 24 SIMD jezgra (RJ), svako jezgro ima 16 SIMD traka (PE) a svaka traka ima VLIW sa 4 operacije; total 1536 instrukcija

- Platforma = implementacija OpenCL API
- Uređaji koje platforma može da cilja su ograničeni na one sa kojima platforma proizvođača ume da rukuje

API funkcije za platformu i uređaje

- clGetPlatformIDs(ne, *pl, *np)
  - 1-poziv: ulaz: ne=0, pl=NULL, izlaz *np=br.platformi
  - 2-poziv: ulaz ne=*np, pl=malloc(), np=NULL, izlaz: *pl=info o svim raspoloživim platformama
- clGetDeviceIDs(pl_id, dt, ne, *dv, *nd)
  - 1-poziv: ulaz: ne=0, dv=NULL, izlaz *nd=br.uređaja
  - 2-poziv: ulaz ne=*nd, dv=malloc(), dt=tip-uređaja, izlaz: *dv=info o svim raspoloživim uređajima
    - dt može da ograniči uređaje samo na GPU, samo na CPU, itd.

Izvršno okruženje (1/2)

- Domaćin prvo mora konfigurisati svoj kontekst, putem kog uređaju šalje komande i podatke
- KONTEKST je apstraktni kontejner koji:
  - Koordinira mehanizme interakcije domaćin-uređaj
  - Rukuje mem. objektima, koji su raspoloživi za uređaje
  - Vodi evidenciju o programima za svaki uređaj
- API funkcije za pravljenje konteksta:
  - clCreateContext – za zadate uređaje
  - clCreateContextFromType – za uređaje datog tipa

Izvršno okruženje (2/2)

- Parametri funkcije clCreateContext:
  - Sfera interesa (eng. scope) konteksta
  - Broj uređaja i njihove identifikacije (ID)
  - Funkcija povratnog poziva (eng. callback)
- Funkcija clGetContextInfo se može koristiti za dobijanje informacije o:
  - broju prisutnih uređaja i
  - o strukturama uređaja

Komandni red

- Unutar konteksta mora da se napravi po jedan KOMANDNI RED za svaki uređaj
- Parametri funkcije clCreateCommandQueue:
  - Kontekst
  - Uređaj
  - Svojstva komandnog reda
    - Omogućeno profilisanje komandi
    - Dozvoljeno izvršenje komandi van redosleda (eng. out-of-order)
    - itd.

Funkcije za ulančavanje komandi

- API funkcije sa prefiksom clEnqueue
  - clEnqueueWriteBuffer – upis podataka u uređaj
  - clEnqueueNDRangeKernel – započni izvršenje jezgra
  - clEnqueueReadBuffer – čitanje podataka sa uređaja
- Komande proizvode događaje, koji služe za:
  - Predstavljanje zavisnosti
  - Obezbeđuju mehanizam za profilisanje
- Zavisnosti:
  - Događaj/lista-događaja se prosleđuje kao parametar sledećeg poziva API funkcije sa prefiksom clEnqueue

MEMORIJSKI OBJEKTI

- Dve vrste memorijskih objekata:
  - Baferi (eng. buffers): kao nizovi u C-u, prave se sa malloc, podaci smešteni u susedne mem. lokacije
  - Slike (eng. images): objekti nepoznatog tipa (eng. opaque), što omogućava razne optimizacije
- Parametri funkcije clCreateBuffer:
  - Kontekst (bafer postoji samo u jednom kontekstu)
  - Zastavice (samo za čitanje, samo za upis, itd.)
  - Veličina bafera
  - Domaćinov pokazivač na podatke kojim treba inicijalizovati bafer

Upis i čitanje iz bafera

- Parametri funkcija za upis/čitanje bafera:
  - Komandni red
  - Pokazivač na bafer
  - Vrsta operacije: blokirajuća/neblokirajuća (sinhron ili asinhron poziv)
  - Pokazivač na podatke za inicijalizaciju bafera
  - Broj elemenata u listi čekanja
  - Pokazivač na listu čekanja
  - Pokazivač na rezultantni (povratni) događaj

Slike

- Omogućavaju optimizacije specifične za uređaje
  - Opcije se dobijaju pomoću clGetDeviceInfo
  - Slike se ne mogu direktno referencirati kao da su nizovi, jer ne moraju biti smešteni u susedne lokacije
- Format deskriptor slike (uvodi koncept KANALA):
  - REDOSLED KANALA: broj elem. koji čine elemente slike (do 4 elementa, na osnovu RGBA piksela)
  - TIP KANALA: veličina svakog elementa, 1 do 4 bajta, celobrojno ili u pokretnom zarezu

Baratanje slikama

- Pravljenje: clCreateImage2D, clCreateImage3D
  - Kao bafer + dodatni parametri: širina, visina i dubina slike (3D), razmak između dva reda (eng. pitch), itd.
- Čitanje/upis slika kao za bafere + param regiona (3 elem.) za proširenje podataka pri prenosu
- Unutar jezgra:
  - Funkcija read_imagef za čitanje pod. u pok. zarezu
  - Funkcija read_imageui za čitanje neoz. cel. podataka
  - Za čitanje potreban objekt ODABIRAČ (eng. sampler)

Pražnjenje i završetak

- PRAŽNJENJE i ZAVRŠETAK su operacije na dva različita tipa barijera:
  - Fun. clFinish se blokira sve dok se sve komande u komandnom redu ne završe (sinhron poziv)
  - Fun. clFlush se blokira sve dok sve komande iz komandnog reda ne budu uklonjene iz reda
    - Ali, izvršenje svih komandi ne mora biti završeno!
  - Param obe funkcije je komandni red

Pravljenja jezgra

- PROGRAM je OpenCL C kod upisan u uređaj
  - Program je zbirka funkcija, koje se nazivaju jezgrima
  - Prevodi se u fazi izvršenja pomoću niza API funkcija
  - Omogućene su optimizacije specifične za uređaj
- Postupak pravljenja jezgra u 3 koraka:
  - Izvorni kod u OpenCL C-u se upiše u niz karaktera
  - Fun. clCreateProgramWithSource pretvori izvorni kod u programski objekat (tipa cl_program)
  - Fun. clBuildProgram prevodi programski objekat za jedan ili više uređaja

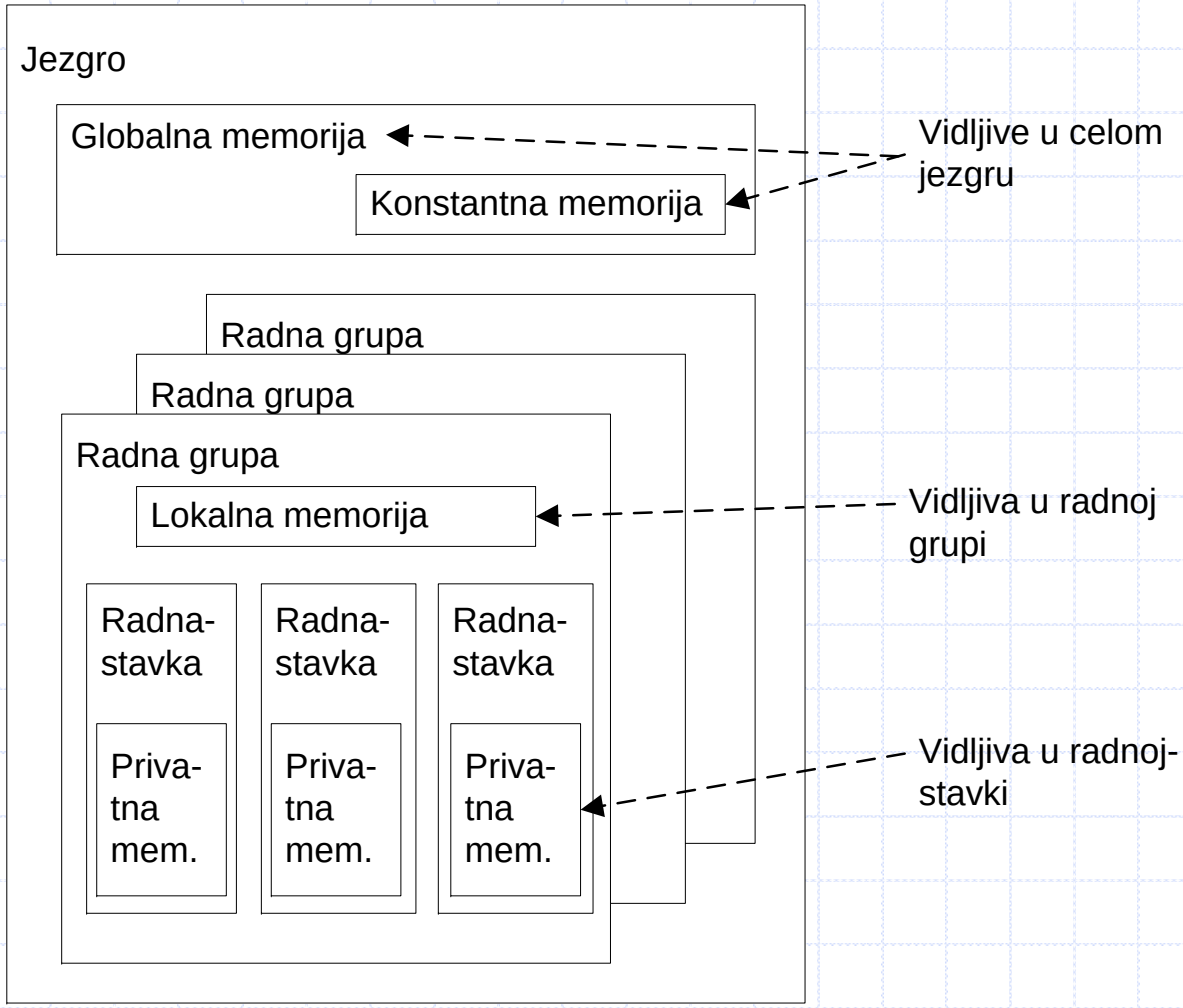
Izdvajanje jezgra i postavljanje parametara jezgra

- Poslednja faza u pravljenju jezgra je izdvajanje jezgra iz programskog objekta
  - To radi funkcija clCreateKernel
- Pre pokretanja moraju se postaviti parametri
  - Za razliku od C f-ije, parametri jezgra su perzistentni
  - Postavljaju se funkcijom clSetKernelArg
    - Parametri ove funkcije su jezgro, indeks stvarnog parametra jezgra, veličina i pokazivač na stvarni parametar jezgra
    - Prilikom izvršenja, izvršna biblioteka raspakuje podatke (eng. unbox) u odgovarajući tip

Pokretanje jezgra

- Parametri funkcije clEnqueueNDRangeKernel:
  - Komandni red, jezgro,
  - Br. dimenzija (1, 2 ili 3) prostora radnih-stavki
  - Br. radnih-stavki po svakoj dimenziji NDO
  - Br. radnih-stavki po svakoj dimenziji radne-grupe
  - Parametrom global_work_offset se mogu zadavati globalne ID radnih-stavki koje ne kreću od 0
  - Br. događaja u listi čekanja, lista čekanja, itd.
- Komanda clEnqueueNDRangeKernel je asinhrona
  - Čekanje kraja sa clWaitForEvents ili clFinish

Model memorije (1/2)



Model memorije (2/2)

- Hijerarhijski nivoi memorije:
  - GLOBALNA MEMORIJA: vidljiva svim RJ u uređaju; podaci koji se prebacuju između domaćina i uređaja
  - KONSTANTNA MEMORIJA: podaci kojim pristupaju sve radne-stavke; uključujući samo čitanje
  - LOKALNA MEMORIJA: dele je radne-stavke u radnoj grupi; pokazivači (parametri jezgra i lokalni) i nizovi
  - PRIVATNA MEMORIJA: jedinstvena za radnu-stavku; lokalni podaci i parametri jezgra koji nisu pokazivači

Pisanje OpenCL jezgara (1/4)

- Uputstvo:
  - Jezgro počinje rečju **__kernel** i mora vratiti tip **void**
  - Lista parametara kao za C f-iju, plus
    - Mora se specificirati adresni prostor svakog pokazivača
  - Bafere je moguće deklarisati u:
    - globalnoj (**__global**) ili konstantnoj memoriji (**__constant**)
  - Slike se pridružuju globalnoj memoriji
  - Kvalifikatori pristupa (opcioni; služe za optimizacije)
    - __read_only, __write_only, i __read_write

Pisanje OpenCL jezgara (2/4)

- Kvalifikator __**local** definiše memorijski prostor koji dele sve radne-stavke u radnoj grupi
  - Npr. "__local float *data" je pokazivač na lokalni niz
    - Prednost: proizvoljna dimenzija niza
  - Alternativa: deklarisanje promenljive na nivou jezgra
    - Nedostatak: dimenzija niza mora biti fiksna – primer ispod

```
__kernel void aKernel(. . .){
        // Dele sve radne-stavke u grupi
        __local float data[32];
        . . .
}
```

Pisanje OpenCL jezgara (3/4)

- Baferovanja podataka:
  - Koji su korišćeni više puta od strane jedne ili više radnih-stavki u istoj grupi (vremenska lokalnost pod.)
  - Postiže se eksplicitnom dodelom vrednosti globalnog pokazivača lokalnom (lok ptr = glob ptr), npr.:

```
__kernel void cache(
        __global float *data,
        __local float *sharedData) {
        int globalId = get_global_id(0);
        int localId = get_local_id(0);
        // Baferuj podatke u lokalnoj memoriji
        sharedData[localId] = data[globalId];
        . . .
}
```

Pisanje OpenCL jezgara (4/4)

- Sadržaj lokalne memorije nije perzistentan, tj. gubi se po završetku radne-stavke
  - Zato je potrebno sve bitne rezultate preneti u globalnu memoriju pre završetka radne-stavke

- Domaći: Izučiti primer sabiranja vektora u knjizi