Porazdeljeni sistemi

6. Računanje na grafičnih procesnih enotah

Predavatelj: izr. prof. Uroš Lotrič
Asistent: Davor Sluga

Začnimo ...

z lahkim uvodom ...

Razvoj računalniških iger

- Nekateri smo začeli spoznavati računalnik takole …

Razvoj računalniških iger

- … potem so stvari za nekaj časa postale slabše…

Razvoj računalniških iger

- … trend se je sredi 90. let počasi začel obračati na bolje …

Razvoj računalniških iger

- … boljše, še boljše, do današnjih že zelo realističnih iger.

Strojna oprema

- Zakaj so pokrajine v igrah lahko postale toliko bolj realistične, dogajanje v igri tako hitro?

Grafični pospeševalniki

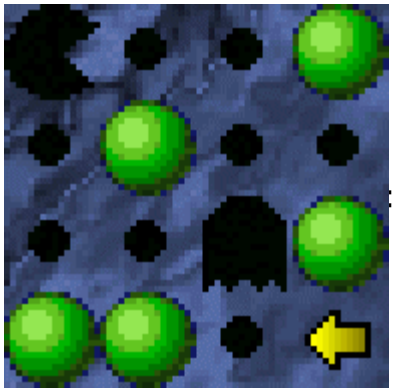
- Kaj so?
  - računalniška vezja z lastnim procesorjem in spominom, specializirana za prikazovanje računalniške grafike
    - grafične naloge opravijo hitreje od centralnih procesnih enot
    - hkrati razbremenijo centralne procesne enote
  - GPE = Grafična Procesna Enota
  - GPU = Graphics Processing Unit
- Pomembni izdelovalci
  - Nvidia,
  - ATI Technologies

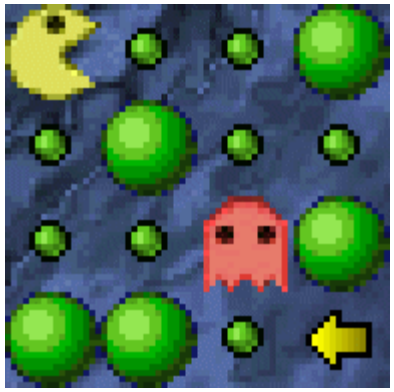
Grafični pospeševalniki

- Kaj počnejo?
  - prikazujejo sliko
  - predvsem pa računajo

Grafični pospeševalniki

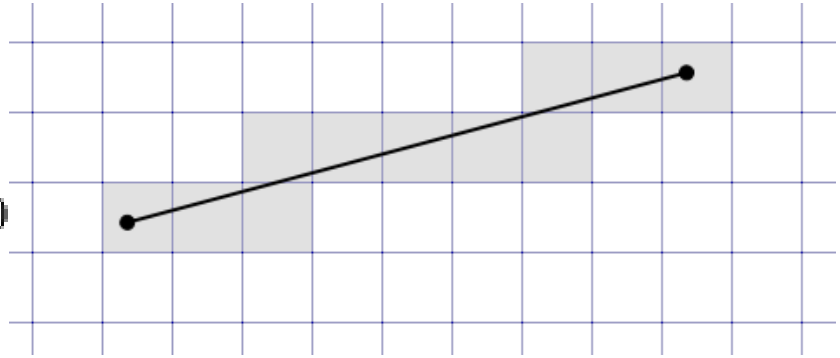
- Grafične operacije v dveh dimenzijah
  - bit-blit
    (BLock Image
    Transfer)





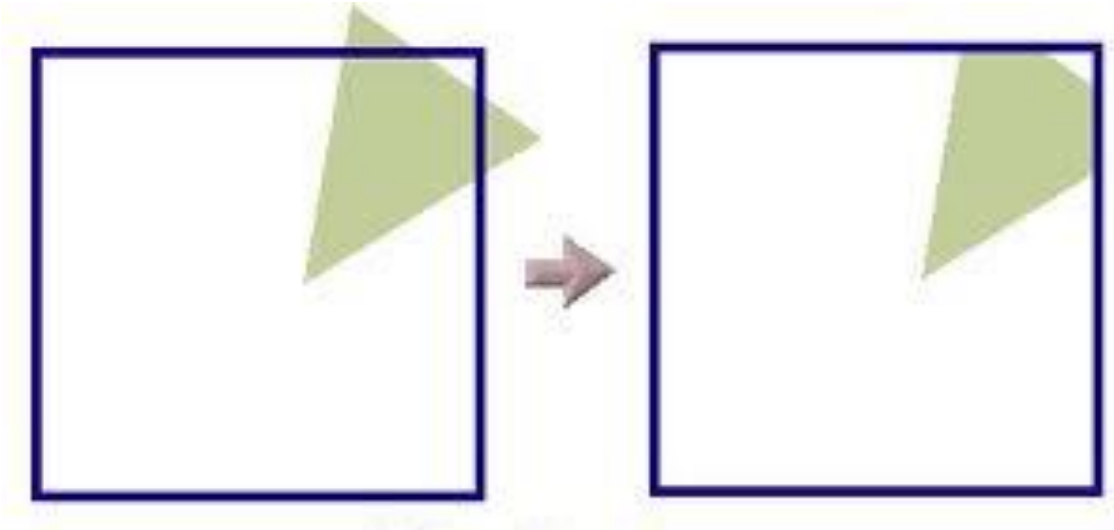
  - risanje črt

$$y = \frac{y_1 - y_0}{x_1 - x_0}(x - x_0) + y_0$$



Grafični pospeševalniki

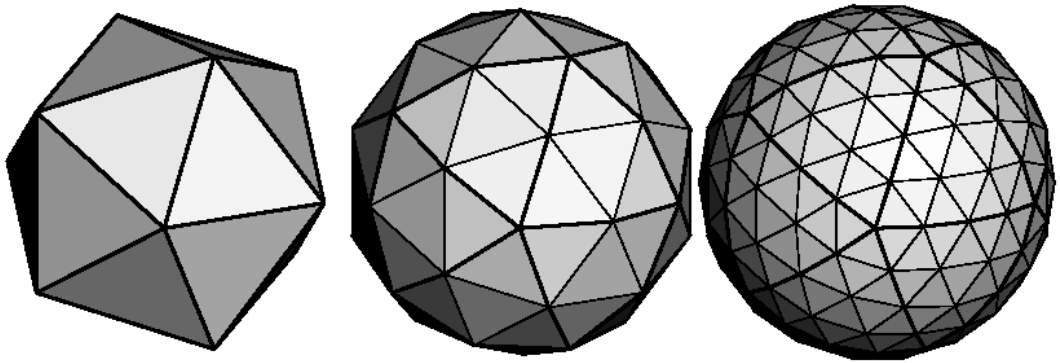
- Grafične operacije v dveh dimenzijah
  - barvanje in vzorčenje površin (zaprtih območij)
  - clipping



  - shranjevanje pogosto uporabljanih stvari v predpomnilnik

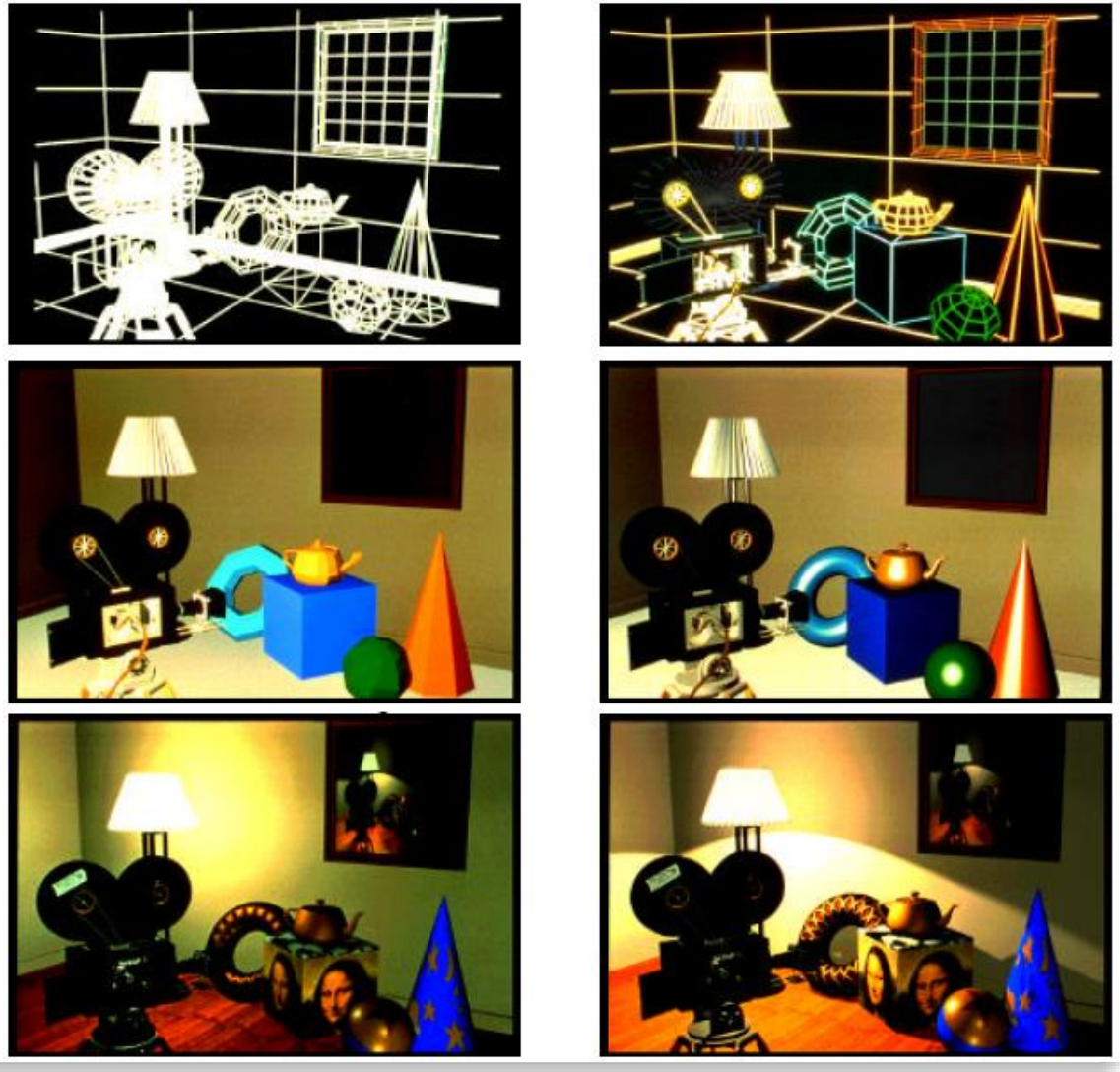
Grafični pospeševalniki

- Grafične operacije v treh dimenzijah
  - objekti 3D se običajno modelirajo s poligoni (trikotniki)
  - trikotnike se projicira na računalniški zaslon
  - večje kot je število trikotnikov, bolj realistična je slika



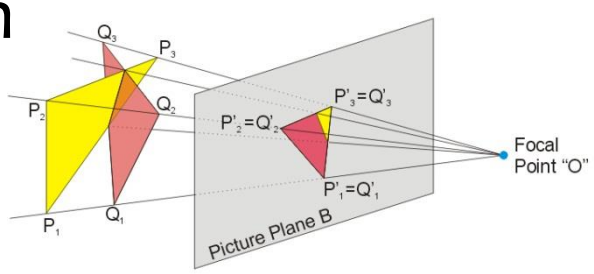
Grafični pospeševalniki

- Grafične operacije v treh dimenzijah
  - Postopek
    - žični okvir
    - nevidni robovi
    - barvanje
    - osvetlitev
    - teksture
    - senčenje



Grafični pospeševalniki

- Grafične operacije v treh dimenzijah
  - Za opisanimi postopki stojijo enačbe
    - Projekcije trikotnikov na želeno ravnino

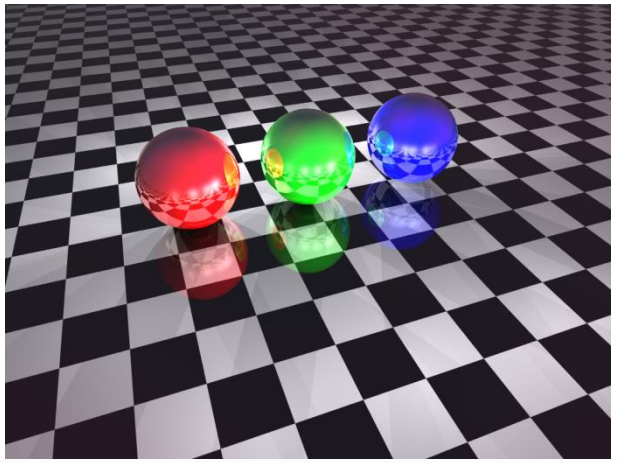


$$\begin{bmatrix}\mathbf{d}_x\\ \mathbf{d}_y\\ \mathbf{d}_z\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta_x & -\sin\theta_x\\ 0 & \sin\theta_x & \cos\theta_x\end{bmatrix}\begin{bmatrix}\cos\theta_y & 0 & \sin\theta_y\\ 0 & 1 & 0\\ -\sin\theta_y & 0 & \cos\theta_y\end{bmatrix}\begin{bmatrix}\cos\theta_z & -\sin\theta_z & 0\\ \sin\theta_z & \cos\theta_z & 0\\ 0 & 0 & 1\end{bmatrix}\left(\begin{bmatrix}\mathbf{a}_x\\ \mathbf{a}_y\\ \mathbf{a}_z\end{bmatrix} - \begin{bmatrix}\mathbf{c}_x\\ \mathbf{c}_y\\ \mathbf{c}_z\end{bmatrix}\right)$$

    - Senčenje
      - Računanje presečišč s telesi, …

$$t = \frac{A \cdot o.x + B \cdot o.y + C \cdot o.z + D}{A \cdot dir.x + B \cdot dir.y + C \cdot dir.z}$$

$$\vec{T} = \left(n_r \cdot (\vec{N} \cdot \vec{I}) - \sqrt{1 - n_r^2 \cdot (1 - (\vec{N} \cdot \vec{I})^2}\right) \cdot \vec{N} - n_r \cdot \vec{I}$$



Grafični pospeševalniki

- Velikost zaslona 2560x 1440
  - Nekdo mora za vsako od 3,68 mio. točk izračunati vse enačbe in to vsaj 25 krat na sekundo
- Nalogo odlično opravijo grafični pospeševalniki

Zasnova

- Senčilnik slikovnih točk (pixel shaders 2D) :
  - njegova naloga je, da za vsako točko določiti njeno barvo - vhodni podatki so numerični: barve svetlobnih teles, lastnosti materialov, teksture, …
- Senčilnik vozlišč (vertex shaders 3D):
  - Njegova naloga je, da za vsa vozlišča v navideznem 3D prostoru opravi vse geometrijske transformacije, potrebne, da se posamezno vozlišče prenese v zaslonske koordinate
    - transformacije modelov,
    - transformacija 3D pogledov,
    - perspektivna transformacija,
    - transformacija 2D pogledov,
    - rasterizacija

Zasnova

- Senčilniki slikovnih točk ter oglišč so bili implementirani kot ločene procesne enote
- Težave
  - Če imamo v neki aplikaciji veliko geometričnega računanja in malo računanja barv, potem so senčilniki vozlišč preobremenjeni, medtem ko senčilniki slikovnih točk ne delajo

  - Če imamo v neki aplikaciji veliko barvnega računanja in malo ali skoraj nič geometričnega, potem so senčilniki slikovnih točk preobremenjeni, medtem ko senčilniki vozlišč ne delajo

Odgovor – CUDA in OpenCL

- Arhitektura CUDA

  (Compute Unified Device Architecture)
  - 2006: NVIDIA GeForce 8800 GTX – prva naprava z arhitekturo CUDA
  - Vsaka ALE na čipu sedaj po potrebi računa, ali izvaja operacije senčenja slikovnih elementov ali oglišč
  - Enote ALE so razširili tako, da podpirajo IEEE standard za plavajočo vejico
  - Razširili so nabor ukazov: več splošno namenskih operacij
  - Procesorji lahko naključno dostopajo do pomnilnika (pišejo v in berejo iz poljubne pomnilniške besede)

Odgovor – CUDA in OpenCL

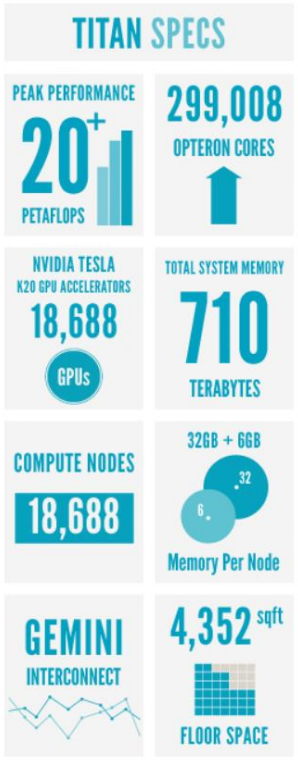
- V Nvidia omogočijo programerjem bolj neposreden dostop do grafične strojne opreme:
  - CUDA C – programski vmesnik z razširitvami programskega jezika C
  - Kmalu po izidu GeForce 8800 GTX, brezplačen prevajalnik za CUDA C
  - CUDA C postane prvi programski jezik, ki omogoča bolj splošno namensko računanje na grafični strojni opremi

Odgovor – CUDA in OpenCL

- OpenCL
  - Ustanovitev skupine KHRONOS
  - Pridružijo se mnogi proizvajalci
  - Posplošitev ideje za druge heterogene arhitekture
  - Hiter razvoj:
    - 2008 – OpenCL 1.0,
    - 2010 – OpenCL 1.1,
    - 2011 – OpenCL1.2
    - 2013 – OpenCL 2.0
    - 2015 – OpenCL 2.1
    - 2016 – OpenCL 2.2

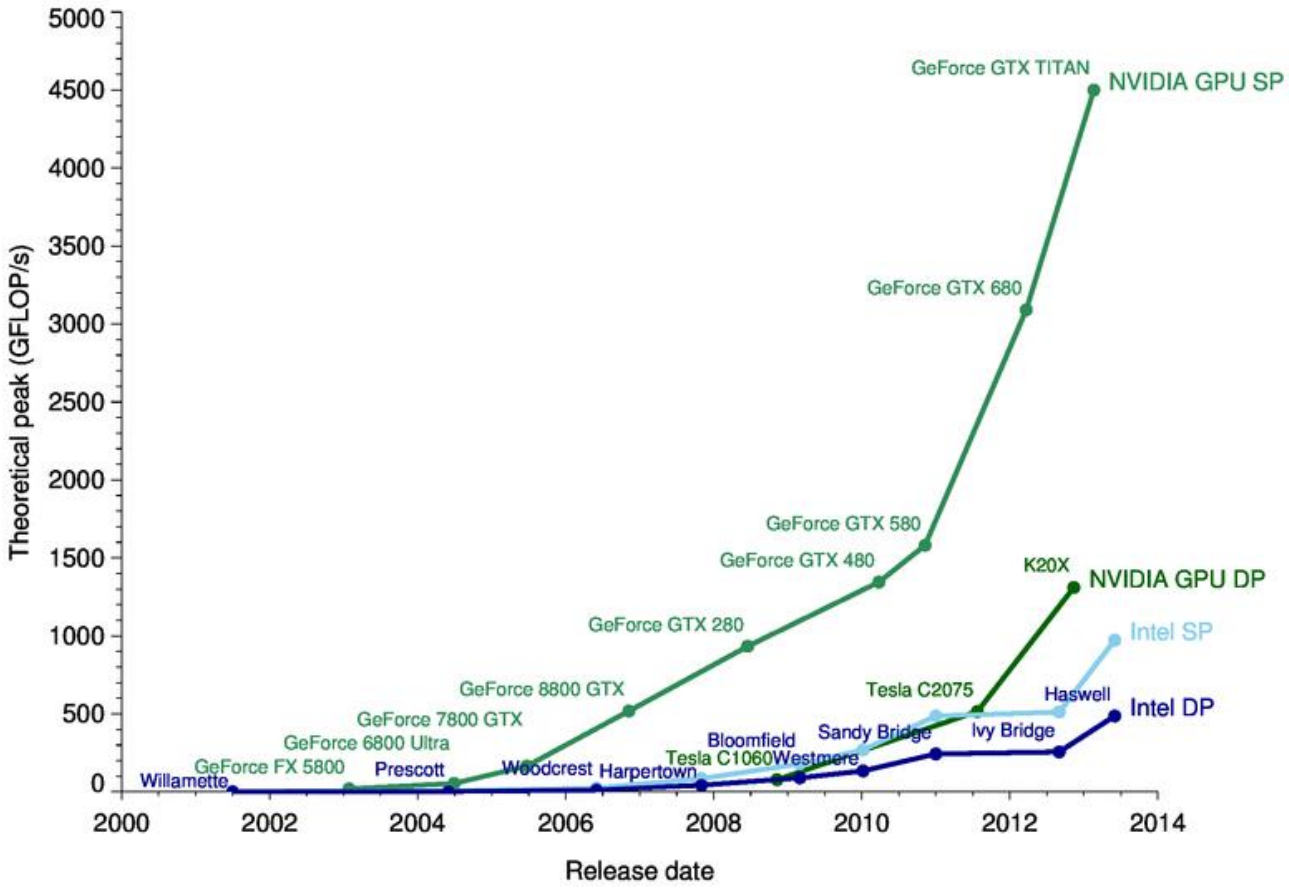
Superračunalniki

- Titan - Cray XK7
  - 18.688 vozlišč
  - vozlišče
    - procesor AMD Opteron 6274, 16 jeder
    - 32 GB pomnilnika
    - Nvidia TESLA K20X s 6GB pomnilnika
  - 40 PB diskovja



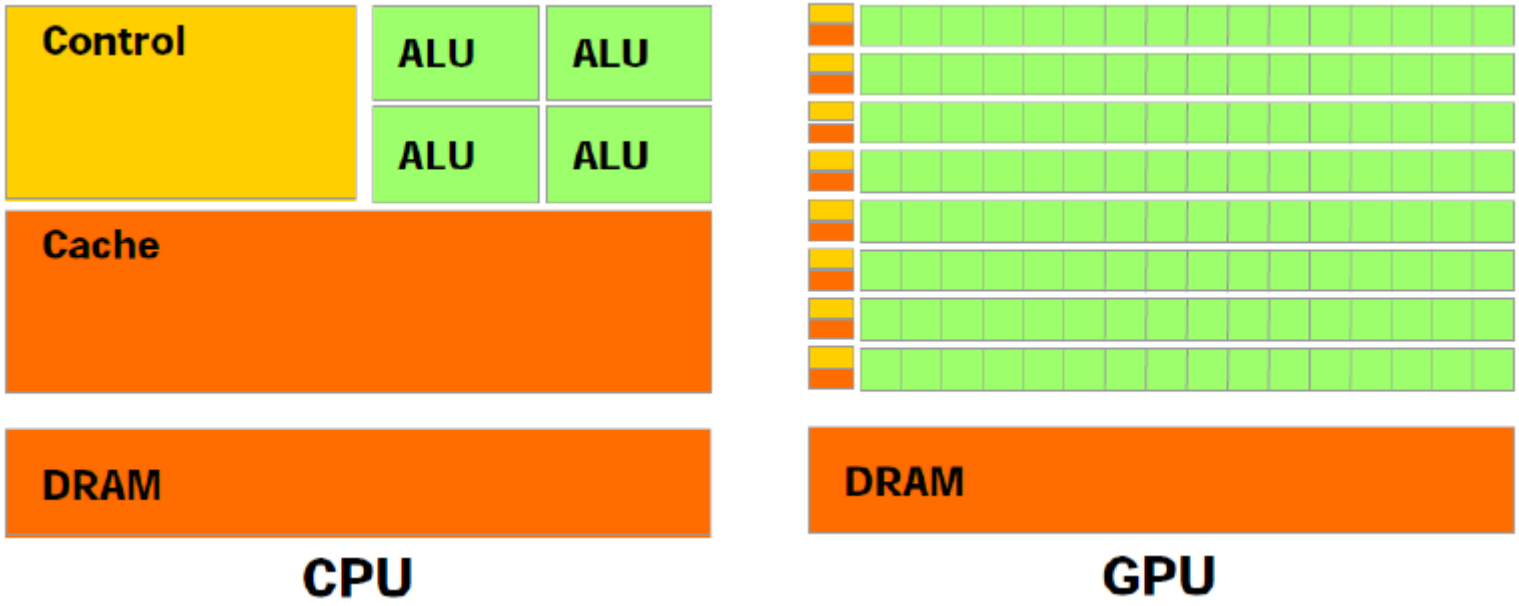
Računanje na grafičnih procesorjih

- Naraščanje zmogljivosti: grafični procesorji (zelena) proti centralnim procesorjem (modra) v enojni (SP) in dvojni (DP) natančnosti



Računanje na grafičnih procesorjih

- Grafični procesorji so računsko zelo zmogljive naprave
  - Delež tranzistorjev: CPU vs GPU

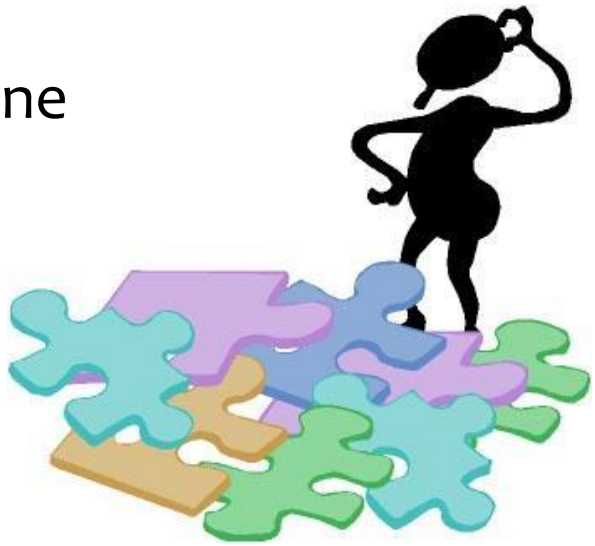


Problemi

- Grafične procesorje je težko uporabljati
- Zasnovani so bili za hitro prikazovanje grafike
  - neobičajen programski model
  - ukazi so vezani na računalniško grafiko
  - precej omejitev pri programiranju
- Strojna oprema je
  - močno paralelna
  - se močno razvija in spreminja
  - zasnova procesorjev precej tajna
- Programov iz CPE ne moremo enostavno prenesti

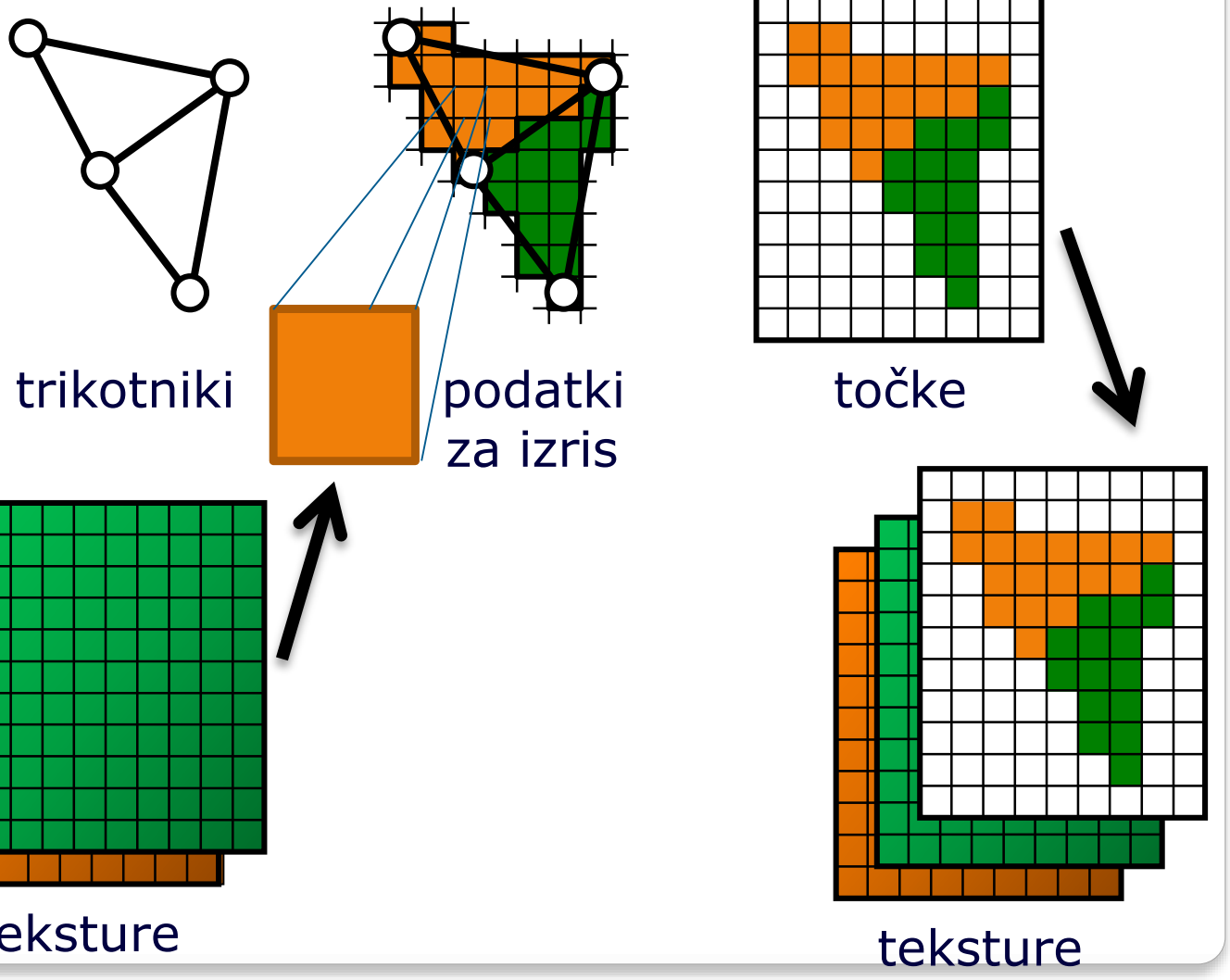
Problemi

- Grafični procesorji niso zdravilo za vse izzive
  - so hitri, ker so ozko specializirani
  - ne znajdejo se v zaporednem svetu
- Zgodovina
  - brez podpore za pomembne matematične operacije: cela števila, bitne operacije
  - paralelizem tipa SPMD (Single Process Multiple Data)
- Danes je mnogo pomanjkljivosti odpravljenih



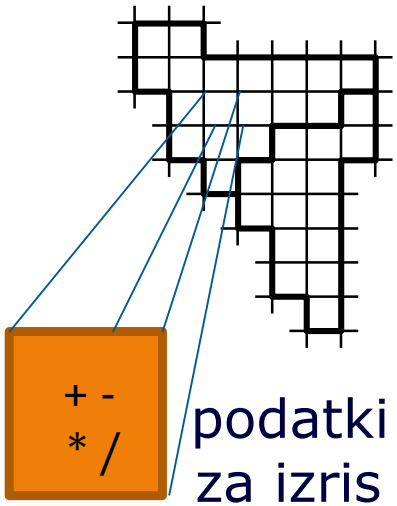
Programiranje

- Grafika
  - slike so sestavljene iz množice objektov
  - objekte moramo obdelati na enak način
  - natančno predpisani postopki

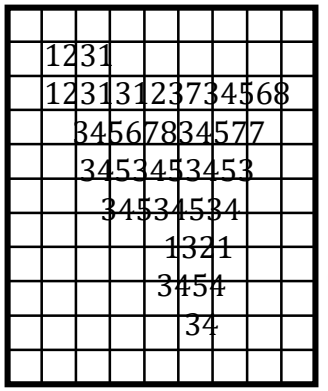


Programiranje

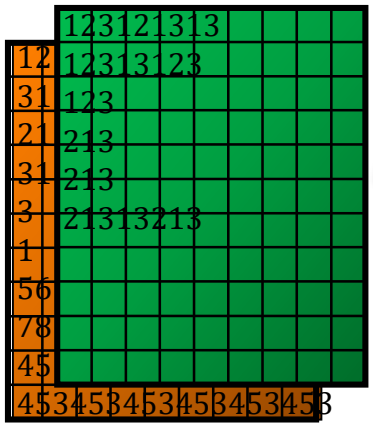
- Splošno
  - Abstrakcija
  - Namesto grafičnih elementov si predstavljamo številke



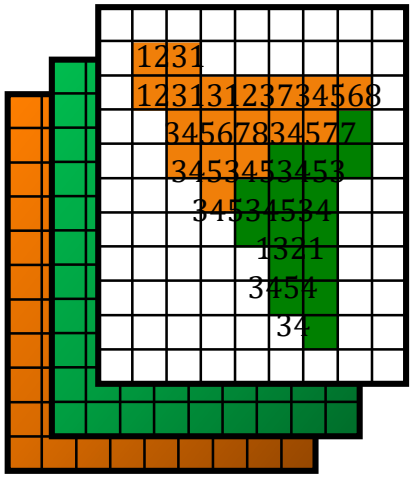
podatki za izris



točke



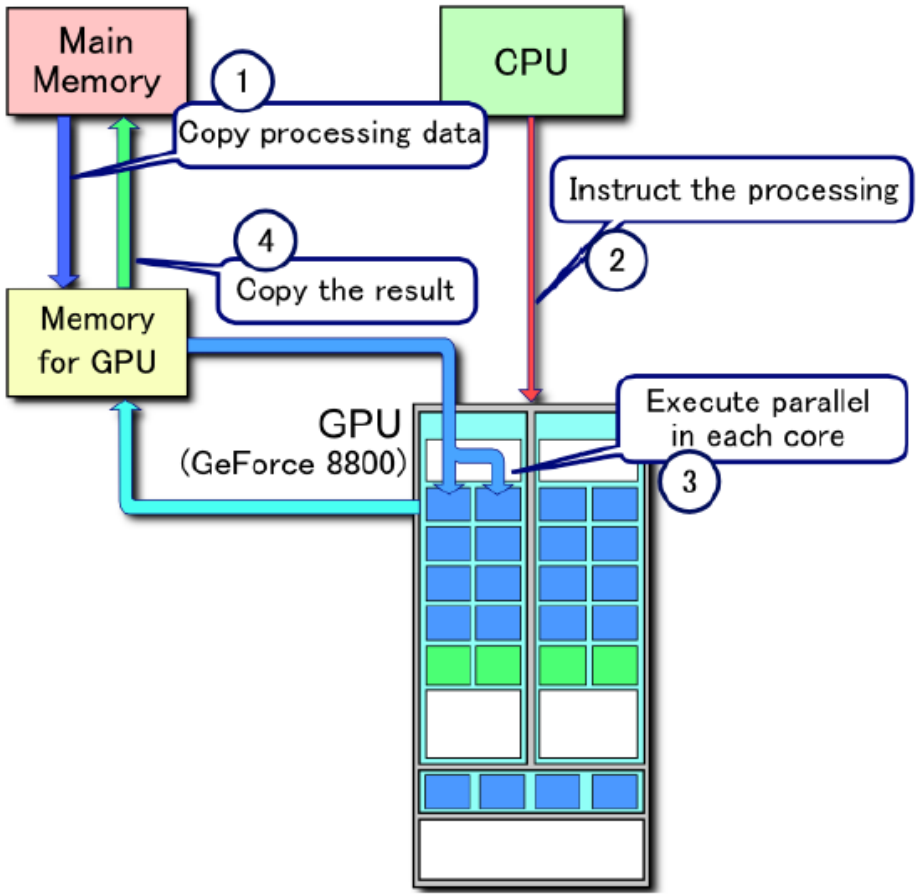
teksturni pomnilnik



teksturni pomnilnik

Programiranje

- Postopek
  1. prenos podatkov na grafični procesor
  2. zahteva za izračun
  3. računanje
  4. prenos podatkov v glavni pomnilnik

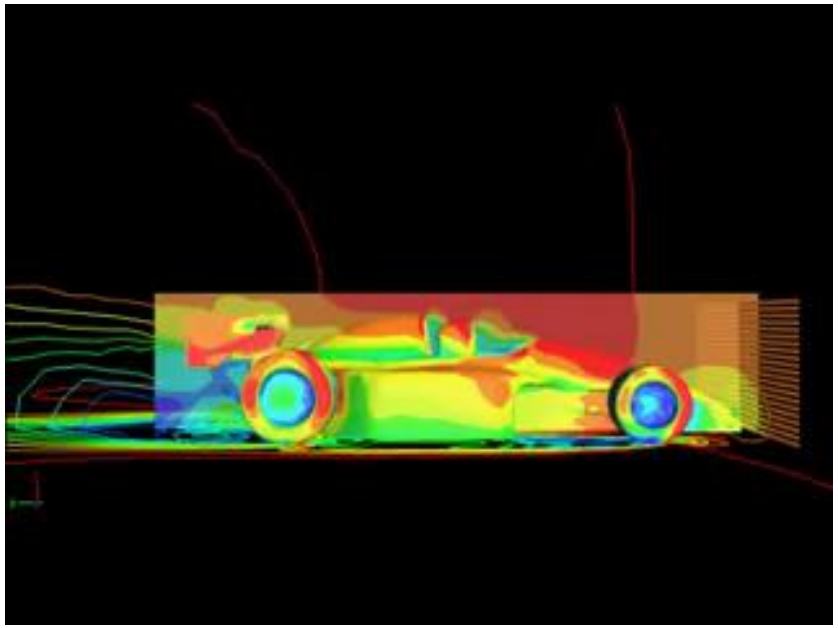


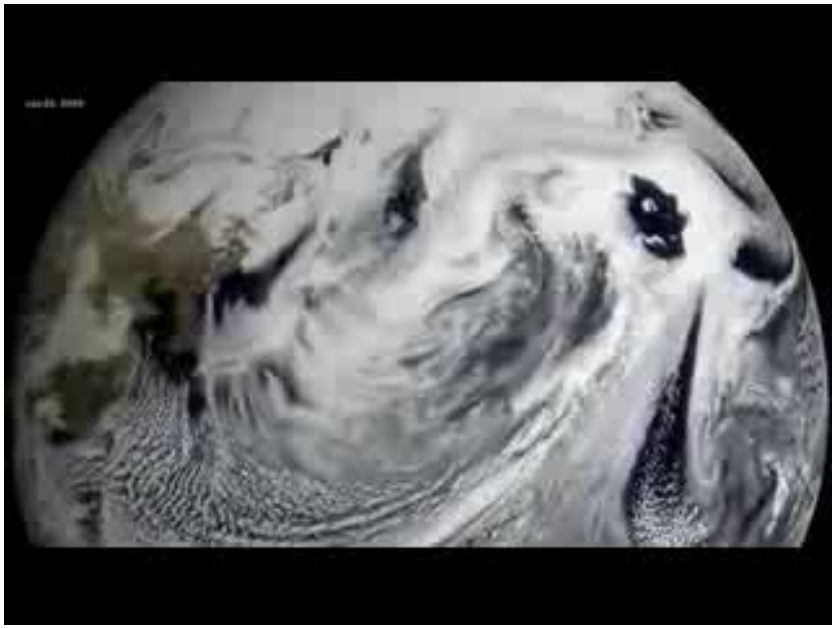
Kje to rabimo?

- Veliko problemov je računsko zelo zahtevnih
- Mnogi so pisani na kožo arhitekturi grafičnih procesnih enot (GPE)

Kje to rabimo?

- Raziskovanje dinamike tekočin in turbulenc
- Modeliranje vremena in obnašanja okolja





- kemija, fizika, statistična mehanika, razvoj materialov, raziskovanje vesolja, astrofizika, medicina, …
- Biologija, genetika, genski inženiring, določanje zgradbe proteinov, delovanje encimov, modeliranje celice, razvoj zdravil

Kje to rabimo?

- Problemi z visoko stopnjo podatkovnega paralelizma:
  - procesiranje slik,
  - simulacija fizikalnih modelov,
  - problemi N teles,
  - iskanje in urejanje,
  - dinamika fluidov,
  - vektorska polja, ...

Kdaj pa na GPE lahko pozabimo?

- Problemi, ki niso pisani na kožo arhitekturi grafičnih procesnih enot (GPE):
  - operacije nad drevesi,
  - operacije nad podatki v povezanih seznamih, ...

… in nadaljujmo …

mnogo bolj zares …

Izvajalni model

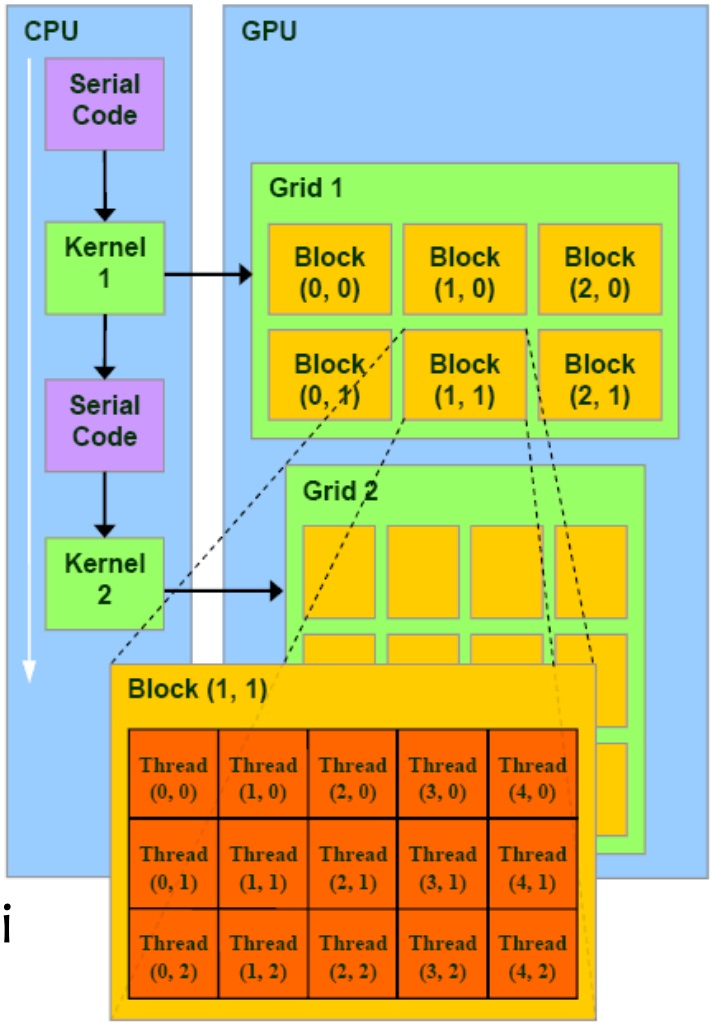
- Filozofija:
  - ustvarimo virtualno neomejeno število vzporednih niti, ki se bodo dinamično razvrščale in izvajale na stroji opremi
- Programski model: na GPE gledamo
  - kot na soprocesor, ki podpira neomejeno število niti,
  - s programsko vidnim hierarhičnim pomnilnikom
- GPE je praviloma uporabna le pri problemih, kjer imamo visoko stopnjo podatkovnega paralelizma
  - operacije se izvajajo na veliki količini podatkov,
  - ti so običajno porazdeljeni v podatkovni strukturi 2D/3D polje (mreža)

Izvajalni model

- Program sestavljajo:
  - ščepci:
    - ščepci so samostojni kosi programske kode,
      ki se izvajajo na napravi GPE,
    - v nekem trenutku izvaja naprava GPE en sam ščepec
      (na novejših sistemih je ščepcev lahko več)
  - serijska koda:
    - izvaja se na gostitelju
      - gostitelj je običajna CPE
    - serijska koda je potrebna za
      - prenos kode posameznega ščepca na napravo GPE
      - za prenos podatkov od gostitelja na napravo GPE in nazaj

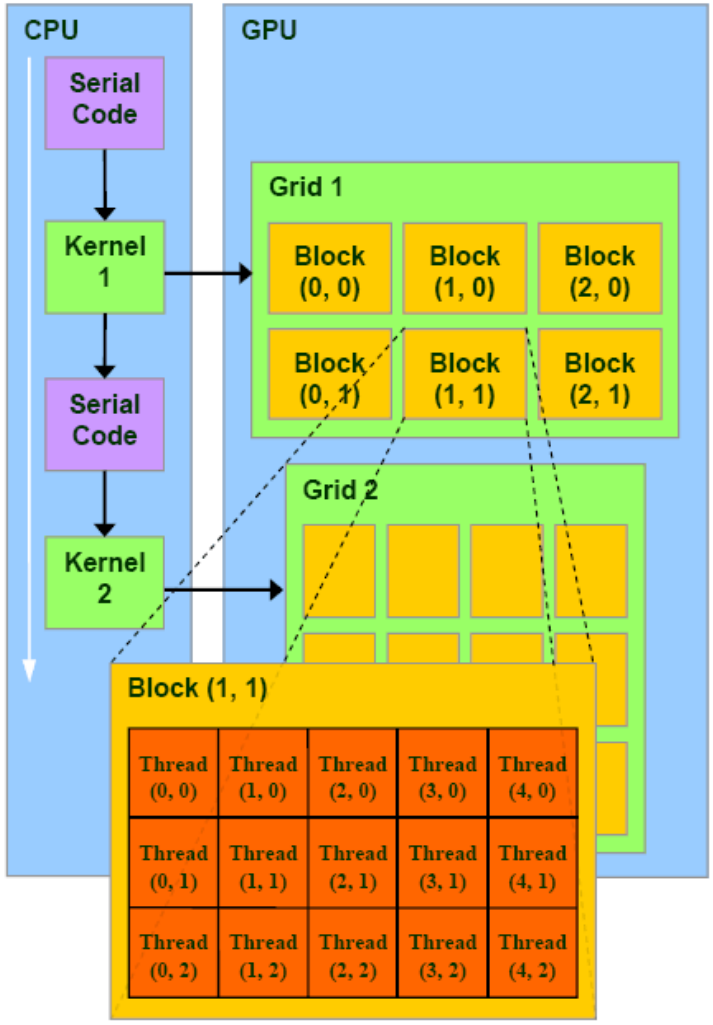
Izvajalni model

- Ščepec (ang. kernel)
  - sestavlja veliko število niti
  - niti so logično porazdeljene v polje
- Organizacija niti v ščepcu
  - blok niti (ang. block)
    - je skupek niti
    - posamezne niti v bloku izvajajo enako programsko kodo
    - vse se začnejo izvajati z istim ukazom
    - med seboj lahko komunicirajo preko skupnega pomnilnika
  - mreža niti (ang. grid)
    - je sestavljena iz neodvisnih blokov niti

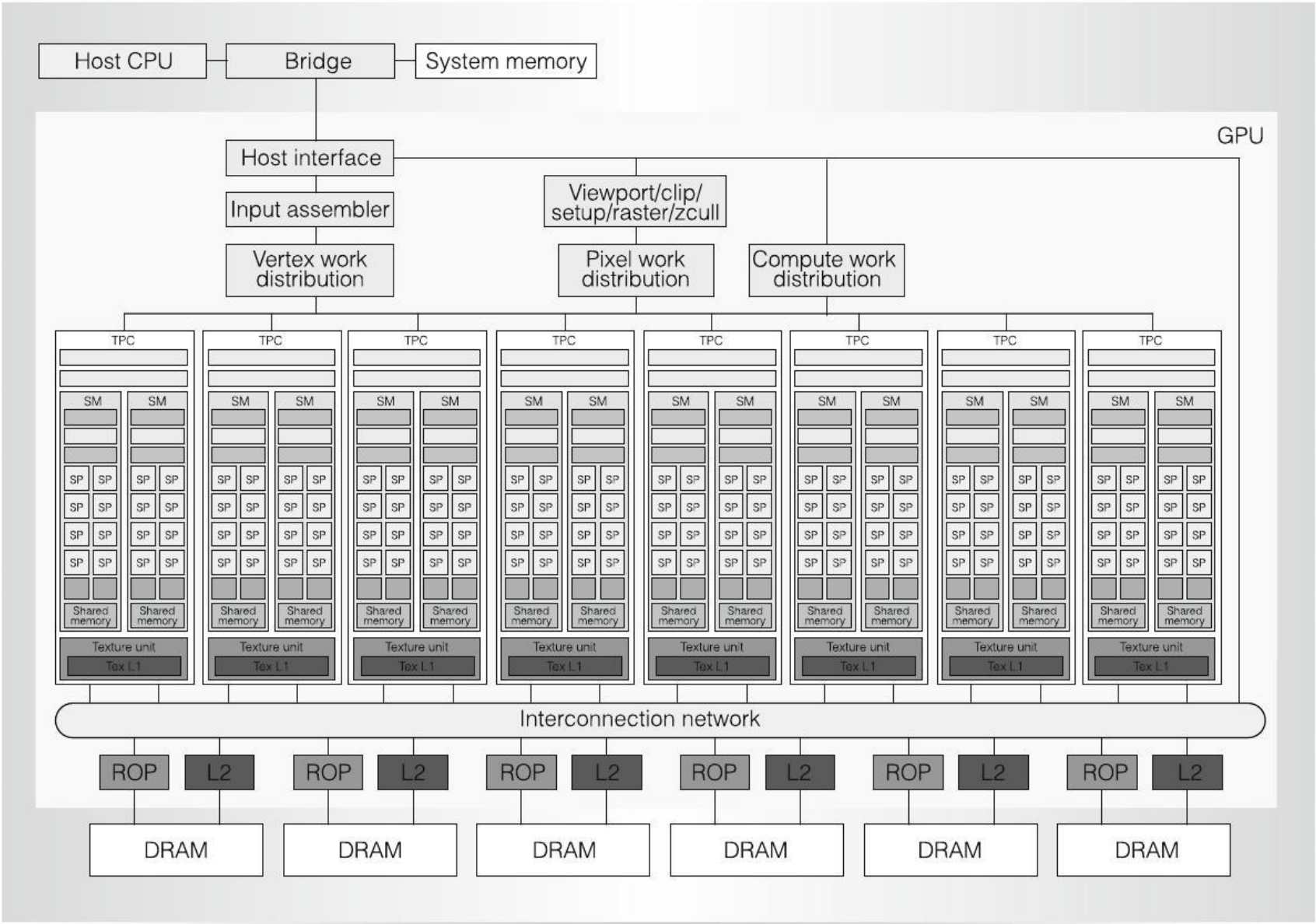


CUDA izvajalni model

- Izvajanje
  - Bloki se izvajajo
    - lahko vzporedno
    - v poljubnem vrstnem redu
  - Ščepec
    - če je pravilno organiziran v bloke
      - se bo pravilno izvajal na napravi GPE, ki lahko istočasno izvaja samo en blok, ali vzporedno več blokov
      - na ta način je zagotovljena visoka raztegljivost (ang. skalabilnost) programske opreme

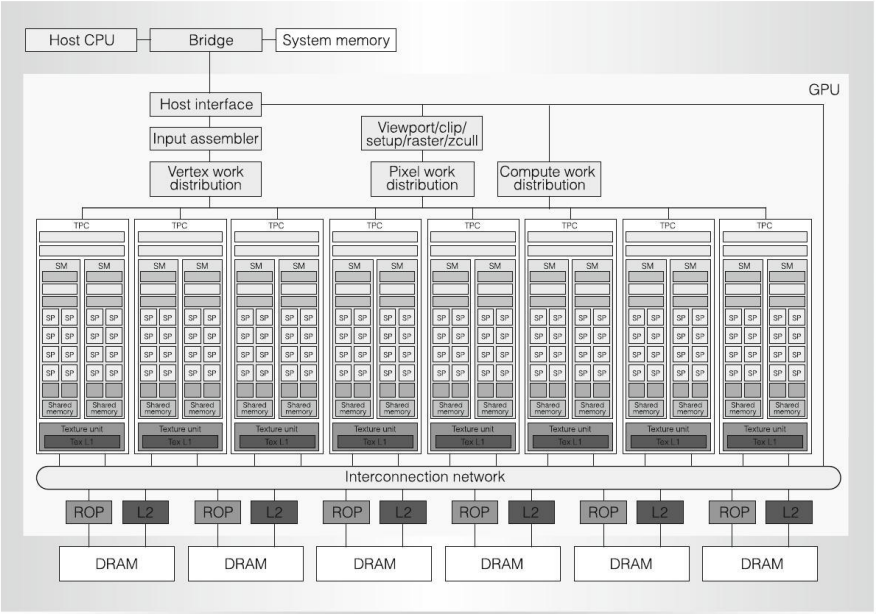


Nvidia GeForce 8800



Nvidia GeForce 8800

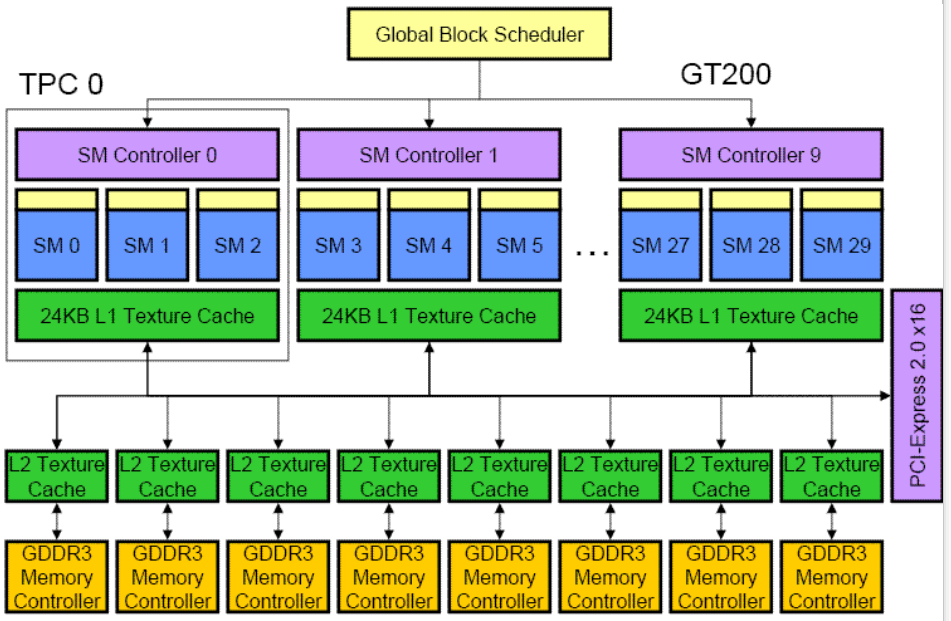
- Procesor SM (Streaming Multiprocesor)
  - Enemu SM se v izvajanje dodeli en blok niti
- Izvajalne enote SP (Streaming Processor)
  - Vsaka nit se izvaja v eni izvajalni enoti SP



- Posameznemu procesorju SM pripada skupni ali deljeni pomnilnik (Shared Memory)
  - hiter dostop
  - preko njega si niti v bloku lahko izmenjujejo podatke
- Globalni pomnilnik DRAM
  - Preko njega si lahko podatke izmenjujejo niti iz različnih blokov

Zgradba GT200

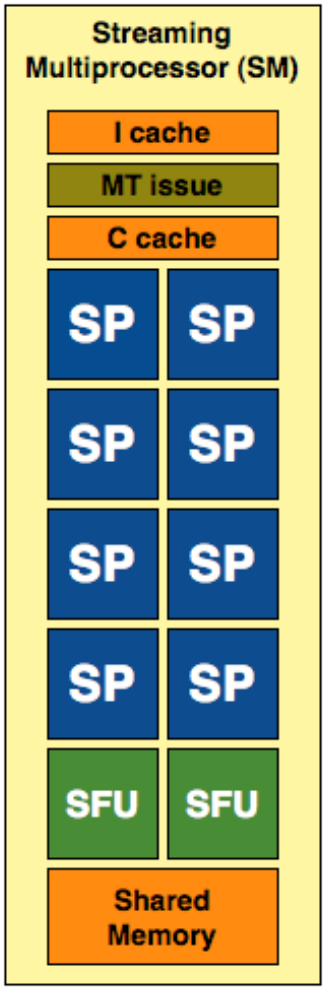
- GT200 je večjedrni procesor z dvema nivojema hierarhije
  - 1. nivo: 10 večjedrnih procesorjev TPC (Texture Processor Cluster)
  - 2. nivo: vsak TPC ima
    - 3 jedra s procesorji SM (Streaming Multiprocessors)
    - cevovod za dostop do globalnega pomnilnika (texture pipeline)



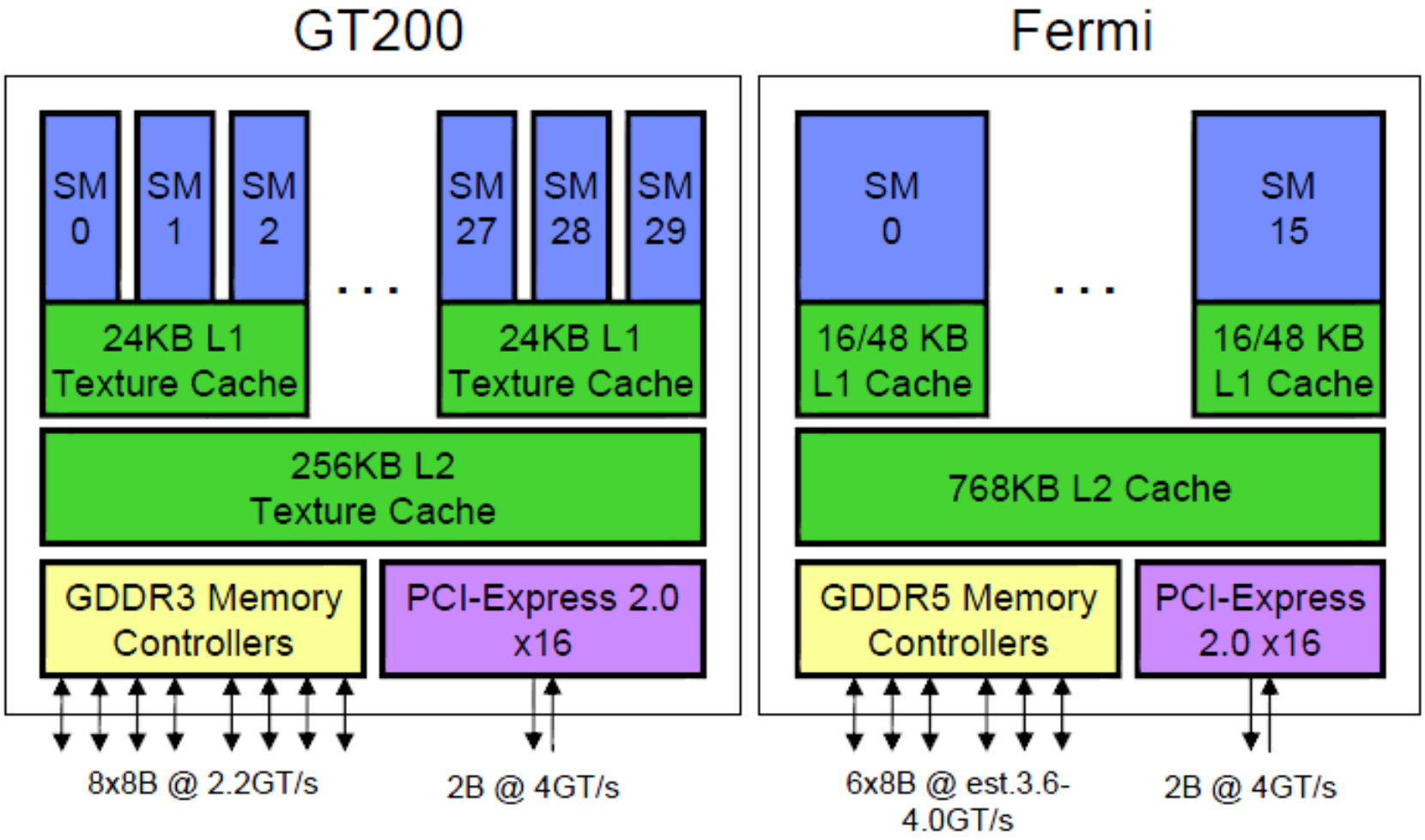
- Vsak procesor SM v TPC je v bistvu samostojen procesor
  - ima celotno logiko za zajem dekodiranje in izstavljanje ukazov, izvajalne (funkcijske) enote,
  - vendar si procesorji SM v istem TPC delijo logiko za dostop do globalnega pomnilnika

Zgradba GT200, SM

- Vsak SM ima 8 blokov izvajalnih enot SP
- Izvajalna enota SP ima
  - lasten programski števec
  - lastne registre
    (dodeli se mu del registrskega niza)
  - manjka mu pa vsa logika za zajem,
    dekodiranje in izstavljanje ukazov
- Izvajalne enote SP
  - niso pravi procesorji
  - so zelo podobne izvajalnemu delu cevovoda
    v modernih procesorjih



Zgradba Nvidia Tesla in Fermi



Zgradba Nvidia Fermi

- Fermi, SM
  - 32 SP
  - Dva razvrščevalnika snopov niti (pri Tesla en sam)
  - Oznake:
    - Core – SP
    - LD/ST – računanje naslovov vira in ponora
    - SFU – Special Function Unit

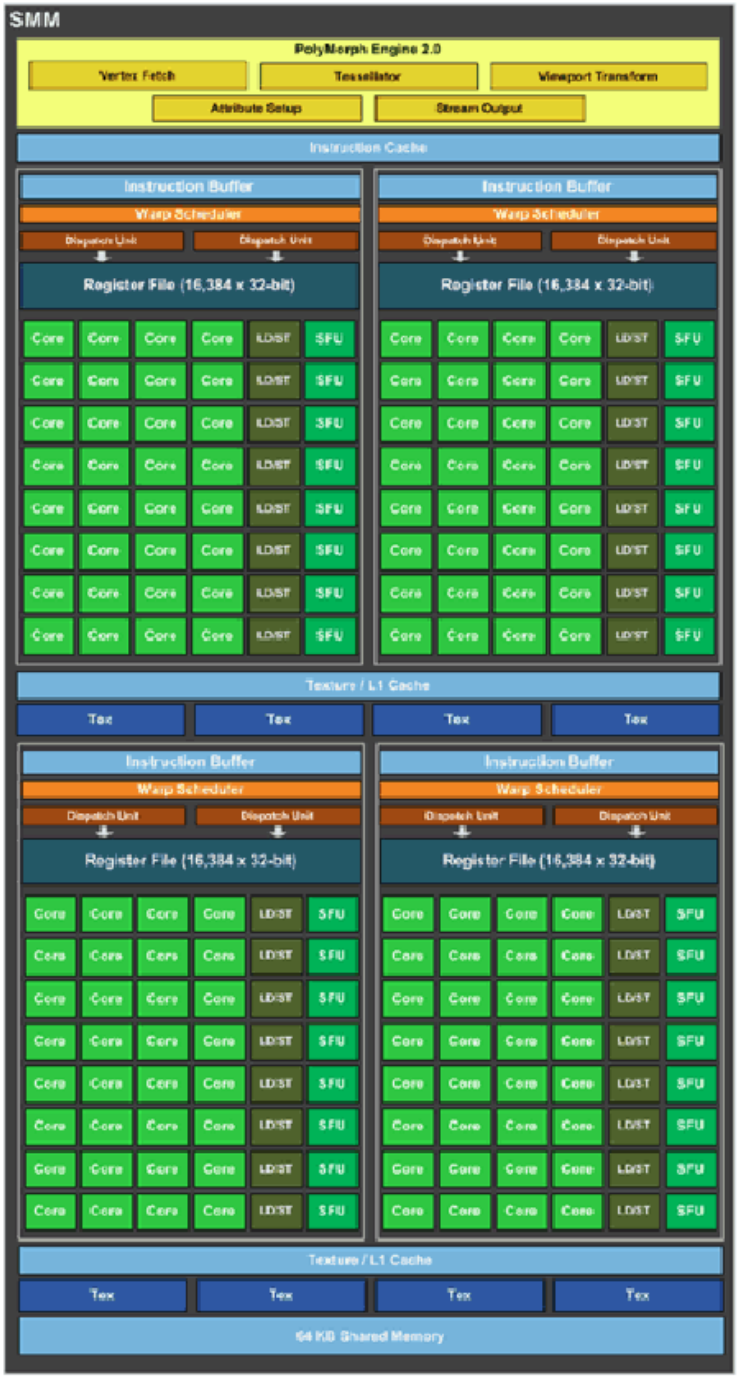
Zgradba Nvidia Kepler

Zgradba Nvidia Kepler

- Kepler, SM
  - 192 SP enot
  - 32 SFU enot
  - 32 LD/ST enot
  - 64 DP enot
  - 4 razvrščevalniki snopov niti
  - Oznaka
    - DP – Double Precision Unit

Nove mikro-arhitekture

- Maxwell (128 jeder)
  - Bolj hierarhična zgradba multiprocesorja



Nove mikro-arhitekture

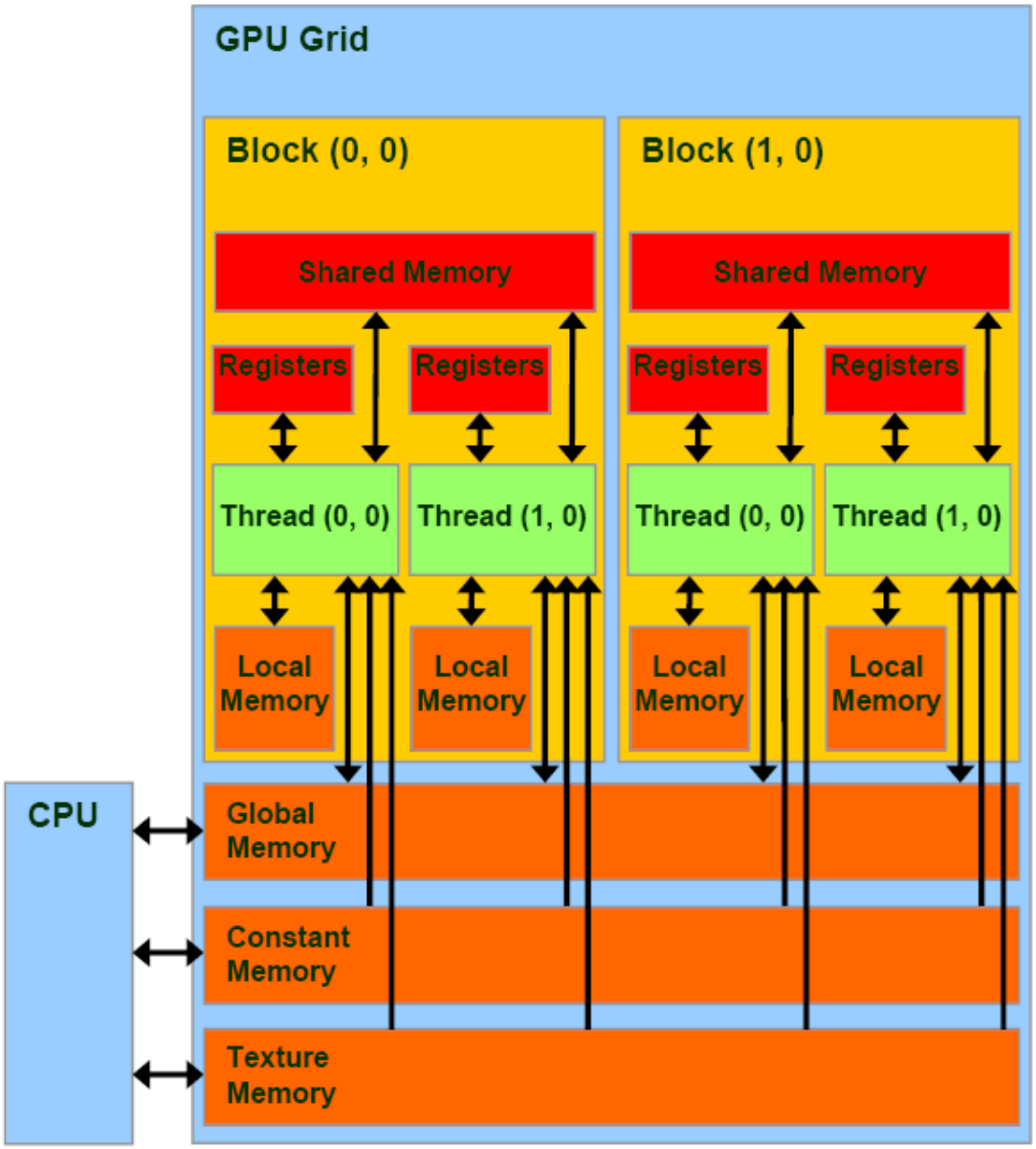
- Pascal (64 jeder za enojno in 32 za dvojno natančnost)
  - Dostop gostitelja in naprave do pomnilnika na drugi napravi
  - Večje število registrov,
  - več deljenega pomnilnika
  - Podpora za polovično natančnost (globoko učenje)

- Volta

Pomnilniška hierarhija

- Na napravi GPE je celoten programsko viden pomnilnik sestavljen iz naslednjih ločenih pomnilniških prostorov:
  - registri (register file), 1 urina perioda
  - deljeni pomnilnik (shared memory), 5 urinih period
  - lokalni pomnilnik (local memory), 500 urinih period
  - globalni pomnilnik (global memory), 500 urinih period
  - pomnilnik konstant (constant memory), 5 urnih period
  - pomnilnik tekstur (texture memory), 5 urinih period

Pomnilniška hierarhija



Registri

- Registrski niz je privaten za vsako nit
  - Tesla (=GT200): 16K x 32 bitov v enem procesorju SM
  - Fermi (=GF100): 32K x 32 bitov v enem SM
  - Kepler (=GK110): 65K x 32 bitov v enem SM
  - Vsaki izvajalni enoti SP pripada 2K registrov (jih ni dovolj za vse!!!)
  - Izvajalna enota hkrati izvaja 32 niti (snop)
  - Nit lahko uporablja do 127 (Tesla), 63 (Fermi), 255 (Kepler) registrov
- Organizacija
  - registri so 32-bitni
  - implementirani so na isti rezini kot procesorji SM
- Dodeljevanje registrov
  - statično: za vsak blok
  - dinamično: za vsako nit v bloku od 4 – 128 registrov

Skupni pomnilnik (Shared Memory)

- Vsakemu procesorju SM pripada:
  - 16 kB skupnega pomnilnika (Tesla)
  - 16/48 kB skupnega pomnilnika (Fermi)
  - 16/32/48 kB skupnega pomnilnika (Kepler)
  - implementiran na isti rezini, kot SM (dostop 1 urina perioda)
- Organizacija pri GT200:
  - 16 modulov (bank) x 256 x 32 bit (16x256x4=16 kB)
  - skupaj 4096 x 32 bitov
- Dodeljevanje
  - dinamično posameznim blokom, ki se izvajajo na nekem procesorju SM
  - do pomnilnika, dodeljenega posameznemu bloku, lahko dostopa do 512 niti

Skupni pomnilnik (Shared Memory)

- Značilnosti
  - omogoča hitro komunikacijo med nitmi v istem bloku
  - podatki iz globalnega pomnilnika se s pomočjo ukaza LOAD prenesejo le v registre
  - če želimo podatke prenesti iz globalnega pomnilnika v skupni pomnilnik, jih moramo najprej prenesti v registre in nato iz registrov shraniti v skupni pomnilnik
  - uporabljamo ga
    - za shranjevanje skupnih podatkov
      - primer: skupni števec za vse niti v bloku,
    - za shranjevanje podatkov, do katerih niti pogosto dostopajo, ...

Lokalni pomnilnik

- Lastnosti
  - implementiran je v pomnilniku DRAM
  - privaten za vsako nit (podobno kot registri)
  - relativno dolga latenca pri dostopu
  - uporablja se ga, kadar zmanjka registrov

Globalni pomnilnik

- Značilnosti
  - viden je vsem nitim v mreži (vsem nitim v ščepcu)
  - za branje in pisanje je dostopen
    - iz gostiteljske CPE in
    - iz naprave GPE
  - (na GT200 ni predpomnjen)
  - implementiran v GDDRx (double data rate)
- GT200:
  - 8 x 64-bitnih GDDR3 krmilnikov – 512 bitno vodilo med procesorji SM in globalnim pomnilnikom
  - krmilniki delajo pri 1107 MHz:
    - teoretično:
      - 1107 MHz x 2 = 2,214 G prenosov/s
      - 2,214 G/s x 64 B = 141,696 GB/s

Pomnilnika tekstur (in konstant)

- Na vsaki napravi GPE sta še dva bralna pomnilniška prostora:
  - pomnilnik konstant
  - pomnilnik tekstur
- Ostanka grafične narave naprav GPE
- Oba sta bralno predpomnjena na napravi
  - v primeru, da CPE piše, se predpomnilnik razveljavi
- Pomnilnik konstant
  - realiziran v DRAM-u
  - velikost: 64 KB
  - uporablja se ga za ukaze
- Pomnilnik tekstur
  - realiziran v DRAM-u
  - ima dvodimenzionalno lokalnost
  - uporablja se ga za shranjevanje konstant in tekstur v grafičnih aplikacijah

Zgradba Nvidia Kepler

- K20 – lastnosti
  - Zmogljivost (dvojna natančnost): 1,17 TFLOPS
  - Zmogljivost (enojna natančnost): 3,52 TFLOPS
  - Prenos podatkov: 208 GB/s
  - Pomnilnik: 5 GB
  - Število jeder: 13 (SMX) x 192 = 2496
  - Poraba: 225 W

Izvajanje ukazov

- Bloki in nitke so arhitekturno vidni programerju
  - na mikro arhitekturnem nivoju sistem razvršča in izvaja snope nitk (warps)
  - snopi niso programsko vidni
    (programerji jih ne vidijo in nič ne vedo o njih)
- Snop
  - je sestavljen iz 32 zaporednih nitk, ki izvajajo isto kodo
  - vsak SM lahko skrbi za izvajanje 32 snopov
    (vsega skupaj je to 32 x 32 = 1024 nitk)

Izvajanje ukazov

- Lastnosti SM
  - Enota SM za razliko od modernih super-skalarnih procesorjev nima špekulativnega izvajanja in napovedovanja vejitev
  - SM predvideva, da bodo vse nitke v snopu izvajale popolnoma isto kodo, ki ne vsebuje vejitev
  - SM so zaradi tega enostavnejši in porabijo manj energije

- SIMT – Single Instruction Multiple Threads

Izvajanje ukazov

- Snopi in hitrost izvajanja niti
  - najbolj učinkovito izvajanje imamo takrat, ko niti ne vejijo – takrat vse niti v snopu potrebujejo iste izvajalne (funkcijske) enote
  - če imamo v snopu N različnih vejitev, računske zmogljivosti padejo približno za faktor N
    - pri N različnih vejitvah v snopu, ima v nekem trenutku N niti različne vrednosti programskega števca
    - takrat se izvaja samo ena nit, ostalih N-1 niti pa čaka
- Vsako urino periodo se izstavi en ukaz
- Funkcijske enote delajo z dvakrat višjo frekvenco (hitra ura) kot poteka zajemanje in izstavljanje ukazov!

Izvajanje ukazov

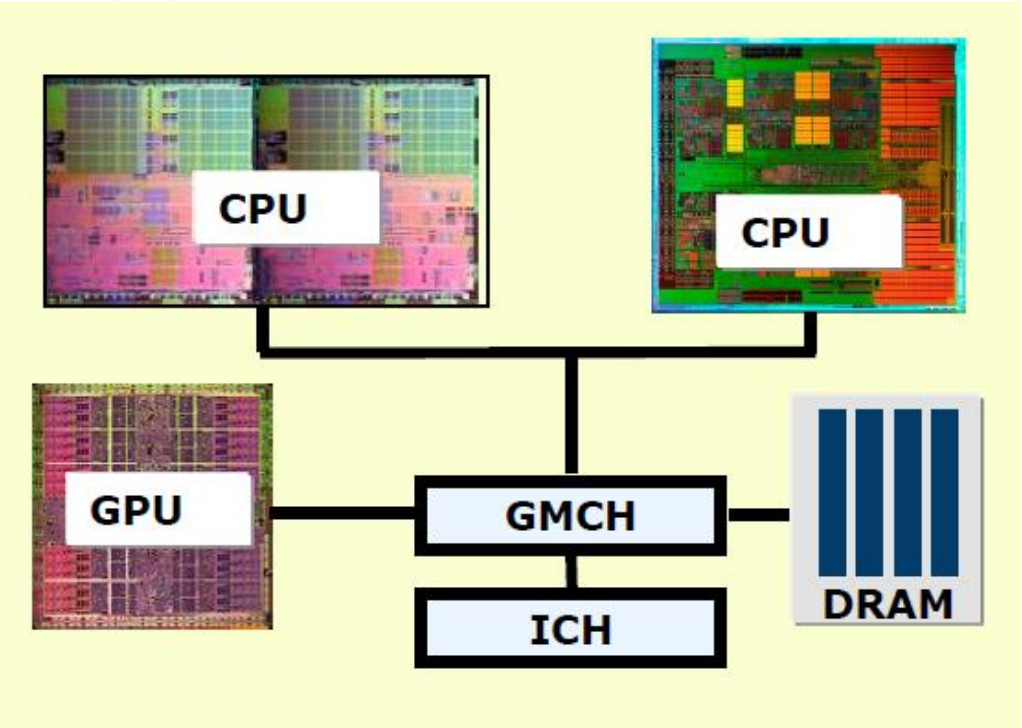
- Vsak SM ima 8 SP (GT200), snop ima 32 niti
  - Ena nit se izvaja na enem SP → nit se izvaja v štirih korakih (cevovod)
  - Niti v istem snopu ne potrebujejo sinhronizacije, saj se izvajajo hkrati
- Zakaj potrebujemo toliko snopov v vsakem SM?
  - Zakrivanje latence (dostop do glavnega pomnilnika 500 urinih period)
  - Med čakanjem se lahko izvajajo drugi, pripravljeni snopi
- Z razvrščanjem niti ni stroškov
  - Na voljo je mnogo snopov, vedno se najde kakšen pripravljen, SM je tako ves čas izvajanja zaseden
  - Množica snopov je tudi razlog, da predpomnilniki niso potrebni

SM – izvajanje ukazov

- Kako lahko 8 izvajalnih enot SP hkrati izvaja 32 nitk?
  - latenca ALE/MAD enot je 4 – to pomeni, da se en ukaz izvaja 4 urine periode hitre ure
  - vsako urino periodo vstopi v cevovode 8 nitk iz istega snopa
  - po 4 urinih periodah se cevovodi napolnijo z 32 nitkami iz enega snopa, nato po začetni latenci vsako urino periodo dobimo rezultat za 8 nitk
  - tudi enote za izvajanje skočnih ukazov imajo latenco 4
  - latenca enot SFU je 16-32 ciklov, ukaz za množenje izvedejo v 4 urinih periodah

Heterogeni sistemi

- Moderni sistemi vključujejo
  - Eno ali več CPU
  - Eno ali več GPU
  - Procesorje DSP
  - …

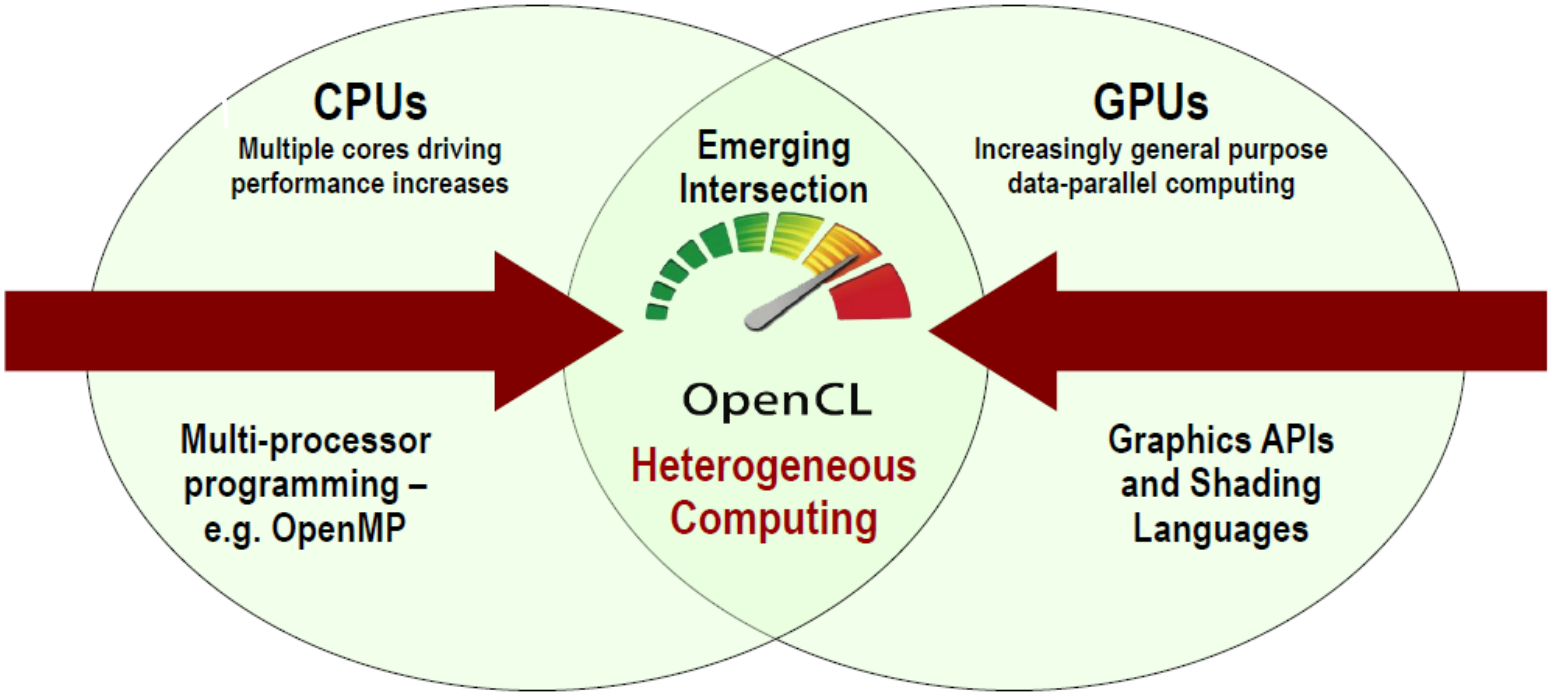


GMCH – Graphic/Memory Control Hub
ICH – Input/output Control Hub

- OpenCL
  - Omogoča programerjem, da napišejo en sam program, ki je prenosljiv med vsemi viri na heterogenem sistemu

Heterogeni sistemi

- Potreba po industrijskem standardu



- OpenCL – Open Computing Language
  - Odprt standard za heterogene sisteme

Razvoj OpenCL

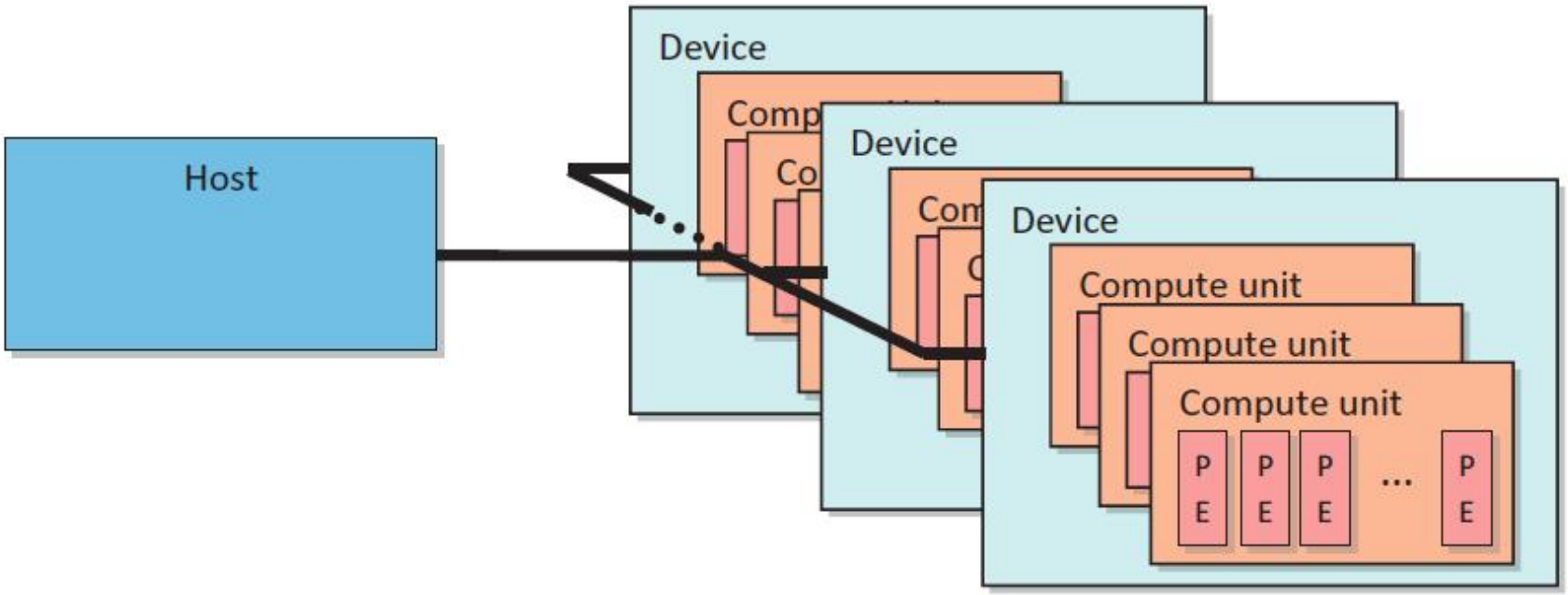
- Pobudniki
  - AMD, ATI
    - Združitev, potreba po enotni platformi
  - NVIDIA
    - Želja po tržnem deležu CPU
  - INTEL
    - Želja po tržnem deležu GPU
  - APPLE
    - Želja po standardizaciji (enakem vmesniku za opremo različnih dobaviteljev)
- Pripravijo osnutek standarda

Razvoj OpenCL

- Ustanovite skupine KHRONOS
  - Pridružijo se mnogi drugi proizvajalci
  - Hiter razvoj:
    - 2008 – OpenCL 1.0, 2010 – OpenCL 1.1, 2011 – OpenCL1.2
    - 2013 – OpenCL 2.0, 2015 – OpenCL 2.1, 2016 – OpenCL 2.2
  - Od mobilnih naprav do superračunalnikov

Model: platforma

- En gostitelj in več računskih naprav
- Vsaka računska naprava je razdeljena na več procesnih elementov

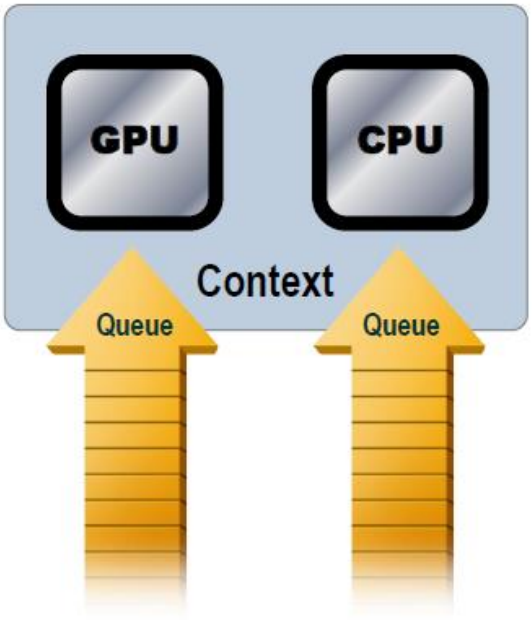


Model: kontekst

- Kontekst je okolje za
  - izvajanje ščepca
  - upravljanje s pomnilnikom
  - med vključenimi napravami poteka sinhronizacija
- Kontekst vključuje
  - Množico naprav
  - Pomnilnik
  - Ukazne vrste, preko katerih poteka izvajanje operacij

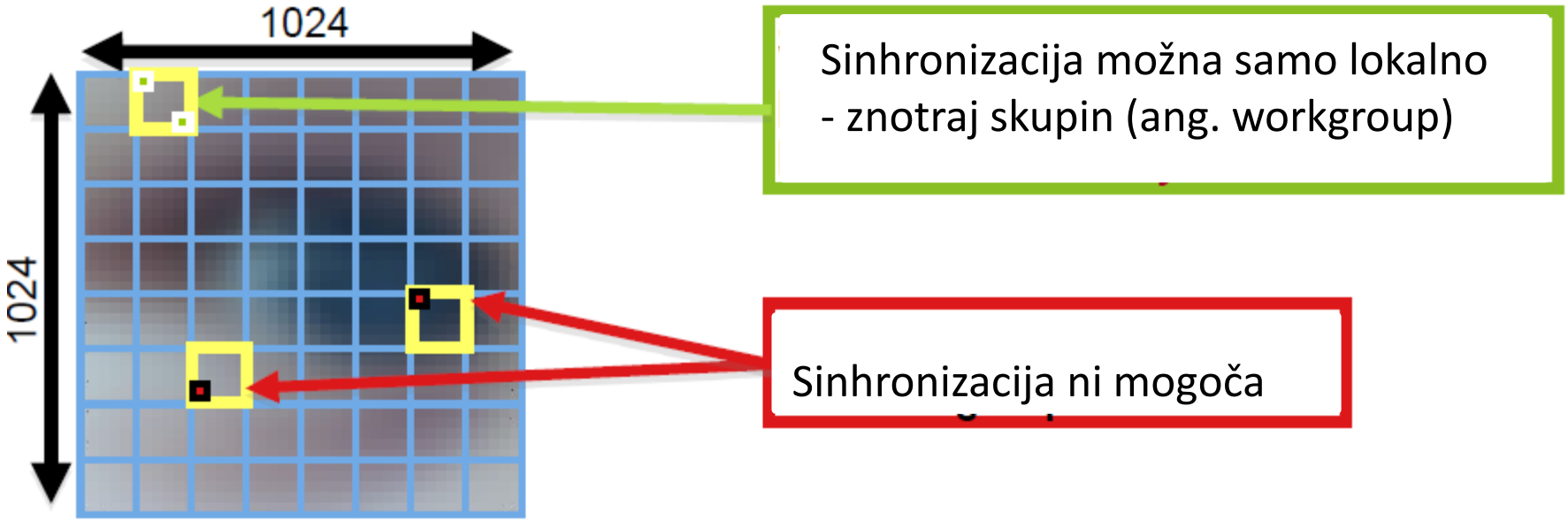
Model: ukazne vrste

- Vsi ukazi napravam so oddani preko ukaznih vrst
- Vsaka naprava ima svojo (vsaj eno) ukazno vrsto
- Več ukaznih vrst za ukaze, ki med seboj ne potrebujejo sinhronizacije
- Tipi sinhronizacije
  - in-order: po vrsti
  - out-of-order: poljuben vrstni red



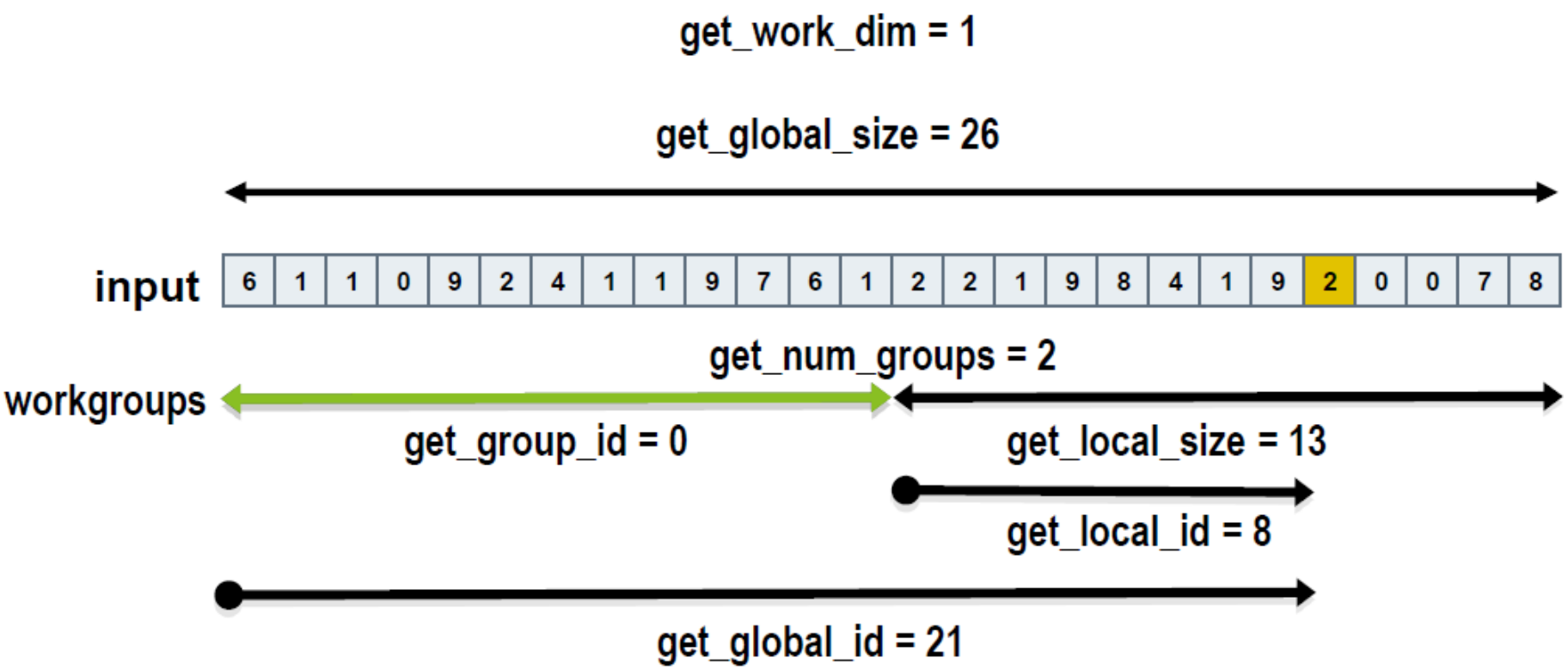
Izvajalni model: ščepec

- Definiramo problemsko področje (velikost, dimenzije)
- Za vsako točko področja izvedemo ščepec
- Primer:
  - Globalne dimenzije: 1024 x 1024
  - Lokalne dimenzije: 128 x 128



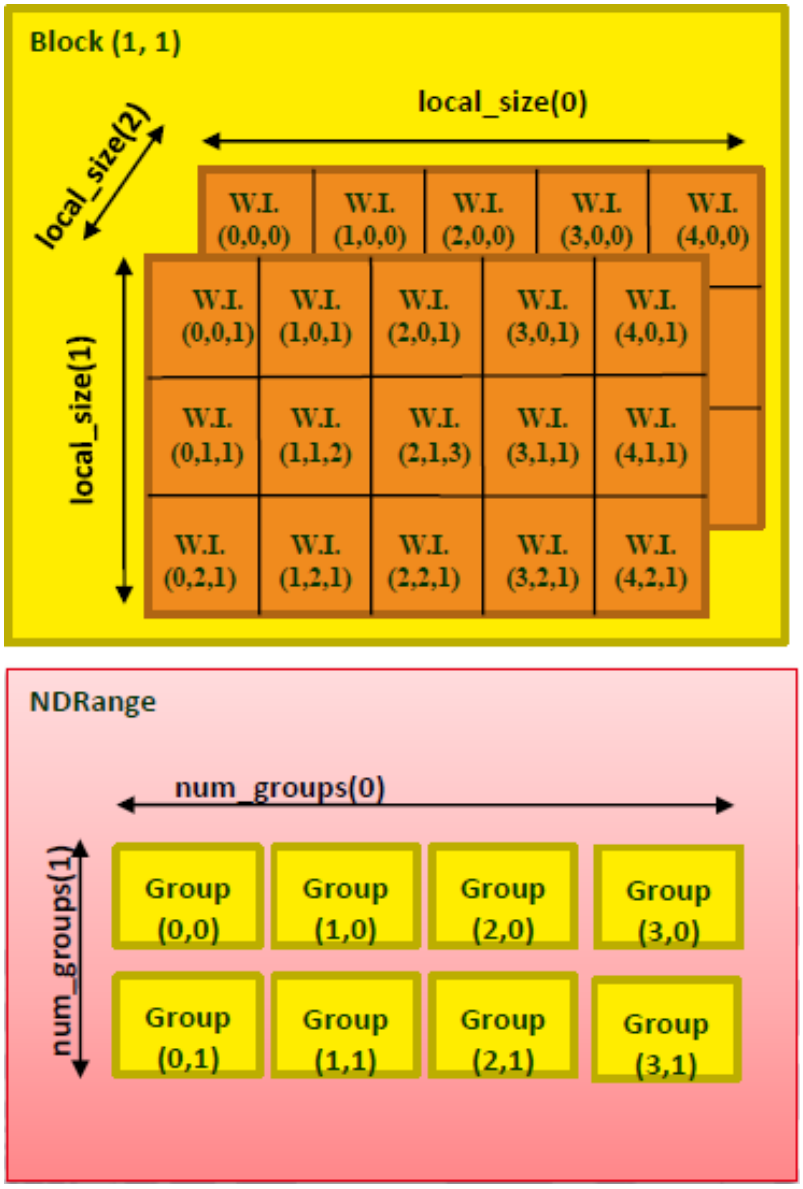
Izvajalni model: ščepec

- Naslavljanje v eni dimenziji
  - skupine (ang. workgroups)
  - elementi (ang. workitems)



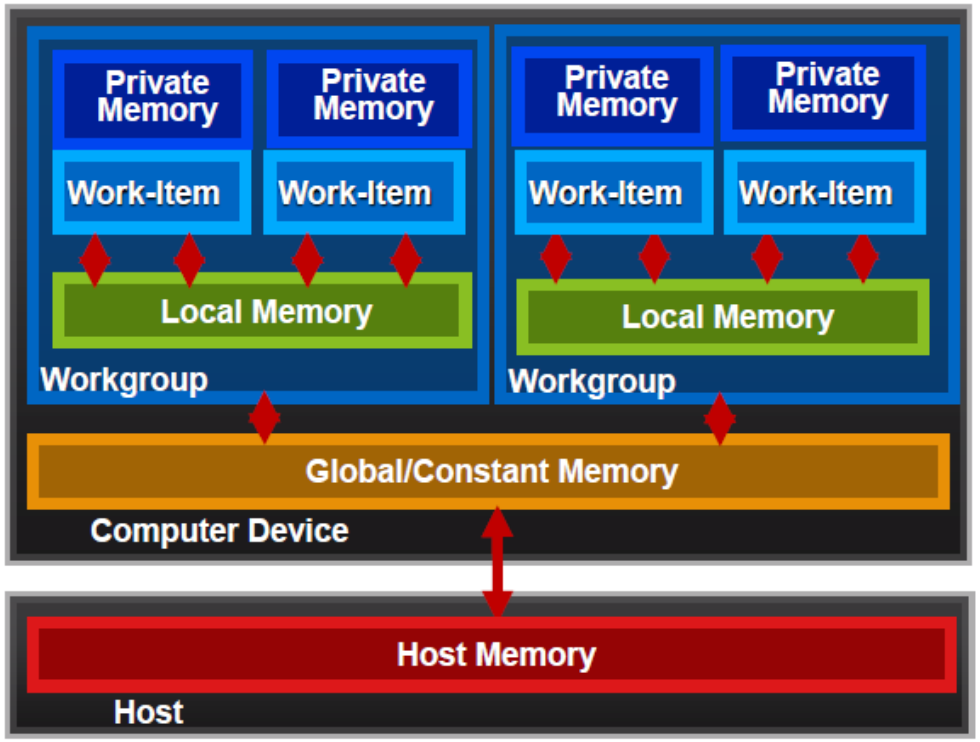
Izvajalni model: ščepec

- Naslavljanje v več dimenzijah
  - ena, dve ali tri dimenzije



Pomnilniški model

- Privatni pomnilnik (workitem)
- Lokalni pomnilnik (workgroup)
- Globalni pomnilnik
- Pomnilnik konstant
- Pomnilnik na CPU

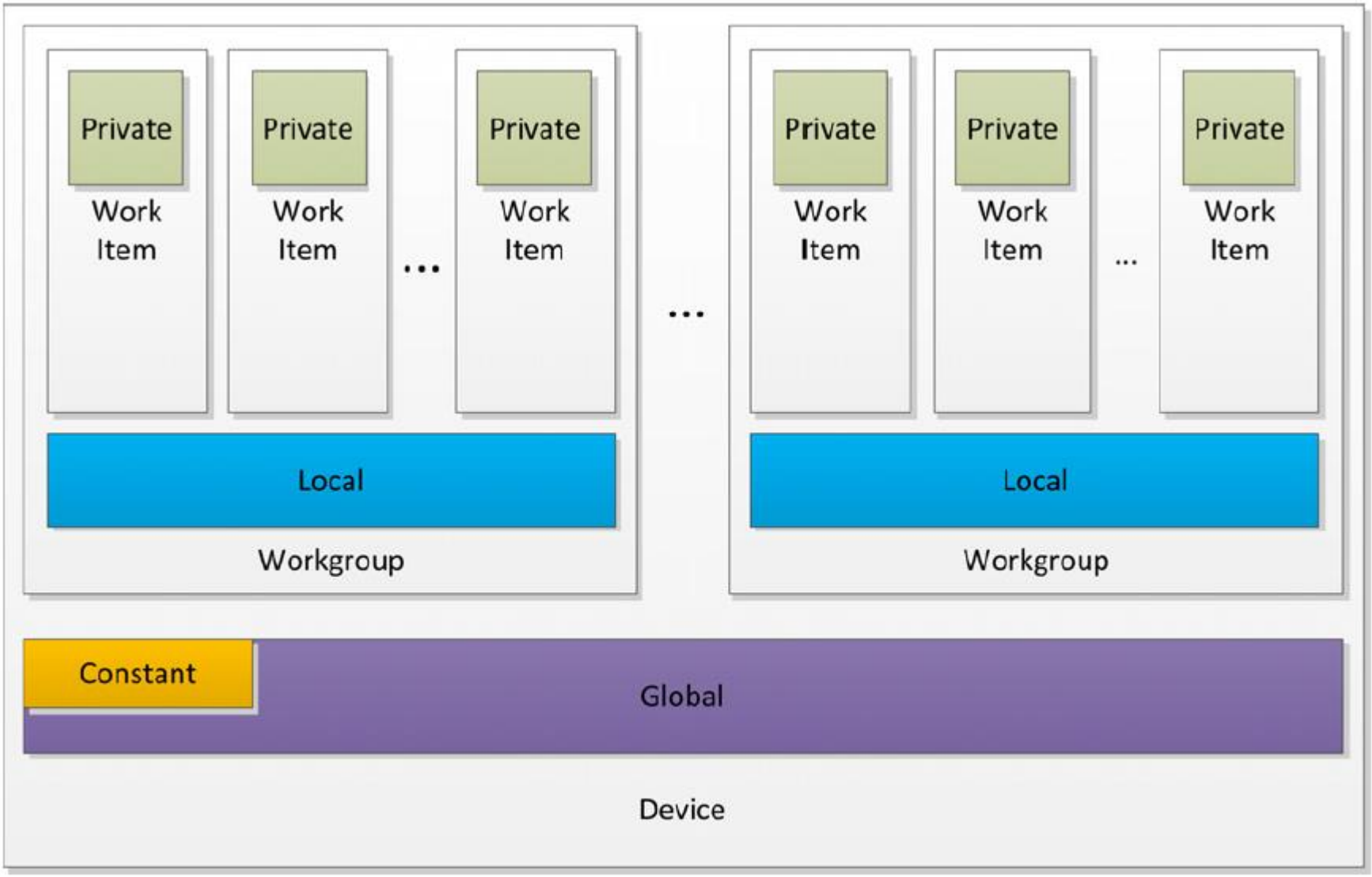


- Eksplicitno upravljanje s pomnilnikom
  - Premikanje vsebine
  - gostitelj → globalni pomnilnik → lokalni pomnilnik …

Pomnilniški model

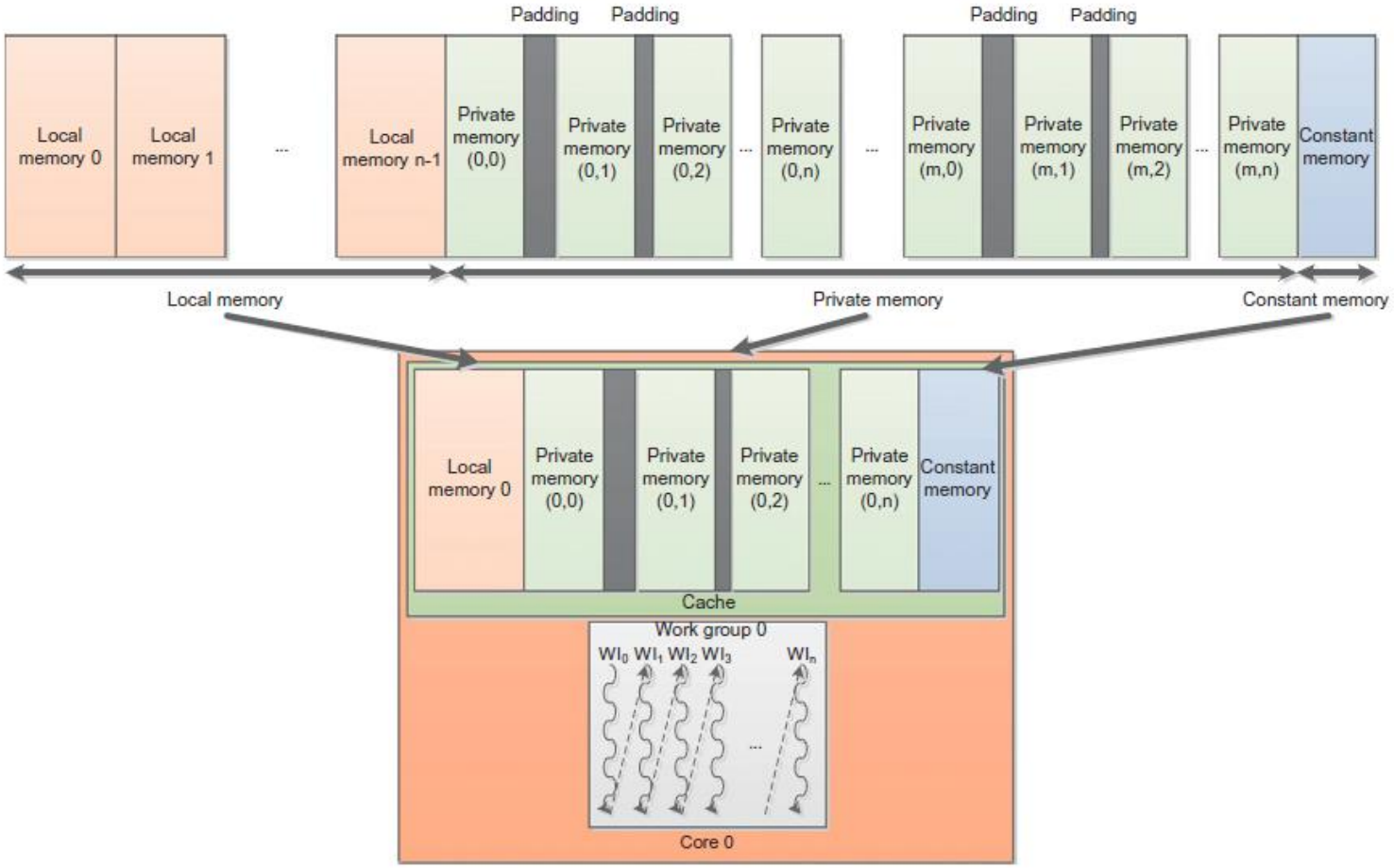
- Konsistentnost
  - Ni zagotovljeno, da bodo vse niti videle enako stanje
  - Privatni pomnilnik
    - Za posamezno nit: konsistentno pri naloži/shrani
  - Lokalni pomnilnik
    - Znotraj skupine: konsistentno ob sinhronizaciji
  - Globalni pomnilnik
    - Za posamezno skupino ob sinhronizaciji, ni zagotovljeno med skupinami
  - Konsistentnost med ukazi v ukazni vrsti
    - Zagotovljena z (vsiljeno) sinhronizacijo

Izvajalni model: ustreza GPU



Izvajalni model: CPU

- CPU in pomnilniki
  - Strnjeni bloki za ohranjanje lokalnosti predpomnilnika

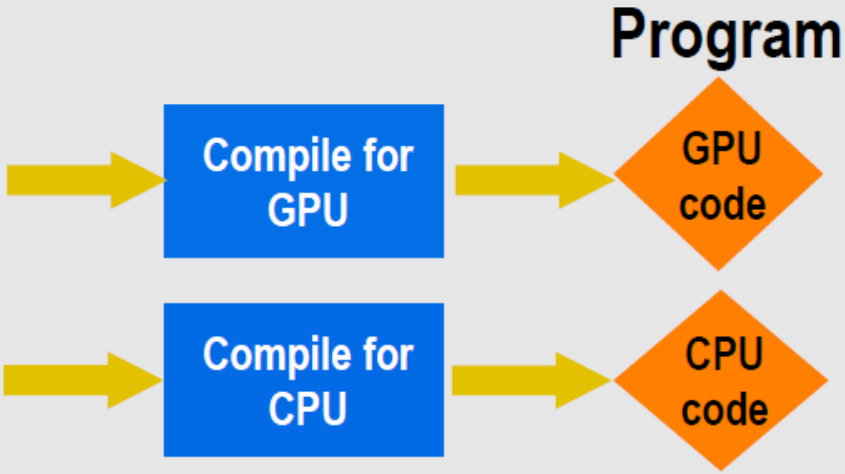


Program

- Programski objekt vsebuje
  - Kontekst
  - Programsko kodo (izvorna ali binarna)
  - Seznam ciljnih naprav in navodila prevajalniku
- Prevajanje Just In Time

Kernel Code

```
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);
}
```



OpenCL C za ščepce

- Izpeljan iz ISO C99
  - Omejitve: ni rekurzije, kazalcev na funkcije, polj in struktur spremenljive dolžine, …
- Vgrajeni podatkovni tipi
  - Skalarni in vektorski, kazalci
  - Rutine za pretvarjanje med podatkovnimi tipi
- Vgrajene funkcije
  - Funkcije za delo z nitmi in skupinami, math.h, delo s pomnilnikom
  - Racionalne, trigonometrične funkcije, sinhronizacija
- Opcijsko
  - Dvojna natančnost, atomični ukazi (globalni in lokalni pomnilnik)…

OpenCL C za ščepce

- Označevanje ščepcev: __kernel
  - ščepec lahko kliče tudi druge ščepce
- Naslovni prostor:
  - __private, __local, __global, __constant
- Funkcije za sklicevanje na problemsko področje:
  - get_work_dim(i), get_global_id(i), get_local_id(i), get_group_id(i), get_global_size(i), get_local_size(i)
  - Indeks i je 0, 1 ali 2 in označuje dimenzijo
- Funkcije za sinhronizacijo
  - Pregrade: vse niti morajo priti do pregrade preden se lahko izvajanje nadaljuje
  - Pomnilniške ograje: poskrbi za pravo zaporedje pri delu s pomnilnikom

OpenCL C za ščepce

- Vektorski podatkovni tipi
  - Dolžine vektorjev 2, 4, 8, 16
  - char2, ushort4, int8, float16, double2, …
  - Poravnavanje
  - Vektorske operacije potekajo med istoležnimi elementi, vgrajene funkcije za delo med elementi v vektorju (dot)

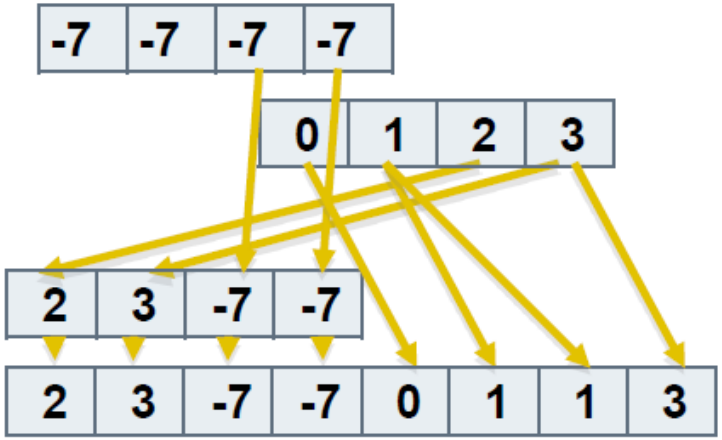
OpenCL C za ščepce

- Vektorski podatkovni tipi

- **Vector literal**

```
int4 vi0 = (int4) -7;

int4 vi1 = (int4)(0, 1, 2, 3);
```



- **Vector components**

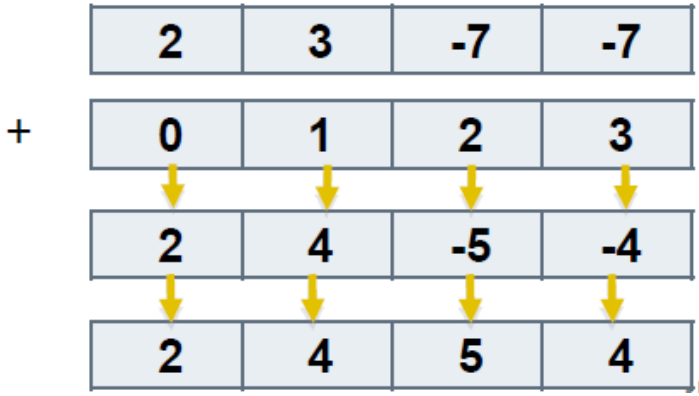
```
vi0.lo = vi1.hi;

int8 v8 = (int8)(vi0, vi1.s01, vi1.odd);
```

- **Vector ops**

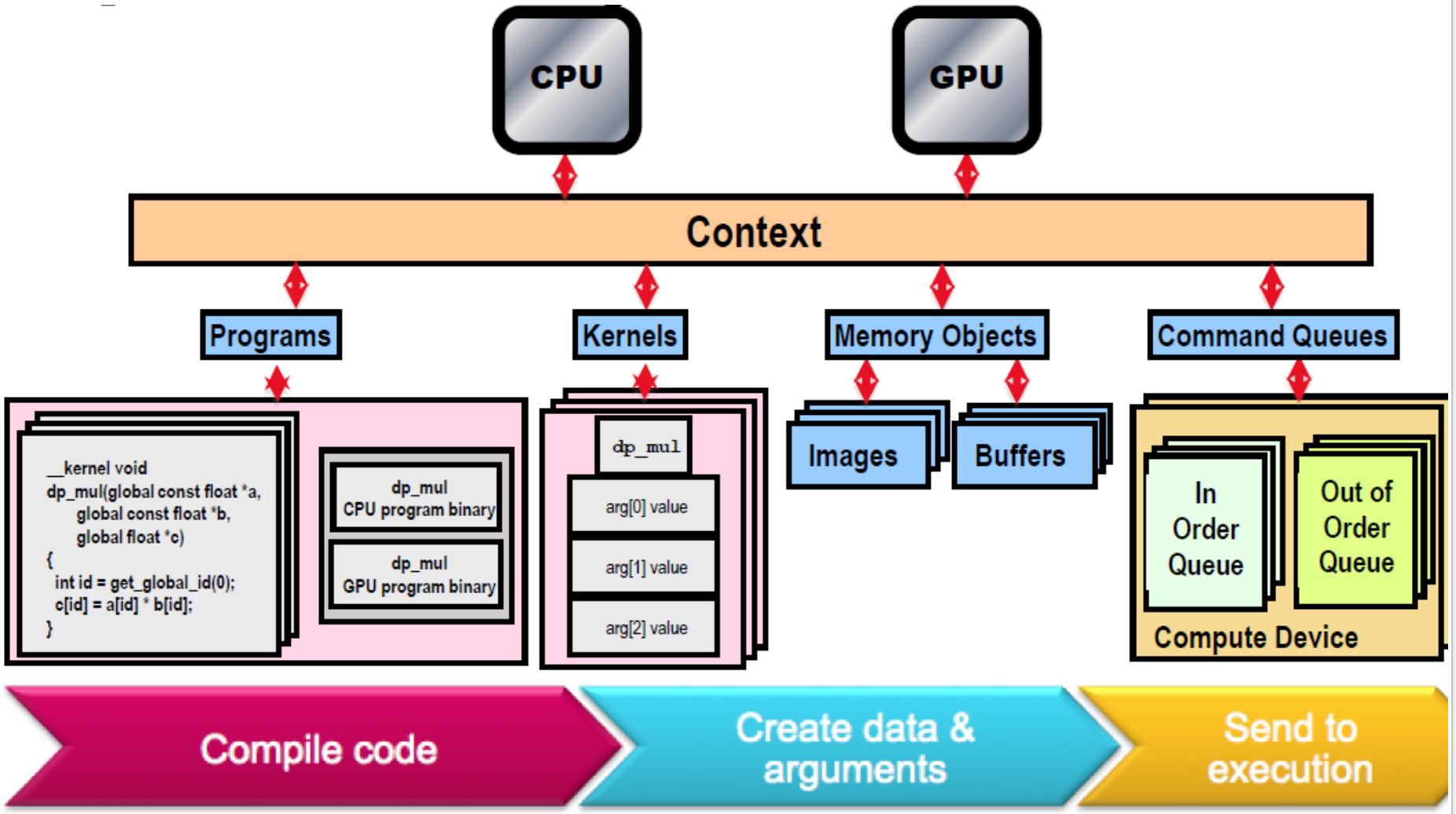
```
vi0 += vi1;

vi0 = abs(vi0);
```



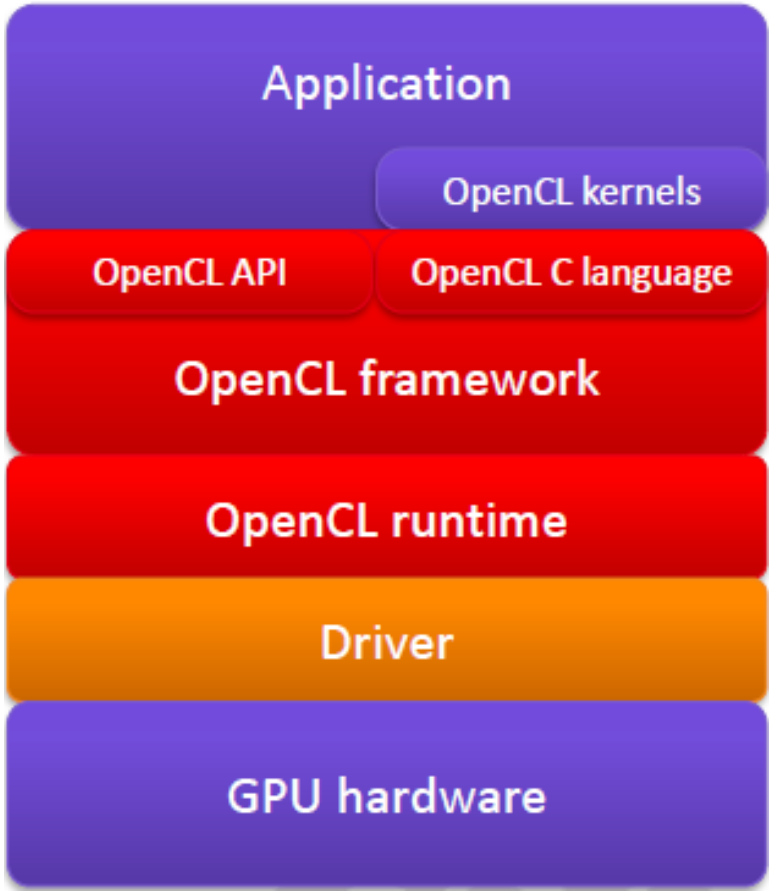
Povzetek

- Shema OpenCL



Arhitektura OpenCL

- Verzije
  - Platforma (AMD, Nvidia, ATI, …)
  - Naprava (gonilnik)
  - Jezik (najvišji podprt standard)



OpenCL aplikacija: platforma in naprave

- OpenCL platforma
  - nameščena programska opreme OpenCL na gostitelju
  - OpenCL naprave: CPU, GPU, DSP, …
  - dostopnost OpenCL naprav je odvisna od nameščenih gonilnikov na gostitelju
    - CPU nimajo gonilnikov
    - Nvidia OpenCL CPU ne podpira
- Seznam nameščenih platform
  - Vrne seznam nameščene programske opreme OpenCL
  - clGetPlatformIDs
  - clGetPlatformInfo

OpenCL aplikacija: kontekst

- Seznam naprav, ki podpirajo posamezno platformo
  - Vrne seznam naprav, ki ustrezajo platformi
  - clGetDeviceIDs
  - clGetDeviceInfo
- Za izbrano listo naprav in z njimi povezanih platform sestavimo kontekst
  - clCreateContext

OpenCL aplikacija: ukazna vrsta

- Ukazna vrsta OpenCL
  - Za razvrščanje ukazov
  - Povezana z natanko določeno napravo
  - Če imamo več vrst, delujejo neodvisno, smiselno za ukaze, ki ne potrebujejo sinhronizacije
  - IN_ORDER, OUT_OF_ORDER
  - clCreateCommandQueue
- Sinhronizacija
  - clFlush
  - clFinish

OpenCL aplikacija: delo s pomnilnikom

- Dva tipa objektov
  - Buffer: za 1D strukture
  - Image: za 2D in 3D strukture
- Prenos podatkov je ekspliciten, podati je potrebno
  - Kazalec na gostitelju
  - Kazalec na napravi
  - Količino podatkov
  - Ukazno vrsto
  - Tip prenosa: blokirajoč / ne blokirajoč
- Ukaza
  - clEnqueueWriteBuffer
  - clEnqueueReadBuffer

OpenCL aplikacija: program in ščepec

- Program OpenCL je sestavljen iz ščepcev
- Programski objekt sestavljajo
  - Kontekst
  - Programska koda
  - Ščepec je predstavljen kot poseben objekt (clKernel)
- Dva načina za pripravo programa
  - Izvorna koda
  - Binarna koda
- Ščepec je označen s predpono __kernel

OpenCL aplikacija: program in ščepec

- Izvorna koda je podana kot niz znakov
  - Vsebuje ščepec
  - Prebere in prevede se ga med izvajanjem
  - Lahko spreminjamo ščepec brez prevajanja osnovnega programa!
  - clCreateProgramWithSource
- Prevajanje
  - Prevajanje za vse naprave v kontekstu
  - clBuildProgram
- Objekt tipa ščepec
  - Je odvisen od programskega objekta
  - clCreateKernel

OpenCL aplikacija: zagon ščepca

- Nastavitev argumentov
  - Potrebno zaradi prevajanja med izvajanjem
  - clSetKernelArg
- Dva načina
  - Za opravilni paralelizem, več ščepcev hkrati (clEnqueueTask)
  - Za podatkovni paralelizem (clEnqueueNDRangeKernel)
    - Nastavitev problemskega prostora
    - Nastavitev velikost skupine
  - Zagon ščepcev je asinhron

OpenCL aplikacija: povzetek

- Izberemo platformo
- Izberemo naprave
- Definiramo kontekst
- Pripravimo ukazno vrsto za napravo
- Alociramo pomnilnik na napravi
- Skopiramo podatke iz gostitelja na napravo
- Pripravimo program in ga prevedemo
- Pripravimo ščepec z argumenti in ga zaženemo
- Skopiramo rezultate iz naprave na gostitelja
- Pospravimo za sabo