GP/GPU

…kak' to zapravo radi?

Motivacija

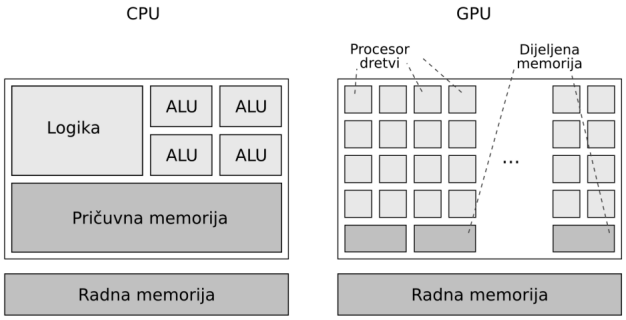
- GPU: hrpa raspoložive računalne snage koju se može iskoristiti

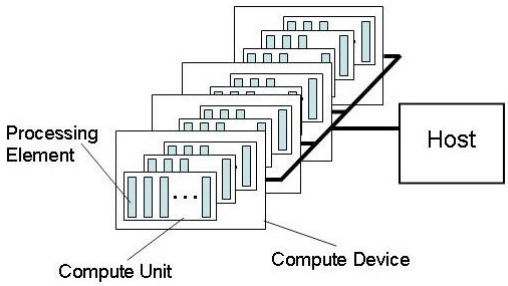
Pitanja:

1. *Što se unutra događa? (sklopovski model)*
2. *Kako to programirati? (programski model)*

1. Sklopovski model

- razlike CPU - GPU



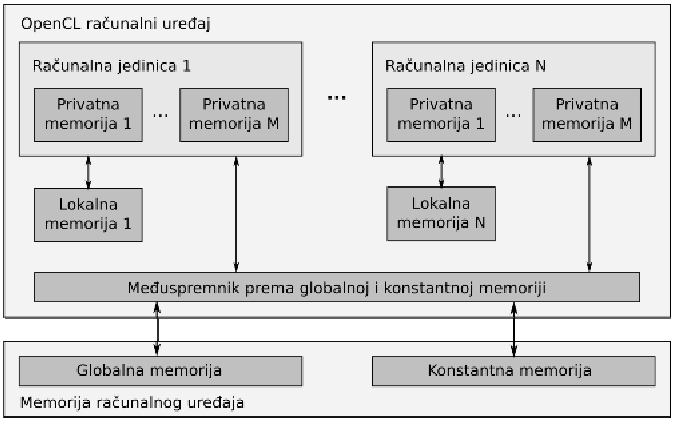


OpenCL
- host: PC
- računski uređaj (*compute device*): GPU, CPU, ...
- računska jedinica (*compute unit*): GPU procesor
- procesni element (*processing element*): GPU procesor dretve

CUDA
- host: PC
- *device*
- *multiprocessor*
- *scalar processor*

Memorijska hijerarhija

- (OpenCL nomenklatura)



Paralelni sustav dretvi

- SIMT model (*Single-Instruction, Multiple-Thread*)
- *host* stvara funkciju dretve – *kernel*
  - u pravilu samo jedna istovremeno na uređaju, ali moguće je i više
- *host* definira ukupni broj dretvi koje se trebaju izvesti
  - u koliko dretvi se kernel treba izvesti
- GPU raspoređuje dretve po GPU procesorima
- GPU procesor (multiprocesor) paralelno izvodi grupu dretvi
  - svaki procesni element unutar GPU procesora izvodi jednu dretvu

Koliko je to zapravo paralelno?

- NVIDIA [1]:
  - "The multiprocessor ... schedules and executes threads in groups of 32 parallel threads called *warps*." [1]
- AMD [2]:
  - "A *wavefront* executes a number of work-items in lock step"
  - veličina ovisna o uređaju, najčešće 64 (ili 32)
  - zaista 64 procesne jezgre??
    - zapravo 16 jezgri, uz protočno izvođenje jedne instrukcije u 4 ciklusa → 64
    - "16 processing elements execute the same instructions for four cycles, which effectively appears as a 64-wide compute unit in execution width"
- navedene grupe (*val*) dijele programsko brojilo, izvode jednake instrukcije u svakom ciklusu
- optimalno: ukupan broj dretvi višekratnik veličine vala!

Grananje?

- najveća učinkovitost je bez grananja
- u slučaju grananja dijela dretvi, sve grane se izvode slijedno! (po dijelovima vala)
- primjer desno:
  - aktivne dretve (bitovna maska) izvode granu A
  - maska se invertira, ostale dretve izvode granu B
- ukupno trajanje: A + B (ako barem jedna dretva divergira)
- Petlje?
  - trajanje izvođenja petlje (broj iteracija) je trajanje dretve s *najvećim brojem iteracija*

```
if(x)
{
// grana A
...
}
else
{
// grana B
...
}
```

Pristup globalnoj memoriji

- pristup globalnoj memoriji >> trajanje jedne instrukcije
  - dretve koje čekaju na pristup: *blokirane* dretve
- što ako su sve dretve u valu blokirane?
- → GPU procesor može održavati više valova dretvi!
- ako su sve dretve nekog vala blokirane, pokreće se val koji ima barem jednu *aktivnu* dretvu
  - "compute device uses number of wavefronts to hide memory access latencies having the scheduler switch the active wavefront whenever the current wavefront is waiting for a memory access to complete" [2]
- usput budi rečeno...
  - ako više dretvi *piše* u istu globalnu lokaciju, pisanje je slijedno, a rezultat je nedefiniran

Zamjena aktivnog vala

- a koliko košta pokretanje drugog vala? (*context switch*)
- ništa! jer se *sve* lokalne varijable svih dretvi čuvaju unutar lokalne memorije GPU procesora
  - "switching from one execution context to another has no cost" [1]
- ... uz koju posljedicu?

Lokalna memorija

podsjetnik:

GPU – globalna memorija
GPU procesor – lokalna memorija (~ priručna memorija)
procesor dretve – privatna memorija (~ registri)

- sustav mora *unaprijed* znati koliko dretvi može pokrenuti na jednom GPU procesoru
  - "number of warps that can reside on multiprocessor for a given kernel depends on the amount of registers and shared memory used by the kernel and memory available" [1]
- količina memorije koju koriste dretve mora biti unaprijed ograničena (tipično ~64KB na GPU procesoru)
- → nema dinamičkog zauzimanja memorije unutar dretvi!
- prevelik zahtjev za memorijom: greška prilikom pokretanja

2. Programski model

- što odabrati?
- low level:
  - CUDA
  - OpenCL
- pomoćne biblioteke
  - Magma, Thrust, OpenACC, PGI Accelerator, Stream, DirectCompute, Renderscript, C++ AMP, CUDAfy.NET...
  - izvedenice: npr. cuBLAS, cuFFT, cuSPARSE, cuRAND, NPP, CULA...

OpenCL!

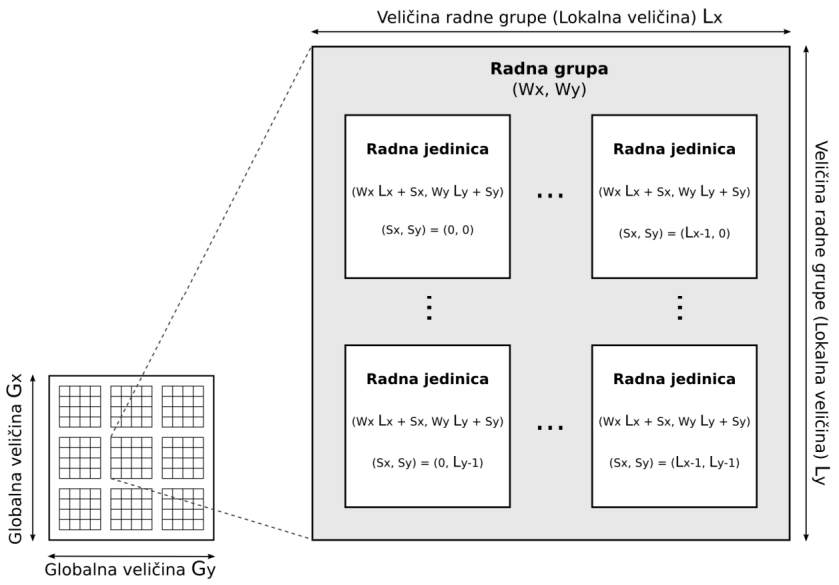
- zašto OpenCL?
  - prenosivost (Nvidia, AMD, Intel…)
  - učinkovitost jednaka ostalim alatima
  - isti programi (kerneli) mogu se pokrenuti na različitim platformama (GPU, CPU)
  - može raditi na mojoj kartici ;)
- gnjavaža:
  - kerneli se pišu u OpenCL C jeziku (ime.cl)
  - kerneli se prevode tijekom izvođenja programa (*runtime*) – zbog prenosivosti
  - za istu platformu moguće je koristiti prevedeni kernel

13/22

Podjela posla na dretve

- skup svih dretvi: *index range, NDRange*
  - globalna veličina skupa: G
  - svaka dretva ima jedinstveni globalni ID (0..G-1)
  - sve dretve idu na jedan uređaj (GPU)
- globalni skup dijeli se na *grupe dretvi* (*workgroup*)
  - lokalna veličina grupe: L
  - unutar grupe dretvi dretva ima lokalni ID (0..L-1)
  - grupa dretvi obično ide na jedan GPU procesor
  - grupa se jednoliko dijeli na valove (redom po 32/64 dretve)
  - definirati grupu kao višekratnik veličine vala!
- jedna dretva unutar grupe: *radna jedinica* (*work item*)
  - radnu jedinicu izvodi procesor dretve
- najjednostavnija podjela: 1D niz dretvi s globalnim indeksom
- česta primjena: podjela dretvi u 2D prostoru *zadataka*

14/22



Rad s podacima

- izvođenje dretvi obično uključuje:
  - pripremu ulaznih podataka na *hostu*
  - kopiranje na uređaj
  - izvođenje svih dretvi
  - kopiranje rezultata na *host*
- isplativost: trajanje izvođenja dretvi trebalo bi biti veće od trajanja prijenosa podataka

Sinkronizacija?

- sinkronizacija unutar vala dretvi – implicitna (*lock step*)
- unutar grupe dretvi – instrukcija ograde (*barrier*)
- između grupa dretvi – ? (nepotrebno)
- između različitih kernela – redovi izvođenja (definira se na hostu)

Jel to radi...?

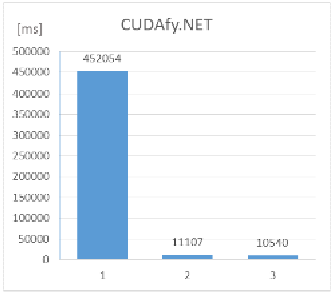
- clinfo
- hrpa primjera uz odgovarajuće *drajvere*
- primjer: računanje broja *pi* uzvraća udarac
  - ... ali uz float varijable baš i ne radi sasvim *točno*

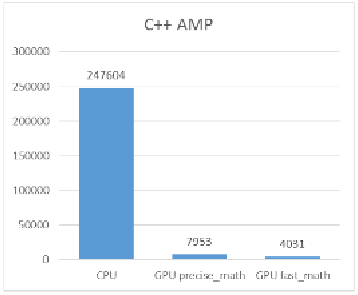
Nove stvari…

- APU = CPU + GPU
  - "…much of the functionality of CUDA and OpenCL, designed to copy memory from the CPU to/from the GPU, will be obsolete. " [3]
- C++ AMP (Accelerated Massive Parallelism)
  - uključivanje GPU funkcionalnosti unutar jezika (nema posebnih kernela!)
- CUDAfy.NET
  - C#, također izbjegavanje eksplicitnih funkcija dretvi

Neki rezultati (seminar 2014)

- isrcpna pretraga za TSP, računanje udaljenosti na GPU
- CUDAfy.NET: relativno ubrzanje ~42x
  - reduciranje max udaljenosti na GPU (scenarij 2) neisplativo
- C++ AMP: ubrzanje 30-60x
  - ali ne podržava 64-bitni integer...





Reference

1. CUDA C Programming Guide (http://docs.nvidia.com/cuda/cuda-c-programming-guide/)
2. AMD OpenCL Programming Guide (http://developer.amd.com/download/AMD_Accelerated_Parallel_Processing_OpenCL_Programming_Guide.pdf)
3. OpenCL vs. CUDA (http://codinggorilla.domemtech.com/?p=669#more-669)

Stranice

- http://developer.amd.com/resources/heterogeneous-computing/opencl-zone/programming-in-opencl/
- http://www.thebigblob.com/getting-started-with-opencl-and-gpu-computing/
- http://docs.nvidia.com/cuda/index.html
- http://www.khronos.org/registry/cl/sdk/1.2/docs/man/xhtml/
- http://opencl.codeplex.com/wikipage?title=OpenCL%20Tutorials
- http://openclnews.com/blog/opencl_tutorial_series_on_the_code_project_parts_1_8
- http://msdn.microsoft.com/en-us/library/hh265136.aspx
- http://w8isms.blogspot.com/2012/09/cudafy-me-part-1-of-4.html