UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Gašper Forjanič

Grafična procesna enota kot procesor za splošne namene

DIPLOMSKO DELO
NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: doc. dr. Janez Demšar

Ljubljana, 2010

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00496/2010

Datum: 15.01.2010

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **GAŠPER FORJANIČ**

Naslov: **GRAFIČNA PROCESNA ENOTA KOT PROCESOR ZA SPLOŠNE NAMENE**

USING GRAPHICS PROCESSING UNIT AS A GENERAL PURPOSE PROCESSOR

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Z razvojem računalniške grafike, predvsem v namene čim bolj realističnih prikazov in efektov v računalniških igrah, se v zadnjem času zelo povečuje moč grafičnih procesnih enot na zmogljivejših grafičnih karticah. Ti procesorji so zmožni visoke stopnje paralelnosti in zato v načelu uporabni tudi za reševanje splošnih problemov, če jih je mogoče primerno paralelizirati.

V diplomski nalogi opišite področje, podajte nekaj primerov uporabe in analizirajte časovne prihranke, ki jih pridobimo s tem, ko določene izračune prenesemo s centralne procesne enote na grafično.

Mentor:

doc. dr. Janez Demšar

Dekan:

prof. dr. Franc Solina

Namesto te strani vstavite original izdane teme diplomskega dela s podpisom mentorja
in dekana ter țigom fakultete, ki ga diplomant dvigne v študentskem referatu, preden
odda izdelek v vezavo!

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisani/-a **Gašper Forjanič**,

z vpisno številko **63040031**,

sem avtor/-ica diplomskega dela z naslovom:

Grafična procesna enota kot procesor za splošne namene

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom (naziv, ime in priimek)

  doc. dr. Janeza Demšarja

  in somentorstvom (naziv, ime in priimek)

- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«.

V Ljubljani, dne **16.04.2010** Podpis avtorja/-ice:

Zahvala

Zahvaljujem se doc. dr. Janezu Demšarju za mentorstvo in pomoč pri izdelavi diplomskega dela, ter kolegu Marku Dolencu za strokovne nasvete.
Posebna zahvala gre tudi družini, ki mi je vse to omogočila in mi čez celoten študij stala ob strani.

Kazalo

Povzetek 1
Abstract 2
1. Uvod 3
2. Primerjava med CPE in GPE 4
3. Heterogeno procesiranje 8
4. Arhitektura CUDA naprave 9
4.1 GTX200b (GeForce GTX 295) 9
4.2 GF100 (GeForce GTX 480) 12
4.3 Tehnologija SLI 13
4.3.1 3DFX SLI 13
4.3.2 NVidia SLI 15
4.4 Mehčanje robov 23
4.4.1 Fullscreen Anti-Aliasing 24
4.4.2 Multisample Anti-Aliasing 25
4.4.3 Coverage Sampled Anti-Aliasing 26
5. Programski model CUDA 28
5.1 CUDA funkcije 28
5.2 Hierarhija niti 29
5.3 Hierarhija pomnilnika 31
5.3.1 Pomnilnik CUDA naprave 33
5.3.2 Globalni pomnilnik 36
5.3.3 Lokalni pomnilnik 37
5.3.4 Skupni pomnilnik 37
5.3.5 Registri 39
5.3.6 Teksturni pomnilnik 39
5.3.7 Konstantni pomnilnik 39
5.4 Različice programskega modela GPE 39
5.5 Vgrajene spremenljivke 41
5.6 Tipi CUDA funkcij 41
6. Primer programiranja v CUDA okolju 43
6.1 Rezervacija pomnilnika 43

6.2 Inicializacija in prenos podatkov ... 43
6.3 Klic kernel funkcije ... 44
6.4 Kernel funkcija ... 44
6.5 Prenos rezultatov nazaj v CPE ... 45
6.6 Izpis rezultatov ... 45
6.7 Priprava razvojnega okolja Visual Studio 2008 ... 46
7. Analiza zmogljivosti ... 51
7.1 Strojna oprema ... 51
7.2 Programska oprema ... 54
7.3 HOOMD Blue Edition ... 54
7.4 BarsWF MD5 ... 58
7.5 Parboil Benchmark Suite ... 62
8. Zaključek ... 68
Priloge ... 70
Compute Capability ... 70
Programska koda ... 72
Kazalo slik ... 73
Kazalo tabel ... 74
Viri ... 74

Seznam kratic

CPE - Centralna Procesna Enota
GPE - Grafična Procesna Enota
CUDA - Compute Unified Device Architecture
MSDNAA - Microsoft Developer Network Academic Alliance
OpenCL - Open Computing Language
OpenGL - Open Graphics Library
GPGPU - General-Purpose Computation on Graphics Processing Units
SDK - Software Development Kit
NVCC - NVidia C Compiler
CUFFT – CUDA Fast Fourier Transform
CUBLAS – CUDA Basic Linear Algebra Subprograms
FLOP/s - FLoating point Operations Per Second
HT - HyperThreading
TSMC - Taiwan Semiconductor Manufacturing Company
PCB - Printed Circuit Board
TDP – Thermal Design Power
SP - Streaming Processor
SM - Streaming Multiprocessor
TPC - Texture Processing Cluster
GPC - Graphics Processing Cluster
TMU - Texture Mapping Unit
SFU - Special Function Unit
ROP - Raster Operation Processor
SLI (3DFX) - Scan-Line Interleave
SLI (NVidia) - Scalable Link Interface
API - Application Programming Interface
VGA - Video Graphics Array
CRT - Cathode Ray Tube
IGP - Integrated Graphics Processor
RGB – Red Green Blue (barvna paleta)
RAM - Random Access Memory
SD RAM – Synchronous Dynamic RAM
ECC – Error Correcting Code
DDR RAM- Double Data Rate RAM
GDDR RAM - Graphics DDR RAM
OC - OverClock
VCS - Visual Computing System
VHDCI - Very High Density Cable Inteconnect
AA - Anti-Aliasing
MD5 - Message Digest algorythm 5
SIMD - Single Instruction, Multiple Data
SIMT - Single Instruction, Multiple Thread
SSE - Streaming SIMD Extensions

Povzetek

Čedalje večje in podrobnejše teksture, fotorealistični efekti in visoke ločljivosti prikazovanja, so privedli do velikih količin podatkov, potrebnih vzporedne obdelave. Posledica tega je eksponentna rast zmogljivosti grafične procesne enote (GPE), še posebno v primerjavi s centralno procesno enoto (CPE).
Zmogljivost vzporedne obdelave je glavni vzrok, da se je GPE razvila v procesor za splošne namene. Kot primer smo si ogledali arhitekturo GPE na NVidiinem čipu prejšnje in sedanje generacije. Podrobno je opisan sistem vezave dveh ali večih grafičnih kart (Scalable Link Interface - SLI). Strojno mehčanje robov, ki predstavlja eno izmed najbolj procesorsko zahtevnih grafičnih operacij, je bilo izpostavljeno kot primer prikaza zmogljivosti GPE.

Ogledali smo si programski model okolja CUDA in z osvojenim znanjem napisali program. Ker je osnova okolja CUDA programski jezik C / C++, je učenje za izkušenega C programerja hitro in učinkovito, največ težav pa lahko predstavlja optimizacija uporabe pomnilnika ter pravilna razdelitev blokov in niti znotraj le teh.

Cilj diplomske naloge je dokazati zmogljivosti GPE na področjih, kjer se najpogosteje uporablja CPE. V ta namen smo v zadnjem poglavju analizirali zmogljivosti CPE in GPE v različnih benchmark aplikacijah, v Windows in Linux okolju. Analiza je temeljila na treh centralnih procesorjih Intel (Q6600, i7 920 in i7 980X) in dveh grafičnih procesorjih NVidia, najšibkejšem GeForce 8400GS in trenutno najmočnejšem GeForce GTX 295.

Ključne besede: vzporedno računanje na GPE, GPGPU, CUDA, SLI

Abstract

Over the past few years, we have seen an exponential performance boost of the graphics processing unit (GPU) compared to the central processing unit (CPU). Reason for that are larger and more detailed textures, photorealistic effects and high display resolutions, which require parallel processing of large amounts of data. This caused the GPU to evolve into the general purpose processor (GPP). We took a look at the GPE architecture on the latest and previous generations of NVidia GPUs and the multi-GPU solution for linking two or more graphics cards (Scalable Link Interface - SLI). As an example of GPU performance, we pointed out hardware anti-aliasing technique, which is one of the most processor demanding graphics operations.

We examined the CUDA programming model and used the acquired knowledge to write an example of a program using the CUDA model. CUDA is based on programming language C / C++, so the learning curve for experienced C programmer is quick and efficient. There may be difficulties with memory optimizations and with finding the best suitable number of blocks and threads.

The purpose of this dissertation is to test the GPU performance within the CPU application domain using benchmark applications on Windows and Linux operating system. The test equipment was based on three Intel CPUs (Q6600, i7 920 and i7 980X) and two NVidia GPUs with low-end GeForce 8400GS and high-end GeForce GTX 295.

Keywords: parallel computing on GPU, GPGPU, CUDA, SLI

1. Uvod

V diplomski nalogi se bomo omejili na dve GPE proizvajalca NVidia, modela GT200b in G86, in ju uporabili kot procesor za splošne namene. Arhitektura grafičnih procesorjev se od klasičnih centralnih procesorjev razlikuje predvsem v večjem številu jeder, ki predstavljajo osnovo paralelnega procesiranja. Trenutno ima NVidia najbolj dovršeno okolje za izrabo zmogljivosti svojih GPE za splošne namene, z imenom CUDA[1]. Predstavljena je bila novembra 2006 in je na voljo za vse grafične procesorje z arhitekturno oznako G80 in naprej (modeli GeForce 8 in naprej, Quadro in Tesla).

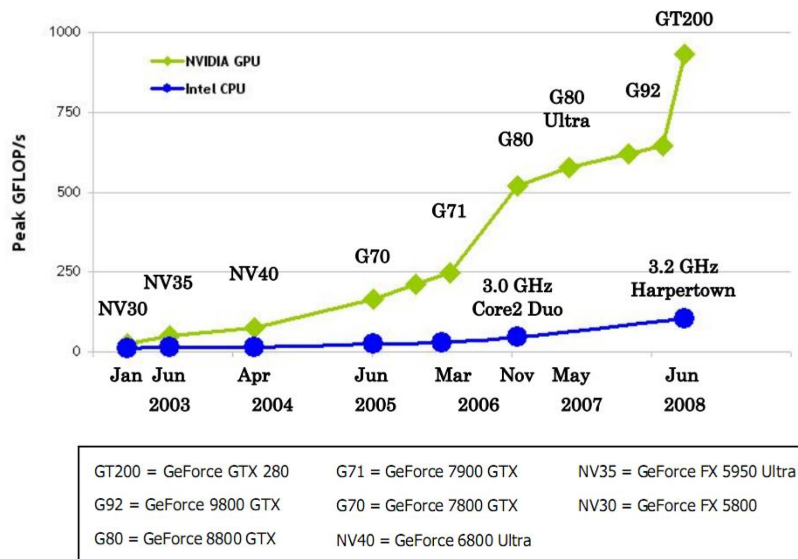
Razlog za omejitev na NVidio temelji na zelo dobri podpori tako s strani dokumentacije, kot tudi programske implementacije. Okolje je razvijalcem na voljo preko računalniškega jezika C++, imenovanega *C for CUDA*. Jezik ni samostojen, temveč je le dodatek najbolj razširjenemu in poznanemu jeziku C++. CUDA podpira tudi programiranje v programskih jezikih DirectCompute, OpenCL in Fortran.

Zanimale nas bodo predvsem prednosti uporabe GPE pred CPE, zato si bomo s strojnega stališča ogledali arhitekturo GPE in vzporedno vezavo dveh ali večih GPE. Za ponazoritev količine podatkov, ki jih je GPE sposobna obdelati, bo podrobno opisana tehnika mehčanja robov. Spoznali bomo osnove programskega modela CUDA in napisali CUDA program v razvojnem okolju Visual Studio 2008.

Ker nas zanima predvsem zmogljivost GPE v ne-grafičnih aplikacijah, mišljenih za CPE, bomo v zadnjem poglavju analizirali zmogljivost obeh procesnih enot. Poskušali bomo dokazati, da je GPE zrela za zamenjavo CPE na področju paralelnega procesiranja podatkov.

2. Primerjava med CPE in GPE

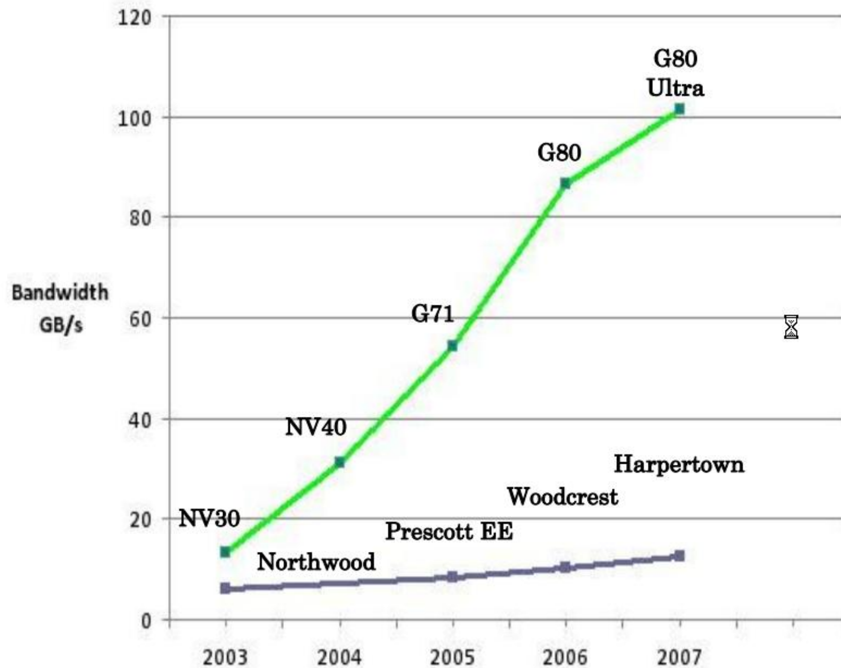
Z visokimi zahtevami predvsem igričarskega trga, kot so realistični efekti in visoko-ločljivostne (HD) teksture, ki že mejijo na filmsko montažo, se je GPE razvila v več-jedrni procesor za splošne namene, ki se ponaša z visoko stopnjo paralelnega izvajanja, veliko pasovno širino pomnilnika ter izjemno zmogljivostjo računanja. Slika 1 kaže teoretično primerjavo zmogljivosti CPE in GPE, medtem ko Slika 2 prikazuje občutno višji pretok podatkov preko pomnilniškega vodila na GPE.



Slika 1: Teoretična primerjava zmogljivosti CPE in GPE [GFLOP/s].

(vir: NVidia CUDA Programming Guide 2.3)

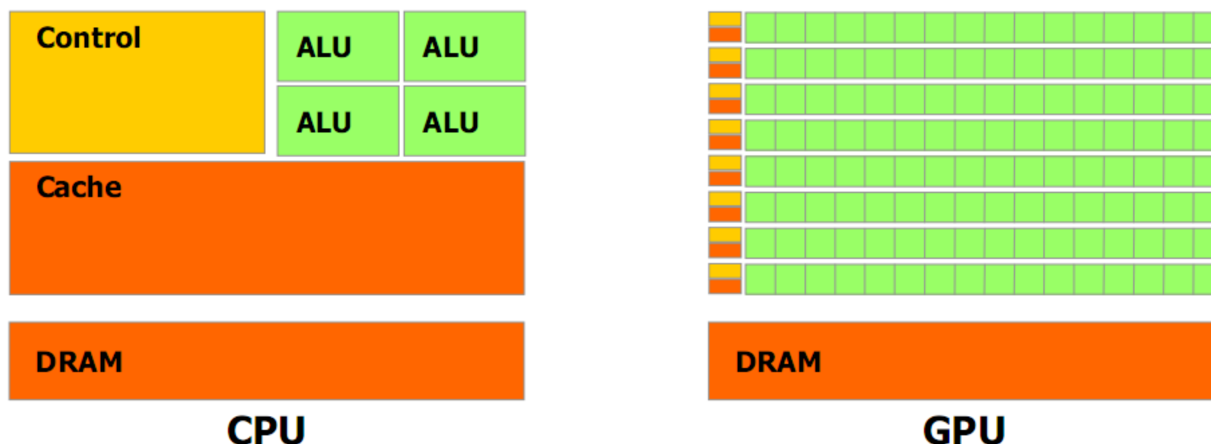
Opazi se velik preskok s strani GPE pri čipu s kodno oznako G80, ki je tudi prvi uradni čip, podprt za okolje CUDA.



Slika 2: Primerjava pasovne širine pomnilnika CPE in GPE [GB/s].
(vir: NVidia CUDA Programming Guide 2.3)

Vzrok za tako razliko med CPE in GPE pri računanju s plavajočo vejico je v sami arhitekturi mikroprocesorja (Slika 3). Osnovna naloga GPE je izračun ter prikaz pik (ang. *pixel*) na ekranu, kar pa predstavlja več ali manj ponovljive procedure z ogromno podatki. Tu nastopi paralelna oz. vzporedna obdelava podatkov. Skorajda vsaka aplikacija, ki procesira večje količine podatkov, bi lahko to procesirala vzporedno, kar pripelje do izrabe grafičnega procesorja za t.i. splošne namene, GPGPU[3]. Potreben je le še vmesnik, ki bi obdelavo poslal na GPE, in ne več na CPE. Večjedrni procesorji so sicer do neke mere učinkoviti, vendar s svojimi kompleksnimi seti ukazov ostajajo neizkoriščeni. Poleg tega imajo na voljo le par jeder, trenutno največ 6 (12) pri Intel Core i7 980X (HT, HyperThreading), kar pa je v primerjavi z grafičnimi procesorji zanemarljivo. Šele ko je govora o večprocesorskih in večjedrnih rešitvah, se moči GPE približamo, se pa oddaljimo iz finančnega vidika.

Primerjavo zmogljivosti med CPE in GPE si bomo ogledali v 7. poglavju.



Slika 3: GPE ima na voljo več aritmetično-logičnih enot za obdelavo podatkov.
(vir: NVidia CUDA Programming Guide 2.3)

Ugotovimo lahko, da se GPE odlično izkaže pri programih, ki se mnogokrat izvajajo na različnih podatkih. S tem odpade potreba po velikem predpomnilniku, saj je dostopni čas do pomnilnika skrit (upoštevan) že v samem računanju. Najnovejši čip GF100 (marec 2010), bolj znan po arhitekturi *Fermi* (Enrico Fermi, it. fizik), je najzmoglivejši in tudi fizično največji grafični procesor v tem trenutku. Narejen je v 40nm proizvodnem procesu, tako kot njegov najresnejši konkurent iz ATI/AMD tabora, Radeon HD5870 oz. HD5970 (dvoprocesorski). Čipi se bodo proizvajali v tajvanskem TSMCju, ki je zaenkrat tudi edina tovarna, sposobna masovne proizvodnje v 40nm procesu. GF100 kot največji čip zavzame 529 mm^2 tiskanine in ob obremenitvi porabi četrt kW električne energije. Za primerjavo, najnovejši Intelov procesor Core i7 980X zavzema vsega 240mm^2 in porabi cca. 130W električne energije, za razliko pa je izdelan v manjšem, 32nm proizvodnem procesu. Ker pa ni dobro primerjati hrušk in jabolk, so v Tabeli 1 zbrani trenutno znani podatki o GF100 čipu v primerjavi s dosedanjimi modeli, osnovanimi na čipu GT200b.

	GTX 260	GTX 275	GTX 285	GTX 295	GTX 480
Kodno ime	GT200b	GT200b	GT200b	GT200b	GF100
Proizvodni proces [nm]	55	55	55	55	40
Velikost čipa [mm^2]	470	470	470	2 x 470	529
Število tranzistorjev [mio]	1,4	1,4	1,4	2 x 1,4	3,2
Frekvenca jedra [MHz]	576	633	648	576	700
Frekvenca SM [MHz]	1242	1404	1476	1242	1400
Frekvenca pomnilnika [MHz]	999	1164	1242	999	924
Količina pomnilnika [MB]	896	896	1024	1796	1536
Tip pomnilnika	GDDR3	GDDR3	GDDR3	GDDR3	GDDR5
Pomnilniško vodilo [bit]	448	448	512	2 x 448	384
Število SP enot	216	240	240	2 x 240	480
Število ROP enot	28	28	32	2 x 28	48
Število TMU enot	80	80	80	2 x 80	60
TDP [W]	171	219	183	289	250

Tabela 1: Uradne specifikacije modelov, osnovanih na čipu GT200b in GF100.

3. Heterogeno procesiranje

Heterogeni računalniški sistemi predstavljajo hibridno vrsto računalniških sistemov, ki za procesiranje podatkov uporabljajo različne vrste računalniških enot[4]. Na grobo jih delimo na:

- GPP (general-purpose processor oz. procesor za splošne namene)
- SPP (special-purpose processor oz. procesor za posebne namene)
- ko-procesorji
- ASIC (Application Specific Integrated Circuit)
- FPGA (Field-Programmable Gate Array)

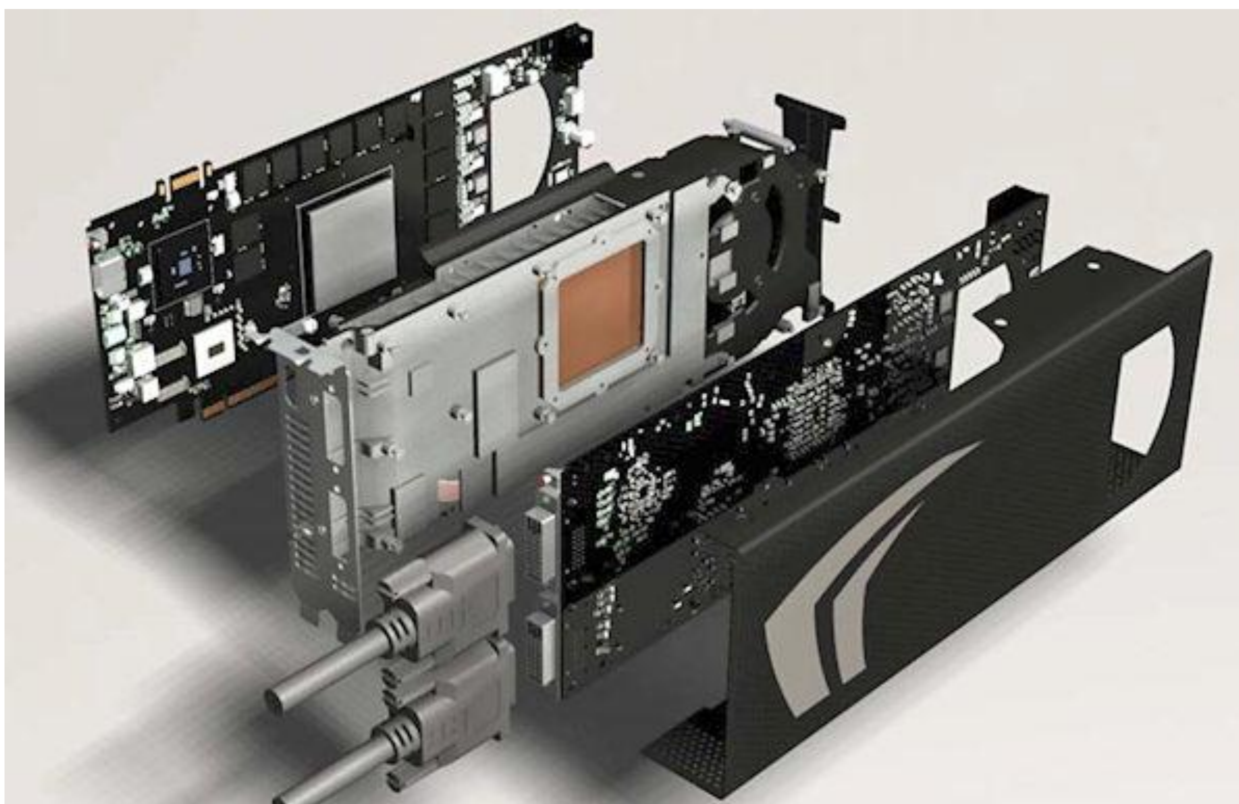
Bistvo heterogenih računalniških sistemov je, da procesorji z različnimi zbirkami ukazov sestavljajo na zunaj enoten sistem. Tako dobimo visoko zmogljivostne sisteme, ki so v današnjem času potrebni na večini specializiranih področij procesiranja. Kar se je v preteklosti reševalo s silovitim napredkom v povečavanju delovne frekvence procesorja, kot ga je, in ga še vedno v neki meri določuje Moore-ov zakon, se danes rešuje predvsem z učinkovito izrabo posameznih procesnih enot, večjedrnih in večprocesorskih rešitev. Razvijalci aplikacij s tem dobijo odprte roke pri razvoju paralelnega procesiranja podatkov v istem časovnem intervalu, prav tako pa niso več omejeni le na CPE. Podatki se preusmerijo na procesno enoto (npr. GPE), kjer jih obdelajo hitreje, kot bi jih sicer CPE.

4. Arhitektura CUDA naprave

4.1 GTX200b (GeForce GTX 295)

Ker diplomska naloga temelji na grafičnem čipu GT200b, natančneje na dvoprocesorski izvedbi NVidia GeForce GTX 295, bo v nadaljevanju podrobneje opisana arhitektura in samo delovanje karte ter njene zmožnosti.

Kot prikazuje Slika 4, sta osnova karte GTX 295 dva čipa GT200b, ločena vsak na svoji 10-slojni tiskanini (*DuaPCB*). Takega procesa izdelave se poslužujejo že od začetka njihove (oz. prevzete od podjetja 3DFX) tehnologije Scalable Link Interface (SLI), h kateri se bomo kaneje še vrnili.



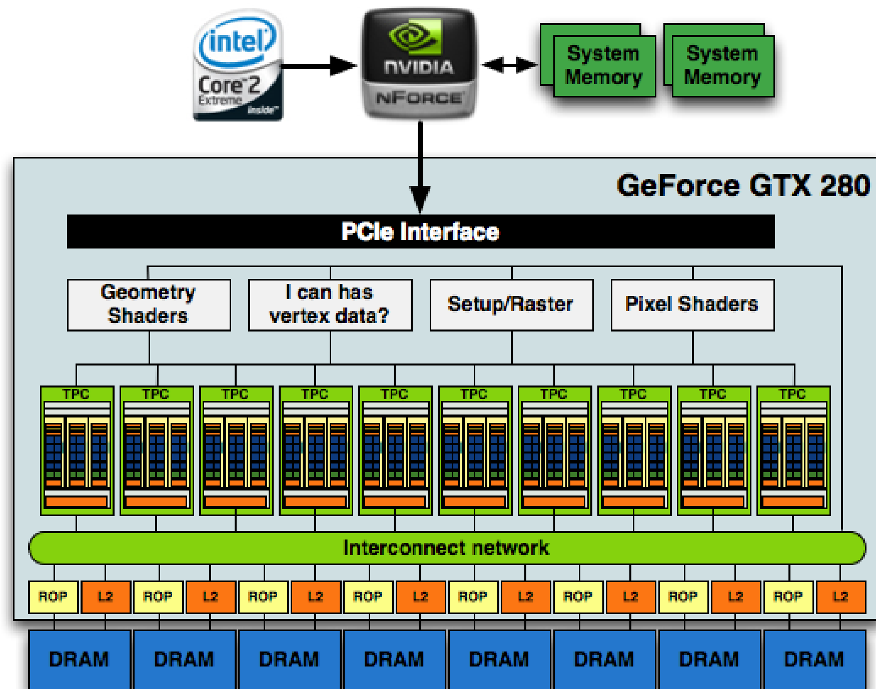
Slika 4: GeForce GTX 295, sestavljen iz dveh tiskanin (*DualPCB*).
(vir: www.guru3d.com)

Kasneje je NVidia izdala t.i. *SinglePCB* (ang. *Single Printed Circuit Board*) izvedbo karte, kjer sta oba čipa na eni sami tiskanini (Slika 5). Tak način izdelave ima kanadski ATI (v lasti AMDja) sicer v navadi že vse od njihovega prvega dvoprocesorskega modela HD3870X2 (januar 2008)[5]. Prednosti takega razvoja so predvsem nižji stroški izdelave, manj pregrevanja in posledično daljša življenjska doba izdelka, ter manjše število okvar.

Slika 5: Manj pregrevanja, nižji stroški izdelave. GTX 295 *SinglePCB*.
(vir: www.dvhardware.net)

GT200b je zasnovan na predhodniku GT200, ki je bil kupcem na voljo junija 2008[6]. Sama arhitektura je torej za računalniško branžo dokaj stara, kljub temu pa v tem trenutku še vedno predstavlja drugo najmočnejšo grafično karto na tržišču, takoj za svežim, DirectX 11 certificiranim Radeon-om HD5970. Ko govorimo o GT200b, le-ta predstavlja B3 revizijo čipa, medtem ko revizija A2 predstavlja predhodnico, GT200. Razlik v sami zgradbi arhitekture procesorja ni, bistvena sprememba je v proizvodnem procesu, ki se je iz 65nm zmanjšala na

55nm. To v praksi pomeni manjše pregrevanje čipa, izdelovalcu pa omogoča povišanje delovnih frekvenc tako jedra (ang. *Core Clock*), kot tudi senčilnih procesorjev (ang. *Shader Processor Clock*). Na ta način je karta sposobna preračunati večje število podatkov, kar pri končnem uporabniku pomeni hitrejše (krajše) izvajanje zahtev. Arhitektura čipa GT200 je pri NVidii poimenovana kot *Scalable Processor Array*[7] (SPA, Slika 6).



Slika 6: Arhitektura (SPA) grafičnega čipa GT200(b).
(vir: www.anandtech.com)

Vsebuje 10 gruč za obdelavo tekstur (ang. *Texture Processing Cluster*, TPC), vsaka izmed njih ima tri pretočne multiprocesorje (ang. *Streaming Multiprocessor*, SM), ki imajo vsak po 8 pretočnih procesorjev (ang. *Streaming Processor,* SP). Za polnokrvni GT200(b) to pomeni, da ima na razpolago 240 SPjev za obdelavo podatkov. Vsak TPC ima 8 enot za mapiranje tekstur (ang. *Texture Mapping Unit,* TMU). GT200 lahko z njimi v eni urini periodi s teksturami olepša 80 pikslov, kar s taktom jedra 576 MHz pomeni 46,1 milijarde tekslov na sekundo. V primeru GTX 295, to pomeni krat dva. Tu so še rasterizacijske enote (ang. *Raster Operation Processor*, ROP), ki skrbijo za pravilen prikaz dejanskih pik (ang. *pixel*) na izhodu (monitor, TV, projektor, itd.). Teh je na modelu GTX 285 dvaintrideset, medtem ko so na šibkejših modelih štiri enote lasersko porezane, tako da so čipi dejansko oskubljeni.

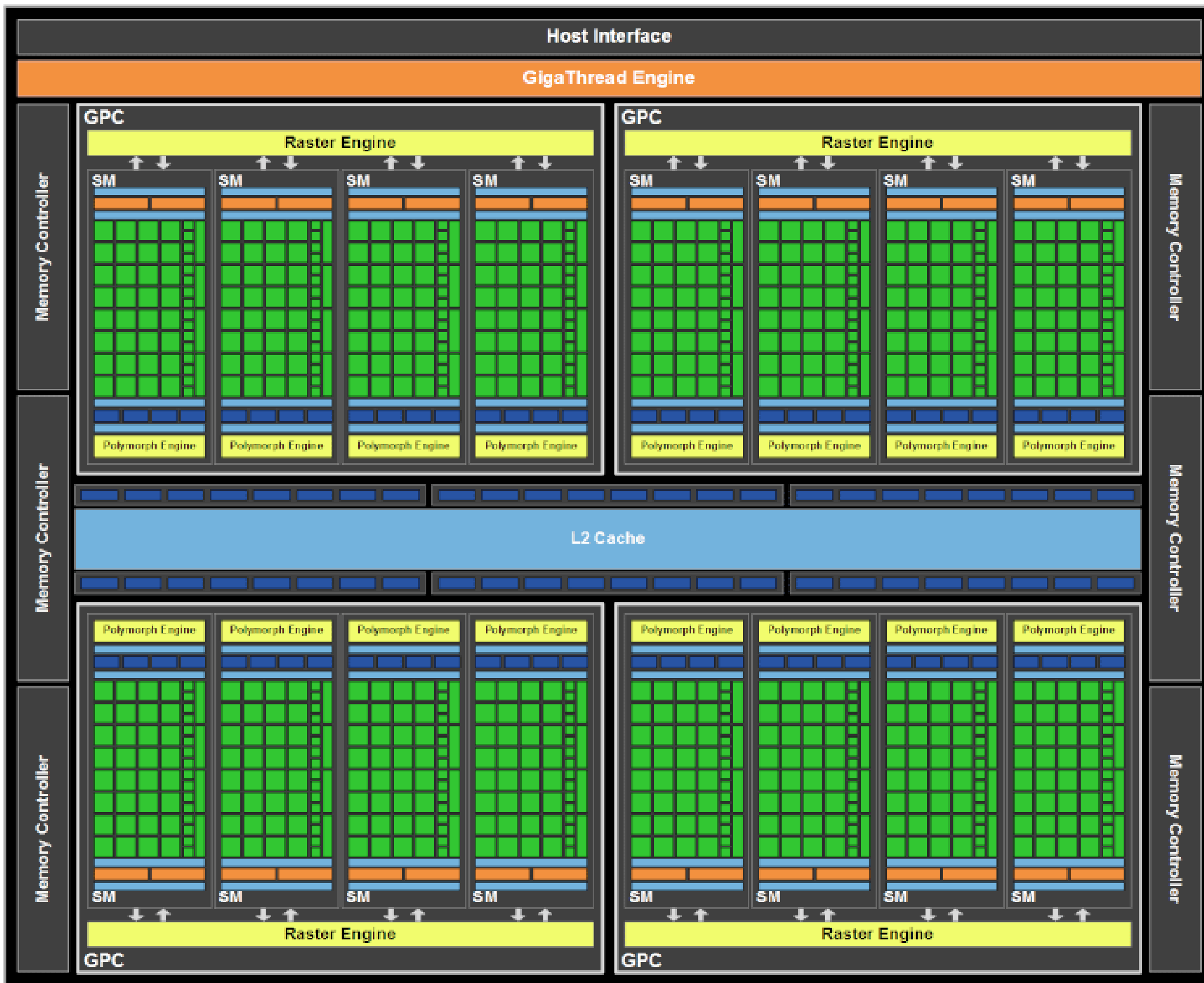
Proizvodnja silicijevih rezin je namreč enaka za vse modele, proizvajajo se le čipi najvišjega razreda. Te čipe nato strojno oz. programsko deklasirajo in spravijo v promet pod drugim tržnim imenom, s primerno nižjo ceno. V začetkih so ta problem reševali programsko, kaj kmalu pa so se pojavila nizko-nivojska programska orodja (npr. *Riva Tuner*), ki so to zaščito obšla, uporabnik pa je s tem pridobil zmogljivosti karte višjega razreda (npr. NVidia GeForce 6800LE in ATI Radeon X800GTO[2]).

4.2 GF100 (GeForce GTX 480)

Grafični čip GF100 (Slika 7) je razdeljen na štiri procesorjske gruče (ang. *Graphics Processor Cluster*, GPC in ne več TPC kot pri GT200), sestavljene iz štirih SMjev in ROP enote. Vsak SM ima 32 SPjev, štiri enote za posebne matematične funkcije (ang. *Special Function Unit,* SFU) in en PolyMorph Engine[8]. Ta predstavlja novo procesno enoto, ki upravlja z oglišči trikotnikov in obdelavo le teh.

Združuje pet osnovnih funkcijskih enot:

- Vertex Fetch,
- Tessellator,
- Viewport Transport,
- Attribute Setup in
- Stream Output.



Slika 7: Arhitektura grafičnega čipa GF100.
(vir: www.anandtech.com)

Zeleni kvadratki predstavljajo SPje, osnovo paralelnega računanja. Teh je skupaj 512, kar je 113% več kot pri čipu GT200. Trenutni izvedenki čipa GF100, modela GTX 470 in GTX 480, s strani števila SPjev nista popolna čipa,. Šibkejši GTX 470 ima lasersko odstranjena dva, medtem ko ima močnejši GTX 480 odstranjen en SM. Po prvih testih[42] je model GTX 480 v povprečju za nekaj odstotkov hitrejši od dvoprocesorske predhodnice GTX 295, ki ima 480 SPjev in manj efektivno arhitekturo.

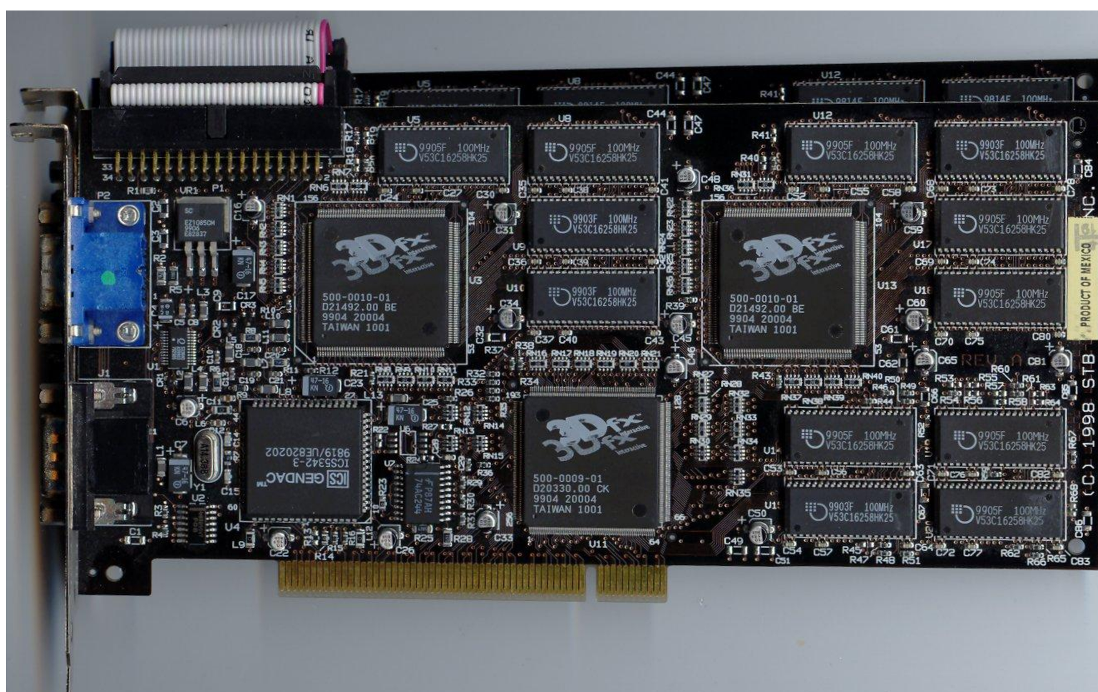
Sivi pravokotniki ob strani so pomnilniški krmilniki. Vsak izmed njih je 64 biten, kar skupaj predstavlja 384 bitno pomnilniško vodilo. Če podrobneje pogledamo GT200b, opazimo osem 64 bitnih pomnilniških krmilnikov, torej skupno vodilo širine 512 bitov. Kje je torej napredek? S stališča širine pomnilniškega vodila ga mogoče res ni, doprinese pa ga sam pomnilnik, ki je tipa GDDR5. Ta se v primerjavi s predhodnikom, GDDR3, razlikuje po višji prepustnosti. V eni urini periodi pošlje namreč štiri podatke[9], kar je v primerjavi z GDDR3, enkrat več. To mu uspe s pomočjo dodatne ure, saj je ena ura namenjena branju, druga pa pisanju. Tako kot GDDR3 (in vse prejšnje generacije *Double Data Rate* pomnilnika), se poslužuje trika za pošiljanje podatkov tako na vrhu urinega signala, kot tudi na dnu, kar je poleg višjih delovnih frekvenc tudi glavna razlika v primerjavi z SDRAM pomnilnikom. Tako pomnilniška prepustnost čipa GT200b ob nazivni frekvenci pomnilnika (1242 MHz) znaša teoretičnih 159 GB/s. GF100 pa, z modelom GTX 480 na čelu, pri 384 bitnem pomnilniškem vodilu in s frekvenco pomnilnika 924 MHz premeče do 177 GB podatkov v eni sekundi. Poleg tega je na GF100 prvič uporabljen ECC (ang. *Error Correction Code*) pomnilnik.

4.3 Tehnologija SLI

4.3.1 3DFX SLI

SLI ni novost na področju večprocesorskih grafičnih rešitev, saj obstaja že vse od leta 1998[10]. Takrat je pionir grafičnega pospeševanja, 3DFX (ustanovljen leta 1994), izdelal svoj drugi grafični pospeševalnik (ang. *graphics accelerator*) z imenom Voodoo2. Ta je nasledil uspešnega Voodoo, ki je dve leti brez prave konkurence vladal 3D grafiki v računalniških igrah. S svojim lastnim API (ang. *Application Programming Interface*) vmesnikom, imenovanim Glide, so več kot konkurirali takrat še nedodelanim in slabo podprtim vmesnikom Direct3D, OpenGL ter QuickDraw 3D. Posebnost Voodoo grafičnih pospeševalnikov je bila v tem, da to niso bile samostojne grafične karte. Za delovanje so potrebovali sistemsko grafično karto, sposobno 2D izrisovanja. Za primerjavo z današnjimi grafičnimi kartami, je Voodoo čip deloval pri pičlih 50 MHz, medtem ko je za shranjevanje tekstur uporabljal 4MB EDO RAM pomnilnika s frekvenco 50 MHz. Voodoo2 je deloval pri 90 MHz, za izris pa je uporabljal 8 oz. 12 MB EDO RAM pomnilnika s frekvenco 100 MHz. Surova moč obdelave je znašala takrat zavidljivih 90 milijonov pikslov na sekundo. Največji minus Voodoo2 s stališča uporabnikov je bila potreba po primarni grafični karti, poleg tega pa je bila najvišja podprta ločljivost 800x600 pik, pri 16 bitni barvni globini. Takratni največji konkurenti, ATI Rage Pro ter NVidia Riva 128, so bile t.i. *single-chip* izvedbe, ki prav tako niso imele težav s prikazom 32 bitne barvne globine. Na žalost pa so bile prepočasne in neoptimizirane, vse to pa je nekaj mesecev kasneje uspelo nasledniku modela Riva 128, Riva TNT[11].

Voodoo2 pa je bil tako prvi izdelek na trgu grafičnih pospeševalnikov, ki je podpiral vzporedno vezavo še enega enakega pospeševalnika. S tem so v teoriji podvojili računsko moč in tako še povečali število izrisanih sličic na sekundo (ang. *Frames Per Second,* FPS). Kratica je v originalu pomenila Scan-Line Interleave[12], princip delovanja pa je bil dokaj enostaven. Posamezna karta je izrisovala polovico horizontalnih linij, skupaj pa sta tvorili celotno sliko (ang. *frame*). Za tak sistem je bila potrebna matična plošča s tremi PCI režami; eno za primarno 2D karto ter dve za obe Voodoo2 karti. Na Sliki 8 je viden sistem vezave dveh Voodoo2 pospeševalnikov preko 34 pinskega kabla. V resnici je šlo za spremenjen kabel disketne enote, pri čemer so bili pini 16, 17, 18 in 19 obrnjeni. Poleg tega je bilo potrebno zunaj ohišja grafični karti povezati še preko D-Sub (VGA) kabla. S tem je bila možna največja ločljivost 1024x768 pik, kar je bilo za takratne CRT (ang. *Cathode Ray Tube*) monitorje bolj izjema kot pravilo.



Slika 8: Voodoo2 pospeševalnika v SLI načinu.
(vir: en.wikipedia.org)

4.3.2 NVidia SLI

Konec leta 2000 je NVidia kupila podjetje 3DFX in s tem tudi vso njihovo tehnologijo. Njihove produkte so opustili, zadnji prodajni model je bil dvoprocesorski Voodoo5 5500, medtem ko razvojni model, štiriprocesorski Voodoo5 6000, ni ugledal prodajnih polic. Prav tako so opustili razvoj gonilnikov, dotedanjim kupcem pa so za določen čas ponudili zamenjavo grafične karte za njihovo lastno[13].

Kratico SLI so v znak predelav ter izboljšav preimenovali v Scalable Link Interface[14], in jo leta 2004 uradno predstavili z novo serijo grafičnih procesorjev, imenovanih NV40, skupaj z novim standardnim vodilom PCI Express 1.0 (PCI-E). Na tržišču so bili prepoznani pod imenom GeForce 6, z najmočnejšo GeForce 6800 Ultra na čelu. Takratnih 150 tisočakov vredna karta je predstavljala višek razvoja grafičnih procesorjev, z možnostjo povezovanja dveh identičnih kart pa je ta sloves le še potrdila. Najboljši nakup je tedaj predstavljal model 6800LE, ki se mu je s pomočjo naprednih nizko-nivojskih programskih orodij dalo "odkleniti" tovarniško zaklenjene cevovode. Tako je karta, v osnovi vredna tretjino cene modela Ultra, postala prodajni hit za množico entuziastov (med katere se šteje tudi avtor teksta). Vendar pa vsi čipi niso bili sposobni delovati pri vseh odklenjenih cevovodih, kar se je poznalo na popačeni sliki (t.i. grafični artefakti)[15], zato je sreča igrala veliko vlogo pri samem nakupu. Slika 9 predstavlja trenutno najzmogljivejšo kombinacijo, QuadSLI dveh dvoprocesorskih GTX295.

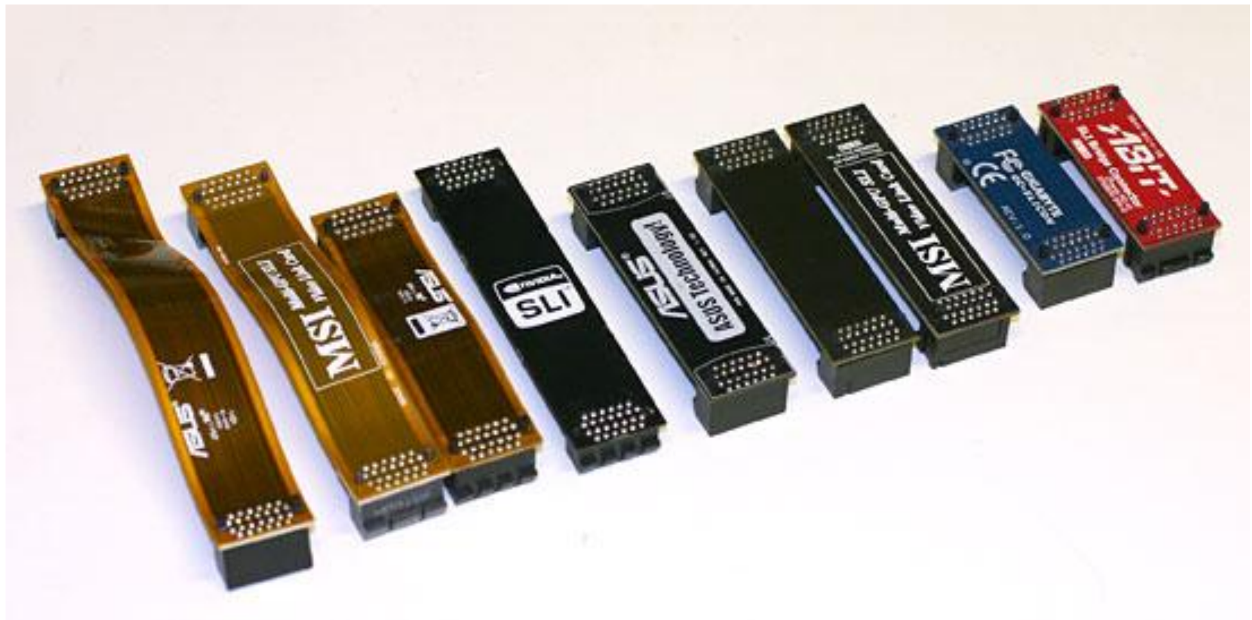
Slika 9: QuadSLI. Za debele denarnice.
(vir: www.hstorm.net)

Tehnologija SLI omogoča delitev procesiranja 3D slike med več kot eno grafično karto. Največ primerov uporabe SLI je v dveh identičnih kartah z enakih čipom, medtem ko obstajajo še druge možnosti uporabe, ki bodo v nadaljevanju obrazložene.

Za pravilno delovanje sta potrebni (vsaj) dve karti z enakim čipom, ni pa nujno da sta od istega proizvajalca. Za najbolj zanesljivo delovanje NVidia priporoča uporabo identičnih kart z enako različico BIOSa, enako količino pomnilnika, ter enakimi delovnimi frekvencami tako jedra kot pomnilnika. V primeru kart z različno količino pomnilnika oz. drugačnimi frekvencami ure, se boljša izmed kart prilagodi slabši. Poleg omenjenih grafičnih kart mora biti matična plošča grajena okrog veznega nabora nForce in vsebovati vsaj dve prosti reži PCI-E 16x. Ker pa je bila ta poteza NVidie zelo monopolna, so kasneje[16] dovolili uporabo tehnologije SLI tudi na Intelovih veznih naborih višjega razreda. Sem spadajo X38, X48 ter sedanji X58 (Core i7, podnožje 1366). X38 uradno nikoli ni bil podprt, se je pa z določenimi programskimi triki ter modificiranimi gonilniki vseeno dalo vzpostaviti delujoč SLI. Za delovanje bi bil sicer na matični plošči potreben dodaten čip, imenovan NF200, vendar pa se je kmalu izkazalo, da je bila to le t.i. licenčnina za uporabo SLI. Vse ukaze čipa NF200 so namreč podpirali že vsi trije prej omenjeni nabori podjetja Intel.

SLI deluje po principu gospodar-suženj (ang. *Master-Slave*), saj ena od kart prevzame vlogo gospodarja, druga pa vlogo sužnja. Glede na uporabljeno metodo procesiranja (SFR, AFR ali SLI AA, razložene v nadaljevanju), dobita obe karti enako količino dela (obdelava pik). Gospodar dobi celotno delo, ki ga razdeli med sužnje (v primeru 3-Way SLI ali QuadSLI). Za to poskrbi poseben SLI mostiček (ang. *SLI Bridge*, Slika 10), ki delo pošlje naprej sužnju v obdelavo. Ko suženj opravi delo, ga pošlje nazaj gospodarju, ta sestavi končno sliko ter jo pošlje na zaslon.

V primerih, ko imamo v SLI vezani karti, ki sta po zmogljivostih v nižjem razredu, je možno delitev in pošiljanje dela opraviti preko veznega nabora matične plošče. To je možno tudi v primeru zmogljivejših kart, vendar pa so zmogljivosti zaradi ozke pasovne širine veznega nabora neprimerno slabše kot ob uporabi SLI mostička.



Slika 10: Od desne proti levi: kronološki pregled nad SLI mostički.
(vir: www.xtremesystems.org)

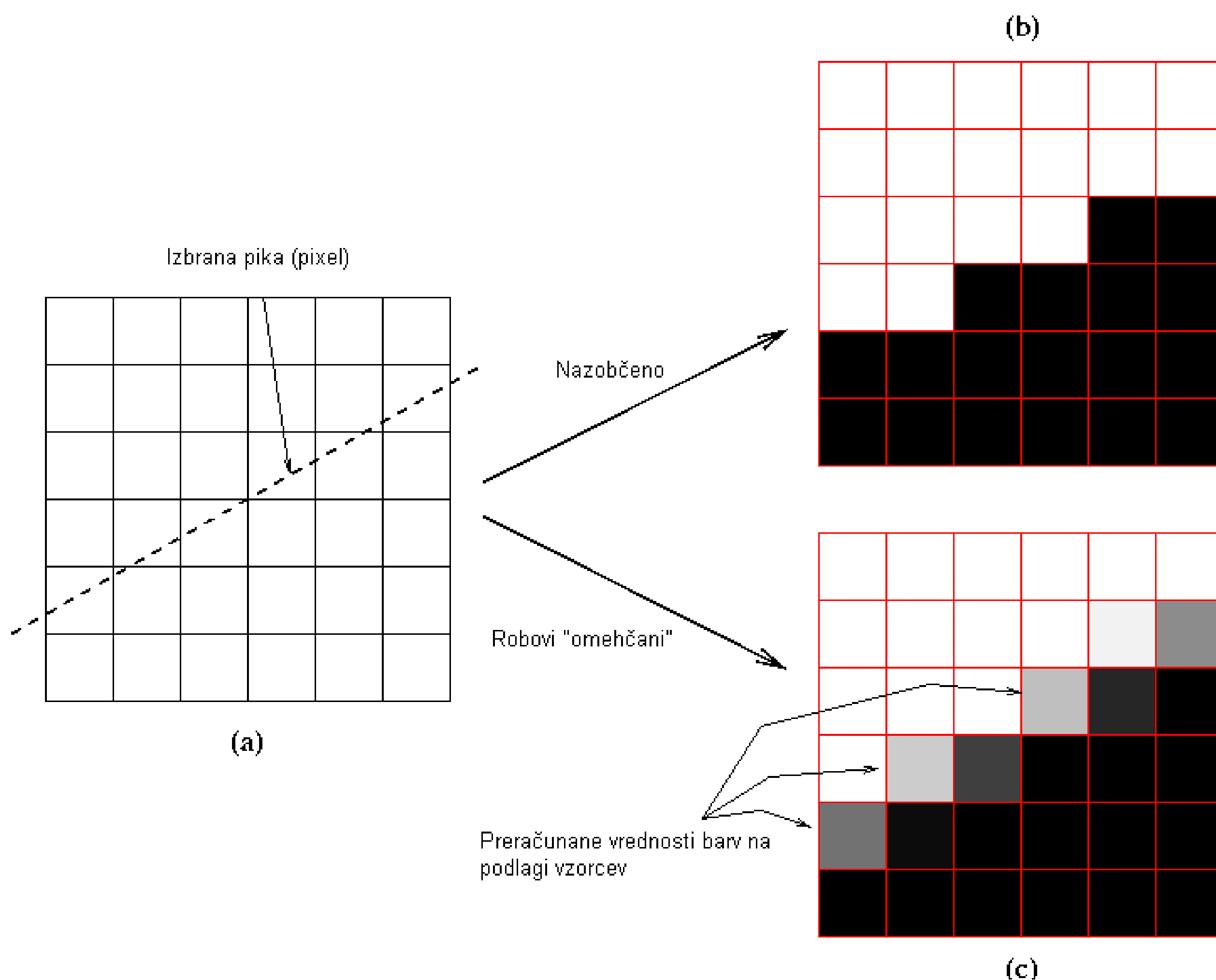
SLI ima na voljo tri metode delovanja[17]:

- Split Frame Rendering (SFR)
- Alternate Frame Rendering (AFR) in
- SLI Anti-Aliasing (SLI AA)

Split Frame Rendering razdeli trenutno sliko po horizontalni liniji med GPE, ki so prisotne v SLI konfiguraciji. Nato jo analizira, kar mu kasneje pomaga pri delitvi obdelave. Na podlagi analize poskuša delo razdeliti čimbolj enakovredno (50/50 v primeru dveh GPE), kar pa ne pomeni delitve slike na zgornjo in spodnjo polovico. Z naprednimi algoritmi ugotovi časovno zahtevnost posameznega dela slike v primerjavi z preostalim delom slike. Primer slike, ki prikazuje nebo na zgornjih dveh tretjinah in podrobne objekte na spodnji tretjini, razdeli delo na približno 25/75. Ta odstotek varira od stopnje podrobnosti, uporabljenih tehnik mehčanja robov, načina filtriranja tekstur in dodatnih učinkov.

Alternate Frame Rendering je preprostejša metoda delitve dela, in spominja na 3DFX-ov način. Vsaka GPE obdeluje sliko vzporedno, s tem da so prvi GPE namenjene lihe, drugi pa sode horizontalne linije. Ko suženj opravi z delom, to sporoči gospodarju in začne s prenosom svojega deleža preko SLI mostička. Gospodar nato sestavi sliko in jo pošlje na privzeti izhod. S takim načinom delitve dela teoretično skrajšamo obdelavo posamezne slike za 50%, v praksi pa se zaradi mnogih zakasnitev čas obdelave zmanjša v najboljšem primeru na 45%.

Tretja metoda, *SLI Anti-Aliasing*, predstavlja tehniko mehčanja robov (Slika 11), in bo podrobno razložena v naslednjem poglavju. Poudarek je na kvaliteti slike in ne na zmogljivosti obdelave, kot v prejšnjih dveh primerih. Mehčanje robov se v tem primeru razdeli med obe GPE, kar teoretično podvoji kvaliteto trenutno obdelane slike. Mehčanje robov poteka na principu vzorčnega odmika, kjer gre ena GPE v ekstremno pozicijo pike (npr. zgoraj levo), medtem ko gre druga GPE v nasprotni ekstrem, v tem primeru spodaj desno. Rezultanto predstavlja povprečje obeh odmikov. S to metodo se lahko uporabnik poslužuje naprednih načinov mehčanja robov; SLI 8X, SLI 16X, SLI 32X ter SLI 64X.



Slika 11: Poenostavljeno prikazana tehnika mehčanja robov.

Poznanih je več vrst vezave grafičnih kart v SLI. Najbolj uporabljena je klasična vezava dveh kart z eno GPE, obstajajo pa še druge možnosti. SLI na eni sami karti je naslednja najbolj pogosta uporaba te tehnologije. Februarja 2005 je Gigabyte izdelal GV-3D1 (Slika 12), prvo grafično karto z dvema 6600GT GPE na enem tiskanem vezju. S strani NVidie je bilo to nepodprto, prav tako je bila potrebna posebna matična plošča (v kompletu s karto) ter prilagojeni gonilniki. Poleg tega je bil komplet bistveno dražji od dveh posamičnih kart 6600GT ter navadne SLI kompatibilne matične plošče. Kasneje je Gigabyte izdal močnejšo različico z dvemi 6800GT GPE, ki pa zaradi podobne zasnove prav tako ni bila uspešna.

Marca 2006 je s podobno idejo prišel Asus, ko je v prodajo poslal naslednico serije GeForce 6, model N7800GT Dual. Karta z dvema high-end GPE na eni tiskanini je takrat bila paša za oči na papirju, medtem ko se je v praksi izkazala le malo bolje od Gigabyte-ove pionirke. Delala je izključno na nForce4 matičnih ploščah, a ne optimalno. Zopet je bila problem visoka cena v primerjavi z dvema posamičnima 7800GT v SLI.

Slika 12: Gigabyte GV-3D1, prvi SLI sistem na eni tiskanini.
Veliko denarja in veliko težav.
(vir: www.hardwarezone.com)

Januarja 2006 je NVidia izdelala svojo prvo uradno dvoprocesorsko grafično karto z imenom 7900 GX2. Temeljila je na dveh 7900GTX GPE z nižjimi frekvencami, vsaka z 512MB GDDR3 pomnilnika. Za razliko od Gigabyte-a ter Asus-a je bila ta karta sestavljena iz dveh tiskanin, z ločenim hladilnim sistemom. Produkt končnim kupcem ni bil na voljo, temveč le OEM podjetjem za distribucijo QuadSLI sistemov. Prvi tak sistem je bil Dell XPS, rezultat odkupljenega podjetja Alienware. Ker so bile vse omenjene karte omejene izdaje, je NVidia izdelala izboljšano verzijo 7950 GX2, ki je bila na voljo širši populaciji. Odstranili so vse probleme, odkrite na 7900 GX2, in karta je tako doživela dokaj pozitiven sprejem med računalniškimi zanesenjaki. Prav tako so jo skrajšali (Slika 13) iz 31cm na dobrih 22cm ter tako olajšali vgradnjo v standardna ATX ohišja.

Slika 13: Primerjava med prvo uradno dvoprocesorsko karto 7900GX2 (zgoraj) in njeno naslednico 7950GX2 (spodaj).

(vir: www.hardforum.com)

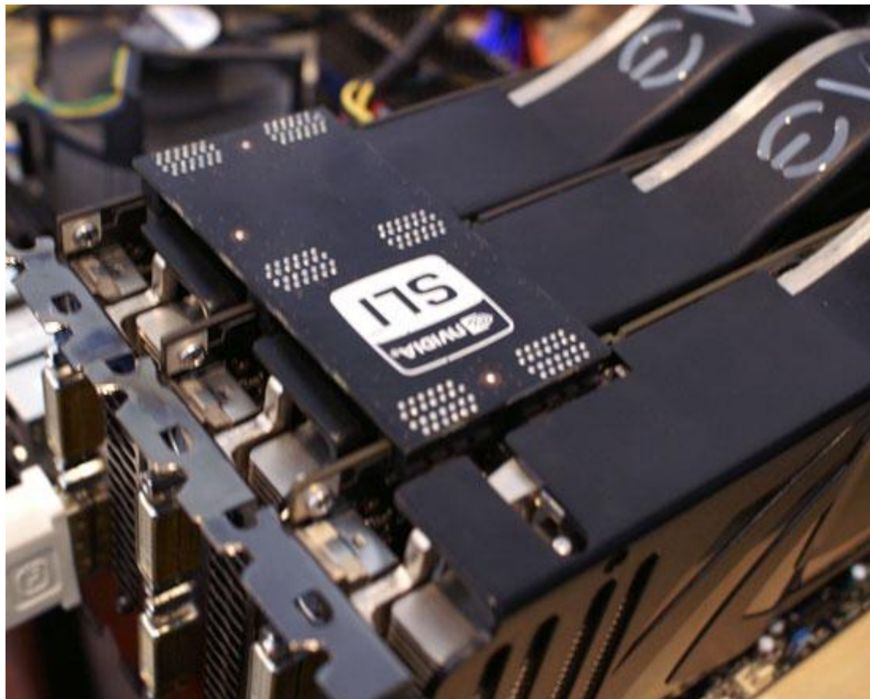
Dve leti kasneje, marca 2008, je NVidia na trg poslala dvoprocesorsko naslednico z imenom 9800 GX2[18]. Podobno kot predhodnica je bila sestavljena iz dveh tiskanih, s to razliko, da sta bili obrnjeni ena proti drugi, z vmesnim hladilnim telesom. Bila je prva dvoprocesorska grafična karta, ki je bila po mnenju avtorja tega teksta vredna nakupa, mogoče manj iz finančnega, pa toliko bolj iz tehničnega vidika.

Kasneje so serijo 9 začeli izdelovati še v 55nm proizvodnem procesu, z modeli GeForce 9800GT ter GTX+. Dvoprocesorske GX2 v 55nm pa niso izdali, ker sta bila naslednika s primerljivimi zmogljivostmi, modela GTX 260 in GTX 280, že v prodaji.

Kasneje je NVidia izdala posodobljene različice čipa GT200, z oznako GT200b (55nm), prisotne na grafičnih kartah GTX 260 216SP, GTX 275, GTX 285 ter GTX 295. Slednja predstavlja dvoprocesorsko različico modela GTX 275, najprej z dvema tiskaninama (*DualPCB*) in kasneje z eno samo (*SinglePCB*).

Če osnovni vezavi dveh GPE dodamo še tretjo, dobimo 3-Way SLI. Pojavil se je s serijo G80 grafičnih procesorjev, z najzmogljivejšema 8800 GTX in 8800 Ultra. Na začetku je bilo tak sistem možno zasnovati le na matičnih ploščah z veznim naborom NVidia nForce serije 6 in 7 (modeli 680i, 780i ter 790i), kasneje pa tudi na X48 ter X58 naborih podjetja Intel. Matična plošča mora imeti 3 polnokrvne (x16) PCI-Express reže, kar pa je bilo na začetku bolj izjema kot pravilo. Danes skorajda ni več matičnih plošč višjega cenovnega razreda brez treh PCI-E x16 rež.

Za pravilno delovanje 3-Way SLI sistema je potreben SLI mostiček s tremi konektorji (Slika 14), podprt pa je le v okolju Windows Vista ter Windows 7. Razlog za to je DirectX 9 API vmesnik, ki v operacijskem sistemu Windows XP dovoli obdelavo le treh slik vnaprej. Windows Vista (DirectX 10) ter na njej osnovani Windows 7 (DirectX 11) s število slik nista omejena. Ker tako 3-Way SLI kot QuadSLI za delitev dela uporabljata prej opisani AFR, je v DirectX 9 API vmesniku optimalna realizacija nemogoča.



Slika 14: 3-Way SLI v praksi
(vir: www.tomshardware.com)

QuadSLI predstavlja vezavo štirih GPE, in je v teoriji najzmogljivejša kombinacija. V praksi je izvedljiva izključno preko dveh dvoprocesorskih grafičnih kart, in tako kot 3-Way SLI prinaša povečanje zmogljivosti le v operacijskem sistemu Windows Vista in Windows 7. QuadSLI prav tako ne prinaša občutnih izboljšav na standardnih ločljivostih. Dandanes se uporablja le v primeru prikazovanja slike na več ekranih ter v načinu SLI-AA, ki občutno polepša prikazano sliko s pomočjo visoko-ločljivostnega mehčanja robov, stopnje 32 oz. 64. Za surovo moč QuadSLI sistema bi v avto-moto žargonu dejali, da *predstavlja konje, ki jih je zelo težko spraviti na cesto*.

Naslednja možna kombinacija SLI sistema, Hybrid SLI, pa je ekološko obarvana. Izdelana je bila januarja 2008, ideja pa je bila povzeta po ATI-jevi rešitvi, imenovani PoweXpress. Osnovo predstavlja matična plošča z integriranim grafičnim procesorjem (ang. *Integrated Graphics Processor,* IGP) ter z nForce veznim naborom serije 7. V povezavi z NVidia grafičnimi kartami osme generacije imamo sistem Hybrid SLI, ki lahko deluje v t.i. ekonomičnem (HybridPower) oziroma v zmogljivostnem načinu (GeForce Boost). Na začetku je bil sistem Hybrid SLI namenjen le za prenosnim računalnikom, kasneje pa so ga razširili tudi med namizne računalnike. Princip delovanja je urejen s strani veznega nabora ter gonilnikov, ki s pomočjo realno-časovne analize uporabe sistemskih virov določijo, kdaj je čas za vklop dodatne GPE oz. kdaj je za obdelavo podatkov dovolj IGP na matični plošči.

Quadro Plex (Slika 15) je zadnja izmed možnosti vzopredne vezave več GPE. Sodi med sisteme za vizualno računanje (ang. *Visual Computing Systems,* VCS), namenjen pa je profesionalni rabi[19]. Izdelan je kot zunanja komponenta, ki je preko posebnega 68-pinskega konektorja VHDCI (ang. *Very High Density Cable Interconnect*) povezana z namensko PCI Express kartico v delovni postaji. V ohišju naprave sta dve oz. štiri GPE, medsebojno povezani preko SLI vodila. Modeli se razlikujejo med seboj glede na število GPE ter količino delovnega pomnilnika.

Slika 15: Quadro Plex, za profesionalno rabo.
(vir: www.f1cd.ru)

4.4 Mehčanje robov

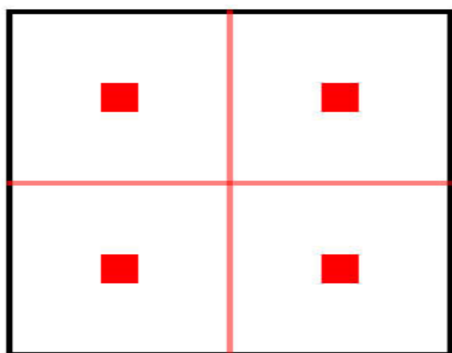
Mehčanje robov (ang. *Anti-Aliasing,* AA) je posebna tehnika v računalniški grafiki, s katero odstranimo nazobčene robove objektov in jih zglajene prikažemo bolj realistično. Nazobčeni oz. narezani robovi so rezultat računalniške grafike, ki v primerjavi z realnim svetom, kjer so linije in vijuge pravih oblik in prehodov (človeško oko je "analogno"), sliko sestavi iz diskretnih točk svetlobe, imenovane pika (ang. pixel). Pike so glede na ločljivost ekrana vedno enake velikosti ter enotnih barv, kar pripelje do ostrih, nerealističnih robov grafičnih objektov.

Obstaja več tehnik mehčanja robov, vsem pa je skupno, da rezultat dosegajo s podvojevanjem ločljivosti (2X, 4X, 8X, ter 16X). Vsaka novejša GPE lahko obdela piko z vsaj 4 vzorci, kako hitro bo obdelava končana, pa je odvisno od samih zmogljivosti GPE. GF100 tako kot najnovejši ATI RV870 podpira 32-kratno mehčanje robov. Najbolj poznane tehnike so Supersampling (FSAA, Fullscreen Anti-Aliasing), Multisample Anti-Aliasing (MSAA) ter Coverage Sampled Anti-Aliasing (CSAA).

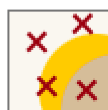
Obstaja še veliko izpeljank, katerih osnova so zgoraj omenjeni. Tehnike mehčanja robov si bomo podrobneje ogledali zato, ker predstavljajo procesorsko najbolj zahtevne obdelave podatkov na grafičnih procesorjih.

4.4.1 Fullscreen Anti-Aliasing

FSAA je požrešna vrsta mehčanja robov, ki se v novejših 3D aplikacijah ne uporablja več. Vsako piko analizira z vnaprej določenim številom vzorcev (ang. *sample*). Kolikšno je to število, je odvisno od izbire uporabnika, omejeno pa je s pasovno širino ter količino pomnilnika GPE. V ozadju algoritma se tako odvija navidezna povečava ločljivosti za faktor vzorcev. V primeru ločljivosti slike 800x600 in izbranem 4-kratnem supersampling-om algoritem dejansko obdela sliko ločljivosti 1600x1200 (Slika 16). Nato jo pomanjša na začetno ločljivost (ang. downsampling) z novo vrednostjo RGB palete, ki je izračunana s povprečjem vseh vzorcev[20], kot prikazuje Slika 17.

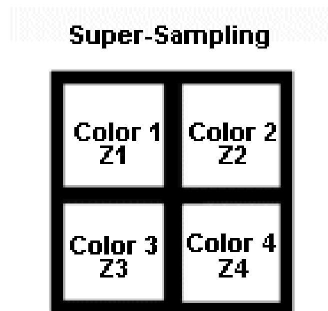


Slika 16: štirikratni SuperSampling.
(vir: en.wikipedia.org)



Slika 17: Računanje vrednosti barve.
(vir: en.wikipedia.org)

Pri FSAA za vsako izhodno piko pri 4-kratnem AA potrebujemo 4-vhodne pike, njihove barve in vrednost koordinate Z (ang. *Z value*, globina), kar je izredno zamudno. Problem bi lahko rešili z ločenim cevovodom za vsakega izmed vzorcev, kar pa je iz arhitekturnega stališča potratno.

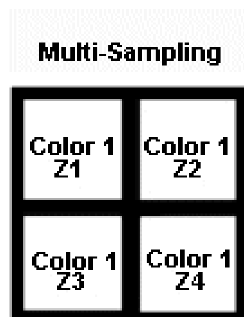


Slika 18: FSAA je s stališča procesorske moči potraten.
(vir: www.firingsquad.com)

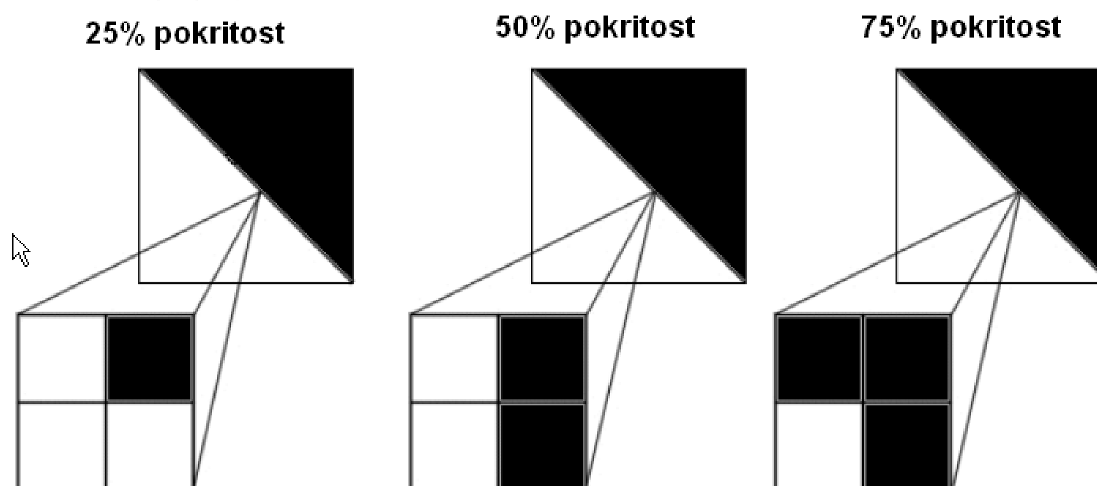
Izboljšana različica zgornjega algoritma je t.i. *adaptive supersampling*, ki obdelavo prilagodi na mejne vzorce povečane pike. Algoritem najprej vzame nekaj vzorcev in jih v primeru, da so si podobni, vzame za računanje nove barve. V nasprotnem primeru doda nove vzorce. S tem pridobimo na hitrosti izvajanja algoritma.

4.4.2 Multisample Anti-Aliasing

Naslednja najbolj znana oblika mehčanja robov na današnjih GPE je Multisample Anti-Aliasing. Osnovan je na FSAA tehniki, z določenimi optimizacijami. MSAA izvede FSAA le na robovih poligonov (trikotnikov, ki zlepljeni skupaj tvorijo 3D objekt), medtem ko se notranjosti ne dotika, zato pride do drastične razlike v hitrosti obdelave posamezne pike. Medtem ko sta oba (FSAA in MSAA) pomnilniško potratna, s tem ko v ozadju povečujeta ločljivost pike, pa MSAA pridobi na času, ker ne računa barve končne pike (Slika 19), ter pri dejanski povečavi robnih vzorcev[21].



Slika 19: MSAA gleda le na Z vrednost.
(vir: www.firingsquad.com)



Slika 20: MSAA za razliko od FSAA obdela le pike, ki so potrebne mehčanja.

Pokritost (ang. coverage) je le redko sto-odstotna, kar lahko pomeni tudi enkrat hitrejše mehčanje robov v enakem časovnem intervalu. Slika 20 predstavlja grafično upodobitev pokritosti pri MSAA tehniki. V prvem primeru en vzorec pike predstavlja rob poligona, s čimer bi MSAA opravil v enem koraku, medtem ko bi FSAA potreboval še dodatne tri (za ne-robne vzorce). Tako MSAA kot tudi FSAA pa se z mehčanjem robov slabo izkažeta pri prosojnih teksturah (ang. *transparent textures*).

4.4.3 Coverage Sampled Anti-Aliasing

Če vzamemo MSAA in ga še malo izboljšamo, dobimo NVidiin Coverage Sampled Anti-Aliasing (CSAA). Za računanje končne barve pike uporablja vrednost koordinate Z, začetno barvo ter informacijo o pokritosti[22]. Poimenovanja posameznih stopenj mehčanja robov pa ne predstavljajo dejanskih stopenj, kar je razvidno v Tabeli 2.

CSAA način	Število vzorcev	Število pokrivnih vzorcev
8x	4	8
8xQ (Quality)	8	8
16x	4	16
16xQ (Quality)	8	16

Tabela 2: Faktor CSAA načina lahko uporabnika zavede.

Uveden je bil tudi nov način, *Quality mode*, ki je primerljiv s dosedanjim MSAA. Način CSAA 8x je le preimenovan MSAA 4x, kar pa končnega uporabnika samo zmede. Na Sliki 21 je prikazana primerjava med navadnim 16x CSAA ter izboljšanim 16xQ načinom. Ker je osnova CSAA klasični MSAA, so tudi zmogljivosti obdelave zelo podobne. Novost so pokrivni vzorci (ang. *coverage samples*), ki preverjajo, koliko pike je pokrite s poligonom. Dodatni vzorci ne vplivajo na samo zmogljivost algoritma, ker ne berejo po teksturnem medpomnilniku.

Slika 21: "Navadni" način izračuna barvo iz štirih, "kvalitetni" pa iz osmih vzorcev.
(vir: www.nvidia.com)

5. Programski model CUDA

V tem poglavju si bomo ogledali osnove programiranja v CUDA okolju. Programski model se v grobem deli na CUDA funkcije (ang. *kernels*), niti (ang. *threads*), hierarhijo pomnilnika (ang. *memory*), ter delitev kode na CPE (ang. *host*) oz. GPE (ang. *device*)[23].

5.1 CUDA funkcije

CUDA omogoča razvijalcu aplikacije, da preko programskega jezika C / C++ definira lastne funkcije, imenovane *kernel*. Locirane so na začetku programa, pod glavo programa. Klic kernel funkcije predstavlja N-kratno izvajanje kode na N nitih v istem časovnem intervalu. Najnovejši Intelov procesor, Core i7 980X, zmore s svojimi šestimi jedri in vklopljeno Hyper-Threading (HT) tehnologijo izvesti največ 12 niti sočasno, ki pa so lahko bolj kompleksne. Ker pa lahko problem največkrat rešimo že z enostavnimi ukazi in zankami, je potreba po kompleksnih ukazih manj pogosta.

Kernel definiramo s predpono

```
__global__ 'ime_funkcije',
```

pokličemo pa ga z ukazom

```
ime_funkcije <<< M, N >>> (atributi),
```

kjer M in N predstavljata število blokov ter velikost posameznega bloka. Za lažjo predstavo je spodaj podan primer, ki seštevek vektorjev A in B zapiše v vektor C.

```
// definicija kernel funkcije
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}

int main()
{
    ...
    // klic kernel funkcije
    VecAdd<<<1, N>>>(A, B, C);
}
```

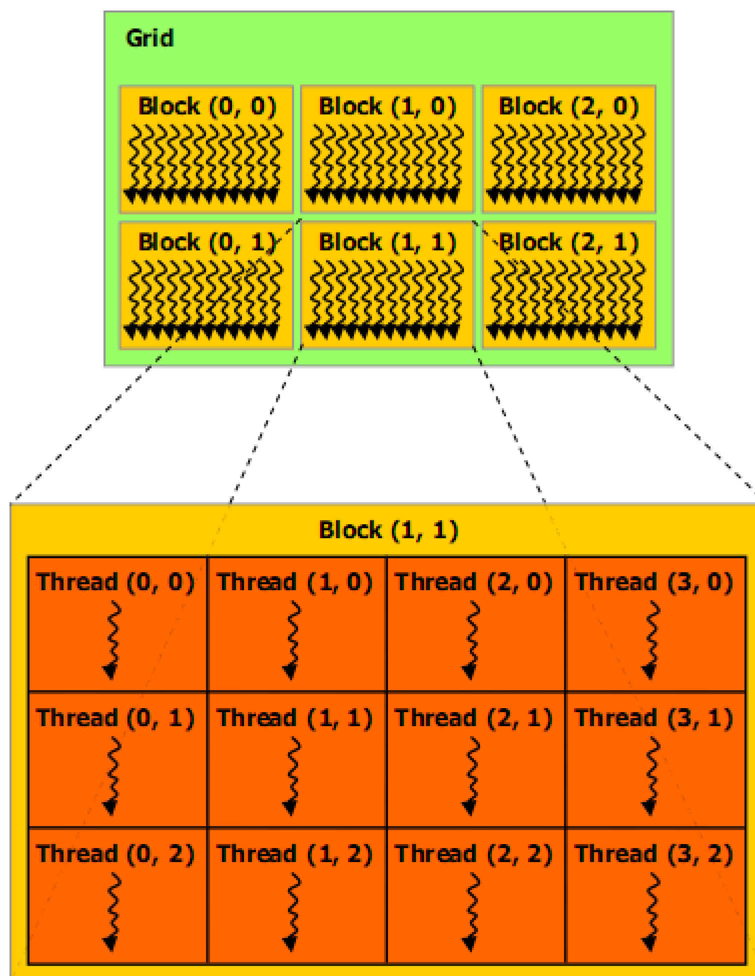
5.2 Hierarhija niti

Spremenljivka `threadIdx` je trikomponentni vektor, ki predstavlja eno-, dvo- ali tri-dimenzionalni indeks trenutno uporabljene niti[24]. S tem indeksom tudi začrtamo (1D/2D/3D) *thread block*, t.i. blok niti. Z njim si razvijalec pomaga pri razvrščevanju podatkov znotraj podatkovnih struktur tipa vektor in matrika. Za primer uporabe dvodimenzionalnega indeksa je spodaj prikazan primer vsote matrik A in B velikosti NxN v matriko C.

```
// definicija kernel funkcije
__global__ void MatAdd(float A[N][N], float B[N][N],
                       float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // klic kernel funkcije
    dim3 dimBlock(N, N);
    MatAdd<<<1, dimBlock>>>(A, B, C);
}
```

Vsaka nit lahko izmenjuje podatke z ostalimi preko skupnega pomnilnika (ang. *shared memory*). Slednji je uporabljen za sinhronizacijo izvajanj niti ter njihovih dostopov do pomnilnika. Sinhronizacija lahko poteka tudi ročno s pomočjo funkcije `__syncthreads()`. Ta deluje kot pregrada, ki pred nadaljnim možnim izvajanjem niti v posameznem bloku najprej počaka, da vse niti končajo s trenutnim izvajanjem. Za učinkovito sinhronizacijo je potreben hiter prvonivojski predpomnilnik (L1 cache) v obliki skupnega pomnilnika, ob pogoju da so vse niti iz istega jedra (SPja). Koliko niti je lahko hkrati v enem samem bloku pa je omejeno s količino skupnega pomnilnika, rezerviranega za posamezen SP. Na današnjih GPE (z izjemo GF100, 768) je možno znotraj posameznega bloka izvajati 512 niti. Dostop do pomnilnika je podrobneje opisan v poglavju 5.3.

Kernel lahko kličemo N-krat, kar pomeni N enakih blokov. S tem je skupna količina niti, ki so na voljo procesiranju za določeno funkcijo (kernel), enaka zmnožku med številom niti v bloku ter številom blokov (Slika 22).



Slika 22: Organiziranost niti znotraj blokov mreže.
(vir: CUDA Programming Guide 2.3)

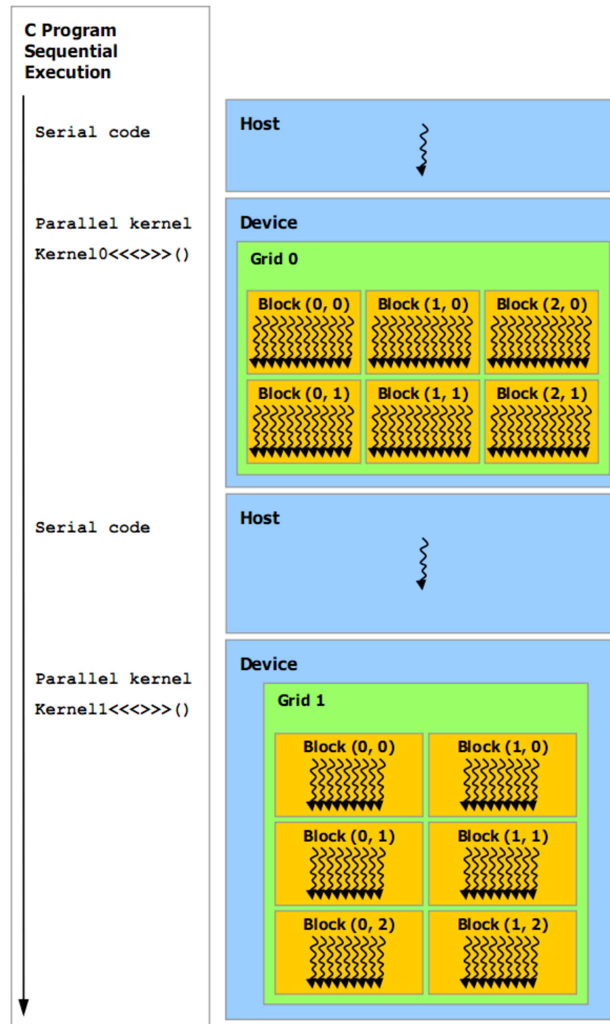
Bloki so organizirani v mrežo blokov (ang. *grid*), ki je lahko eno- ali dvo-dimenzionalna. Velikost (dimenzija) mreže se poda kot prvi parameter pri klicu funkcije. Če vzamemo zgornji primer vsote dveh matrik, s parametrom '1' kličemo en blok, ki bo operiral z `dimBlock` številom niti.

```
MatAdd<<<1, dimBlock>>>(A, B, C);
```

Vsak blok je lahko identificiran preko enoličnega `blockIdx` indeksa, ki deluje na enak princip kot že prej omenjeni `threadIdx`.

5.3 Hierarhija pomnilnika

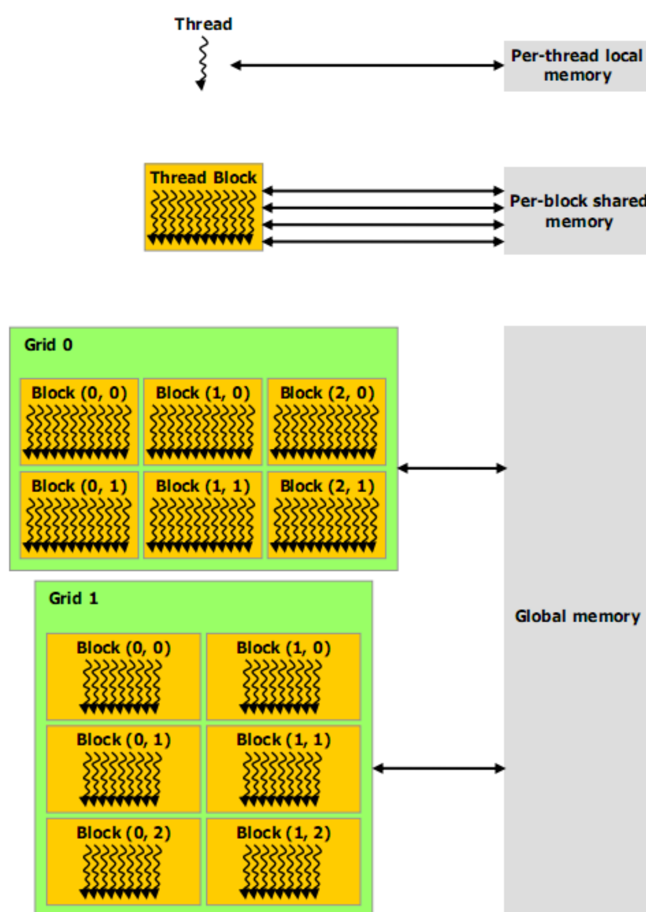
Programski model okolja CUDA skrbi za to, da se niti poganjajo preko CUDA GPE, ki deluje kot nekakšen koprocesor centralni procesni enoti, medtem ko se vsa ostala programska koda še naprej izvaja preko CPE (Slika 23). Skrbi tudi za to, da obe procesni enoti opravljata z lastnim pomnilniškim prostorom. V osnovi delimo ta prostor na pomnilnik CPE (ang. *host memory*) in na pomnilnik GPE (ang. *device memory*). Ker ima GPE več različnih pomnilniških prostorov, se je potrebno v kodi sklicevati na pravega, potrebna je vnaprejšnja rezervacija, kot tudi kasnejša sprostitev pomnilnika. Ročno poteka tudi prenašanje podatkov iz CPE v GPE in obratno.



Slika 23: Učinkovita izraba množice niti v primerjavi s CPE.
(vir: CUDA Programming Guide 2.3)

Niti lahko dostopajo do podatkov preko različnih pomnilniških prostorov (Slika 24). Vsaka posamezna nit ima med potekom izvajanja poleg hitrih registrov na voljo tudi lasten lokalni pomnilnik. Vsak blok niti ima prav tako lasten pomnilnik, ki je v skupni rabi z vsemi nitmi istega bloka. Njegova življenjska doba je enaka življenjski dobi bloka. Največji je globalni pomnilnik, do katerega lahko dostopajo vse niti, ne glede na blok oz. mrežo blokov. Globalni pomnilnik nima možnosti predpomnenja, zato je upravljanje z njim ključnega pomena[25].

Poleg omenjenih sta za vse niti na voljo še dva dodatna pomnilniška prostora, konstantni ter teksturni. Skupaj z globalnim tvorijo namenski pomnilnik za točno določeno vrsto uporabe. Teksturni pomnilnik med drugim omogoča različne načine naslavljanja ter možnostjo filtriranja podatkov. Obstojnost podatkov oz. življenjska doba omenjenih pomnilnikov je določena s časom izvajanja.



Slika 24: Hierarhija pomnilnika.
(vir: CUDA Programming Guide 2.3)

5.3.1 Pomnilnik CUDA naprave

Kot je bilo omenjeno že v prejšnjem poglavju, si programski model okolja CUDA predstavlja celoten sistem kot hibrid med CPE in GPE, vsakega s svojim pomnilniškim prostorom. V tem poglavju se bomo omejili na *Device memory*, ki je programsko lahko dodeljen kot linearni pomnilnik (ang. *linear memory*) oz. v obliki polja (ang. *CUDA arrays*).

Linearni pomnilnik je GPE na voljo kot 32-bitni pomnilniški prostor, kjer so podatki dosegljivi s pomočjo kazalcev. Rezerviranje pomnilnika se izvede preko funkcije `cudaMalloc()`, sprosti pa s `cudaFree()`. Za prenos podatkov med CPE in GPE skrbi `cudaMemcpy()`. Spodnji primer prikazuje uporabo omenjenih ukazov, ki prenese v pomnilnik GPE vsebino vektorjev A in B, ju sešteje, ter vsoto vrne CPE[26].

```
// Kernel funkcija (GPE)
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}

// Glavni program (CPU)
int main()
{
    // Rezerviranje pomnilnika za vektorje v pomnilniku GPE
    size_t size = N * sizeof(float);
    float* d_A;
    cudaMalloc((void**)&d_A, size);
    float* d_B;
    cudaMalloc((void**)&d_B, size);
    float* d_C;
    cudaMalloc((void**)&d_C, size);

    // prenašanje vektorjev iz CPE v GPE
    // h_A, h_B in h_C so ze shranjeni vektorji v pomnilniku CPE
    cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
    cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);

    // klic kernel funkcije
    int threadsPerBlock = 256;
    int blocksPerGrid =
            (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C);

    // prenašanje rezultata nazaj v pomnilnik CPE
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);

    // sproščanje pomnilnika GPE
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
}
```

Pomnilnik GPE je možno rezervirati tudi preko drugih funkcij. Funkcija `cudaMallocPitch()` se uporablja za 2D polja, medtem ko se za 3D polja uporablja funkcija `cudaMalloc3D()`. Sprehod skozi vse elemente 2D polja velikosti `width x height` je mogoč s pomočjo omenjene funkcije `cudaMallocPitch()`.

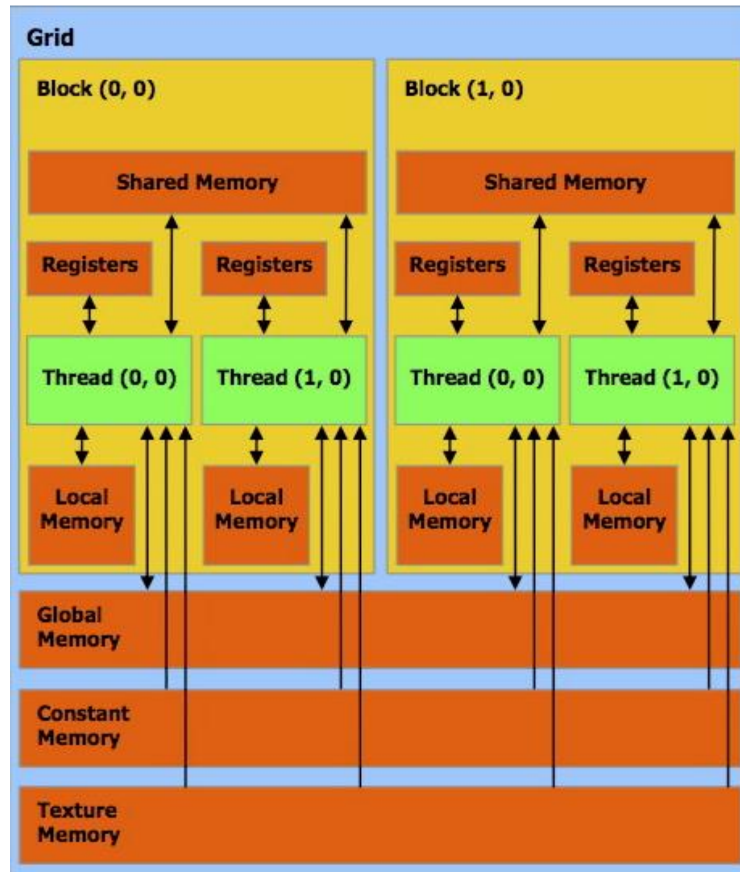
```
// del glavnega programa (CPE)
float* devPtr;
int pitch;
cudaMallocPitch((void**)&devPtr, &pitch,
                width * sizeof(float), height);
myKernel<<<100, 512>>>(devPtr, pitch);

// Kernel funkcija (GPE)
__global__ void myKernel(float* devPtr, int pitch)
{
    for (int r = 0; r < height; ++r) {
        float* row = (float*)((char*)devPtr + r * pitch);
        for (int c = 0; c < width; ++c) {
            float element = row[c];
        }
    }
}
```

Ko je pomnilnik GPE dodeljen v obliki polja (*CUDA arrays*), ga je najbolj učinkovito uporabiti za delo s teksturami, saj je to njegova primarna naloga. Če ta prostor priredimo v računanje za splošne namene, dobimo eno-, dvo- ali tri-dimenzionalna polja elementov, kateremu lahko priredimo eno, dve ali štiri komponente[27].

Ti elementi so lahko različnih tipov:

- 8 bit integer
- 16 bit integer
- 32 bit integer
- 16 bit float
- 32 bit float

Dostopanje do njih je možno preko kernel funkcije, ki zna upravljati s teksturami. Prav tako je potrebno biti pozoren pri programiranju, ker se polja vežejo ena z drugim le preko teksturnih referenc z enakim številom stopenj (1, 2 ali 4). Na Sliki 25 so vidni vsi možni dostopi do različnih vrst pomnilnika CUDA naprave.



Slika 25: Tipi pomnilnika CUDA naprave ter dostopi do njega.
(vir: CUDA Programming Guide 2.3)

V Tabeli 3 so zbrani podatki o tipih pomnilnika GPE, pravicah dostopa in njihovem obsegu ter hitrosti delovanja[28].

Možnost	obseg delovanja	tip pomnilnika	učinkovitost
Branje, pisanje	nit	registri	najhitreje
Branje, pisanje	nit	lokalni pomnilnik	počasno
Branje, pisanje	blok	skupni pomnilnik	hitro
Branje, pisanje	mreža	globalni pomnilnik	počasno
Branje	mreža	konstantni pomnilnik	počasno, + predpomnilnik
Branje	mreža	teksturni pomnilnik	počasno

Tabela 3: Tipi pomnilnika GPE.

5.3.2 Globalni pomnilnik

Globalni pomnilnik nima lastnega predpomnilnika, zato so dostopi vanj, tako časovno kot v smislu pomnilniške prepustnosti, potratni. S posameznim ukazom lahko GPE iz globalnega pomnilnika v registre prebere 4, 8 ali 16 zložne (ang. *byte*) besede. Pri tem si pomaga z vgrajenimi tipi spremenljivk ter njihovimi privzetimi velikostmi. Velikost posameznega tipa se programsko reši z ukazom `sizeof(type)`, kjer `type` predstavlja tip spremenljivke.

Za učinkovito rabo pomnilniške prepustnosti globalnega pomnilnika je zelo pomembna tehnika združevanja pomnilniških dostopov (ang. *coalesced memory access*). Deluje tako, da posamezne dostope do pomnilnika (branje ali pisanje), klicane s strani niti, združi v enotno transakcijo velikosti 32, 64 ali pa 128 zlogov. To je tudi glavni razlog, zakaj je globalni pomnilnik razdeljen na dele oz. segmente po 32, 64 ali 128 zlogov[29].

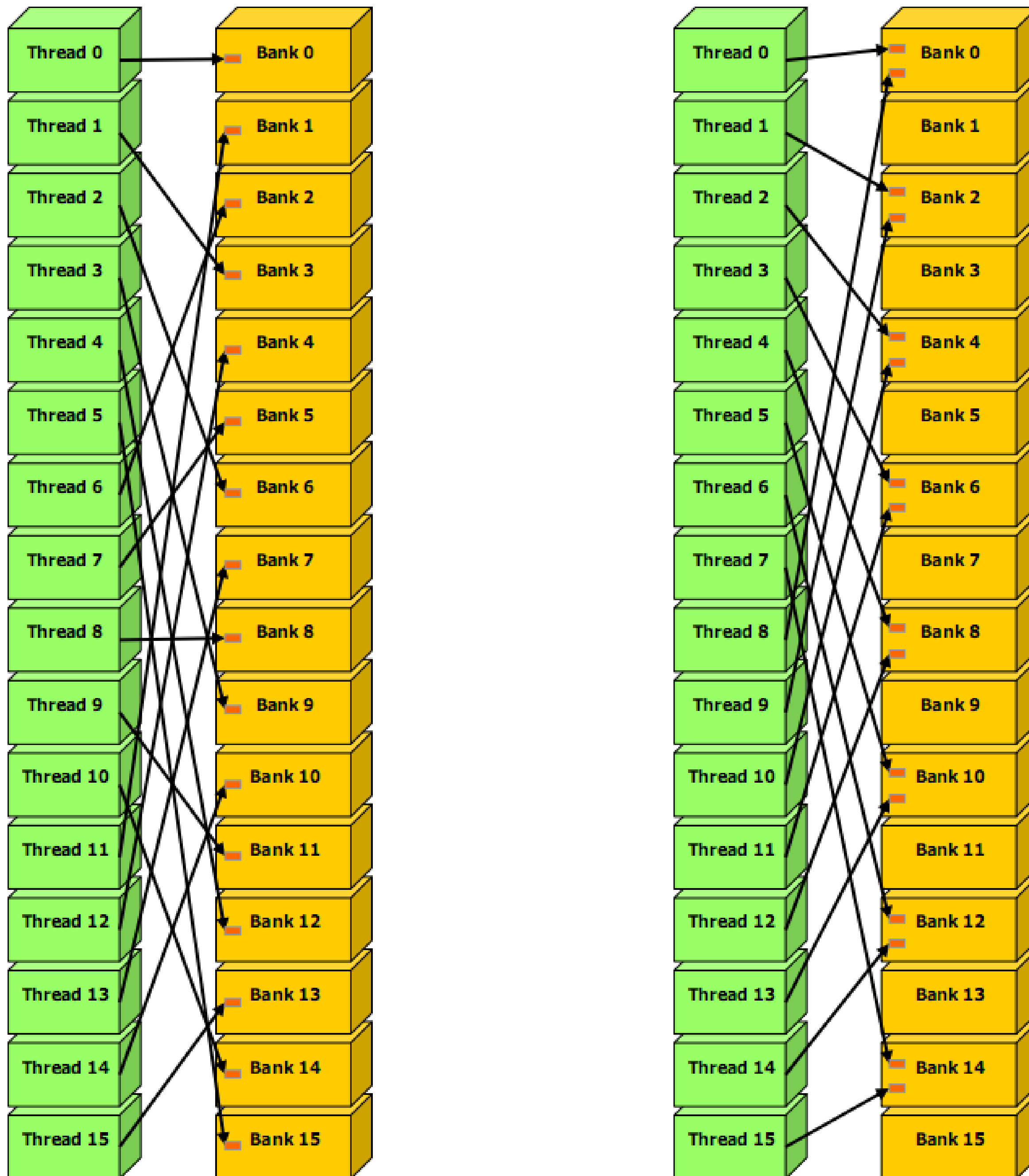
5.3.3 Lokalni pomnilnik

Lokalni pomnilnik, tako kot globalni, nima možnosti predpomnenja, zato je vsak poseg vanj časovno zahteven. Za lokalni pomnilnik velja, da vedno združuje pomnilniške dostope posamezne niti. Dostop do lokalnega pomnilnika potrebujejo le t.i. samodejne spremenljivke. Vanj vpisujejo večje podatkovne strukture oz. polja, ki bi z navadnimi registri zasedli preveč prostora, ter polja, ki jih prevajalnik vidi kot neindeksirana.

5.3.4 Skupni pomnilnik

Skupni pomnilnik se nahaja znotraj čipa, zato je dostopanje do njega v primerjavi z lokalnim in globalnim pomnilnikom hitrejše. Skupni pomnilnik je na voljo vsem nitim istega bloka. V praksi se izkaže za podobno hitrega kot sami registri, dokler ne pride do konflikta med posameznimi moduli oz. bankami pomnilnika. Banke so enako velike posamezne enote skupnega pomnilnika. Dostop do njih lahko poteka vzporedno, neodvisno od drugih. V primeru, da imamo N dostopov (bralnih ali pisalnih) do pomnilnika, ki poteka preko N bank, dobimo N-krat večjo pasovno širino, kar pa velja zgolj v teoriji. V praksi se največkrat zgodi, da dva pomnilniška naslova zahtevata isto banko. Na tem mestu pride do konflikta (ang. *bank conflict)*, ki se nato reši s pomočjo serializacije. Ta poteka tako, da pomnilniški krmilnik konfliktne dostope razdeli in razvrsti v novem, nekonfliktnem vrstnem redu. Število razdelitev je odvisno od števila pomnilniških dostopov, ki zahtevajo eno in isto banko.

Vse CUDA naprave, ki so programsko podprte z različico 1.x (Tabela 8, priloga), imajo na voljo 16KB skupnega pomnilnika (na posamezni SM), razdeljenega enakomerno v 16 bank. Glede na to, da warp enota sprejme 32 niti, smo prikrajšani za polovico možnih dostopov. Problem se rešuje zaporedno, tako da prvi del warp enote (prvih 16 niti) dobi najprej dostop do skupnega pomnilnika in njegovih 16 bank, takoj zatem pa še drugih 16 niti. Na ta način se izognemo konfliktom, prav tako pa podaljšamo dostop do pomnilnika na dva cikla. Leva stran Slike 26 prikazuje brezkonfliktno situacijo dostopanja do pomnilniških bank, medtem ko desna stran prikazuje t.i. *2-way bank conflict,* torej primer ko dve niti poskušata v enakem trenutku dostopati do iste banke[30].



Slika 26: Primer brezkonfliktne (levo) ter konfliktne (desno) situacije dostopa.
(vir: CUDA Programming Guide 2.3)

5.3.5 Registri

Dostop do registrov je najhitrejši, saj se izvede brez dodatnih ciklov, na račun fizične povezanosti oz. minimalne oddaljenosti od SM-jev. V praksi se pokažejo majhni zamiki zaradi konfliktov pri dostopu do istih bank, kar predstavlja podoben problem kot pri skupnem pomnilniku (Slika 26).

5.3.6 Teksturni pomnilnik

Do teksturnega pomnilnika se dostopa preko strojnih TMU-jev, enot za mapiranje tekstur. Te imajo možnost predpomnenja, kar pripomore k hitrosti branja podatkov iz njega. Teksturni pomnilnik je optimiziran za delo z 2D podatki, zato ga najbolje izkoristimo z uporabo warp enote, katere niti imajo naslovne prostore čim bolj skupaj. Warp enota je poslana s strani SMja, in je programsko določljiva. Teksturni pomnilnik je tudi pametno uporabiti v primerih, ko prihaja do naključnih dostopov, saj dobro skriva zakasnitve.

5.3.7 Konstantni pomnilnik

Konstantni pomnilnik ima enako kot teksturni možnost predpomnenja. Zato je lahko podobno hiter kot registri, vendar le dokler vse niti berejo iz istega naslova. V trenutku ko se to spremeni, se vsak cikel branja podvoji. Vsaka nit najprej gleda v predpomnilnik, in v primeru da ga na omenjenem naslovu ni (t.i. *cache miss*), gre gledat v pomnilnik GPE (globalni / lokalni / skupni). Namenjen je za hranjenje konstantnih podatkov, do katerih dostopa kernel funkcija med izvajanjem.

5.4 Različice programskega modela GPE

Različica podprtega programskega modela (ang. *Compute Capability*) posamezne grafične karte je definirana s t.i. *major* in *minor* oznako. Grafične karte z enako major oznako temeljijo na grafičnem procesorju ene in iste družine (npr. serija G8x). Minor oznaka predstavlja postopne izboljšave same arhitekture jedra, in največkrat prinaša nove podprte ukaze in funkcije. V Tabeli 8 (priloga) so zbrani modeli grafičnih procesorjev glede na podprti programski model[31].

Glede na to da bo programska koda tekla na grafičnem procesorju GT200b, so v Tabeli 4 zbrane lastnosti Compute Capability različice 1.3, ki so pomembne tako pri programiranju kot tudi pri poznavanju arhitekture grafičnega čipa.

Opis lastnosti	Velikost / količina
Največje število niti na posamezen blok	512
Največje dimenzije bloka, (x,y,z)	(512,512,64)
Največja velikost posamezne dimenzije mreže blokov	65535
Velikost warp enote	32
Število registrov na eno SM enoto	16384
Skupni pomnilnik, ki je na voljo posamezni SM enoti	16KB v 16 bankah
Količina konstantnega pomnilnika	64KB
Količina lokalnega pomnilnika posamezne niti	16KB
Delovni predpomnilnik konstantnega predpomnilnika	8KB
Največje število aktivnih blokov na posamezno SM enoto	8
Največje število warp enot na posamezno SM enoto	32
Največje število niti na posamezno SM enoto	1024
Velikost kernel funkcije	$2x10^6$ ukazov

Tabela 4: Programske omejitve Compute Capability v1.3.

5.5 Vgrajene spremenljivke

Vgrajene spremenljivke (ang. *built-in variables*) se uporabljajo pri klicu CUDA funkcije in vnaprej določajo velikost bloka, mreže, niti ter njihovih indeksov[32]. Zbrane so v Tabeli 5.

Ime spremenljivke	Tip spremenljivke	Naloga spremenljivke
gridDim	dim3	vsebuje podatek o dimenziji mreže blokov
blockIdx	uint3	predstavlja indeks pozicije bloka v mreži
blockDim	dim3	vsebuje podatek o dimenziji bloka
threadIdx	uint3	predstavlja indeks pozicije niti znotraj bloka
warpSize	int	vsebuje podatek o velikosti warp enote

Tabela 5: Vgrajene spremenljivke za klic kernel funkcije.

Spremenljivka tipa `dim3` se uporablja pri predstavitvah dimenzij. Osnovana je na tipu `uint3`, ki je ne-negativni celoštevilski (ang. *unsigned*) vektorski tip, sestavljen iz treh komponent (x, y, z).

5.6 Tipi CUDA funkcij

Okolje CUDA deluje v okviru treh osnovnih tipov funkcij, ki so prevajalniku prepoznavne. Funkcije razčlenijo glede na izvajanje (host / device) in na zmožnost klicanja le-te.

Funkcija `__device__` deklarira CUDA funkcijo, ki:

- se lahko izvaja le na CUDA napravi (device) in
- se jo lahko kliče le iz CUDA naprave

Deklarirana je kot podfunkcija glavne CUDA funkcije, ki je vidna in lahko operira le znotraj omenjene glavne funkcije. Gostitelj (CPE) je ne vidi, zato jo ne more niti klicati niti uporabljati.

```
__device__ void f(float x)
{
// koda funkcije
}
```

Funkcija __global__ deklarira glavno CUDA funkcijo, imenovano kernel. Za kernel velja:

- izvaja se le na CUDA napravi
- klic funkcije le preko gostitelja (CPE)

Funkcija __host__ deklarira CUDA funkcijo, ki:

- se lahko izvaja le na gostitelju (CPE)
- se jo kliče le preko gostitelja (CPE)

Posebnost funkcije __host__ je v tem, da se jo lahko uporabi v kombinaciji z __device__ funkcijo. V praksi to enostavno pomeni prevajanje funkcije tako za GPE kot tudi za CPE.

Za omenjene funkcije obstajajo določena pravila in programske omejitve, ki jih je treba upoštevati[40].

Funkciji __device__ in __global__ ne podpirata rekurzivnega izvajanja, znotraj njih se ne da deklarirati statičnih spremenljivk in število argumentov, ki jih funkciji sprejmeta, mora biti fiksno.

Funkcija __device__ ima vnaprej določen pomnilniški naslov, katerega se ne da spreminjati, medtem ko funkcija __global__ podpira kazalce, na katere se lahko sklicujemo.

Funkciji __host__ in __global__ ne moreta biti skupaj v uporabi.

Funkcija __global__ vedno vrača rezultat tipa void. Ker je asinhrona, lahko vrne rezultat še preden GPE konča s procesiranjem.

6. Primer programiranja v CUDA okolju

Preden se v naslednjem poglavju lotimo analize zmogljivosti CPE in GPE, si najprej oglejmo program, napisan za CUDA podprte grafične procesorje. CUDA program navadno poteka v štirih glavnih korakih:

1. CPE inicializira podatke
2. CPE pošlje te podatke v pomnilnik GPE
3. GPE obdela dobljene podatke
4. GPE vrne obdelane podatke nazaj v pomnilnik CPE

Program je na videz enostaven, a ravno prav kompleksen za prvo soočanje s CUDA programiranjem. Od CPE bo prevzel polje števil, jih prenesel na GPE, tam vsako izmed njih kvadriral ter prenesel nazaj na CPE.

6.1 Rezervacija pomnilnika

Za rezervacijo tako CPE kot tudi GPE pomnilnika potrebujemo dva kazalca. `A_h` kaže na polje v pomnilniku CPE, `A_d` pa na polje v pomnilniku GPE. Rezervacija CPE pomnilnika poteka po običajnem postopku preko `malloc()` funkcije,

```
// rezervacija pomnilnika CPE
A_h = (float *)malloc(size);
```

medtem ko za GPE obstaja podobna metoda, `cudaMalloc()`.

```
// rezervacija pomnilnika GPE
cudaMalloc((void **) &A_d, size);
```

Za razliko od CPE rezervacije, `cudaMalloc()` dobi kot argument kazalec na `A_d` kazalec. S tem shrani naslov polja v `A_d`. Pomnilnik GPE bi prav tako lahko rezervirali preko ukaza `cudaMallocPitch()` oz. `cudaMalloc3D()`, vendar ni potrebe po 2D oz. 3D prostoru, ker bomo uporabljali zgolj 1D polje.

6.2 Inicializacija in prenos podatkov

Za vnos podatkov v polje elementov ne bomo komplicirali, tako bo vsak element dobil vrednost svojega indeksa.

```
// dodelitev vrednosti elementom
for (int i=0; i<N; i++) A_h[i] = (float)i;
```

Sedaj nastopi drugi korak (izmed omenjenih štirih), kjer je potrebno prenesti vsebino pomnilnika CPE v GPE. Prenos poteka preko funkcije `cudaMemcpy()`, ki kot atribute sprejme kam (`A_d`), od kod (`A_h`), velikost (`size`) ter smer prenosa (`cudaMemcpyHostToDevice`).

```
// prenos podatkov iz CPE v GPE
cudaMemcpy(A_d, A_h, size, cudaMemcpyHostToDevice);
```

6.3 Klic kernel funkcije

Vsak klic CUDA funkcije zahteva vnaprejšnjo definicijo števila blokov in števila niti, ki se bodo izvajale znotraj posameznega bloka. Zato je potrebno oceniti oz. izračunati obe spremenljivki. Število blokov mora vedno biti zadostno, zato je potrebno dodatno preveriti, če se število blokov izide na dano število elementov (N), ki jih bo funkcija obdelala.

```
// stevilo niti znotraj bloka
int block_size = 4;
// koliko blokov potrebujemo za N elementov
int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
```

Naslednji korak je klic CUDA funkcije. Izvede se s pomočjo treh levih in desnih puščic (<<< >>>). Tako NVCC ve, da gre za funkcijo, ki jo mora prevesti in pognati na CUDA napravi. Prvi atribut znotraj puščic predstavlja število blokov (`n_blocks`), ki smo ga prej določili. Drugi atribut (`block_size`) pa predstavlja velikost bloka, torej število niti znotraj vsakega izmed teh `n_blocks` blokov. Celoten ukaz je sestavljen še iz imena funkcije (`kvadriraj`), ter njenih atributov (`A_d, N`).

```
// klic kernel funkcije
kvadriraj <<< n_blocks, block_size >>> (A_d, N);
```

6.4 Kernel funkcija

V tem poglavju bo opisana kernel funkcija, ki sprejme polje elementov in nato vsakega kvadrira. Funkcija se izvede na vsaki niti, kar pomeni N vzporednih izvajanj. Vsaka nit je enolična, od ostalih se razlikuje po thread in block indeksu. Vsaka nit dobi v obdelavo en element polja, ki ga kvadrira. Element polja (`idx`) se izračuna tako, da se množi indeks bloka trenutno uporabljene niti (`blockIdx.x`) s številom niti na blok (`blockDim.x`), temu pa se prišteje še indeks trenutno uporabljene niti (`threadIdx.x`). Če izračunan indeks obstaja znotraj polja elementov (`N`), se vrednost kvadrira.

```
// Kernel funkcija, ki se izvede na GPE
__global__ void kvadriraj(float *a, int N){
int idx = blockIdx.x * blockDim.x + threadIdx.x;
if (idx<N) a[idx] = a[idx] * a[idx];
}
```

6.5 Prenos rezultatov nazaj v CPE

Za prenos podatkov iz GPE nazaj v CPE se poslužimo že uporabljene funkcije `cudaMemcpy`, z obrnjenimi atributi. Prenašamo iz `A_d` v `A_h`, pri tem pa ne smemo pozabiti na smer prenosa, ki je v tem primeru `cudaMemcpyDeviceToHost`.

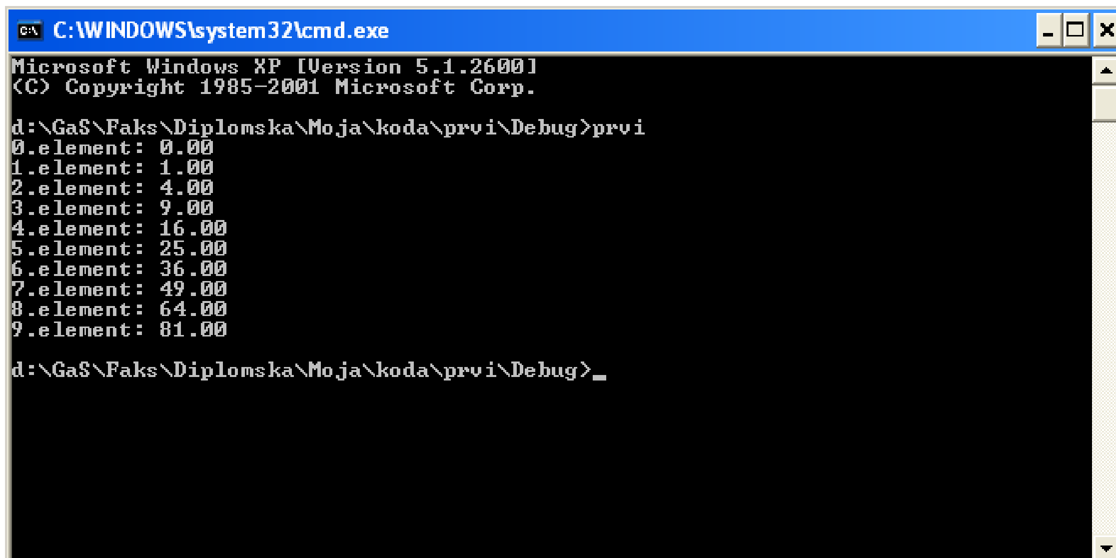
```
// prenos rezultatov v pomnilnik CPE
cudaMemcpy(A_h, A_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
```

Sledi izpis rezultatov na standardni izhod, na koncu pa še sprostimo rezerviran pomnilnik tako CPE (`free(A_h)`) kot tudi GPE (`cudaFree(A_d)`).

```
// izpis rezultatov
for (int i=0; i<N; i++) printf("%d.element: %.2f\n", i, A_h[i]);
// sprostitev CPE in GPE pomnilnika
free(A_h); cudaFree(A_d);
```

6.6 Izpis rezultatov

Za 10 elementov dobimo na standardni izhod rezultate, ki jih prikazuje Slika 27. Celotna koda programa je na voljo v prilogi.



Slika 27: Rezultat na standardnem izhodu v ukazni konzoli.

6.7 Priprava razvojnega okolja Visual Studio 2008

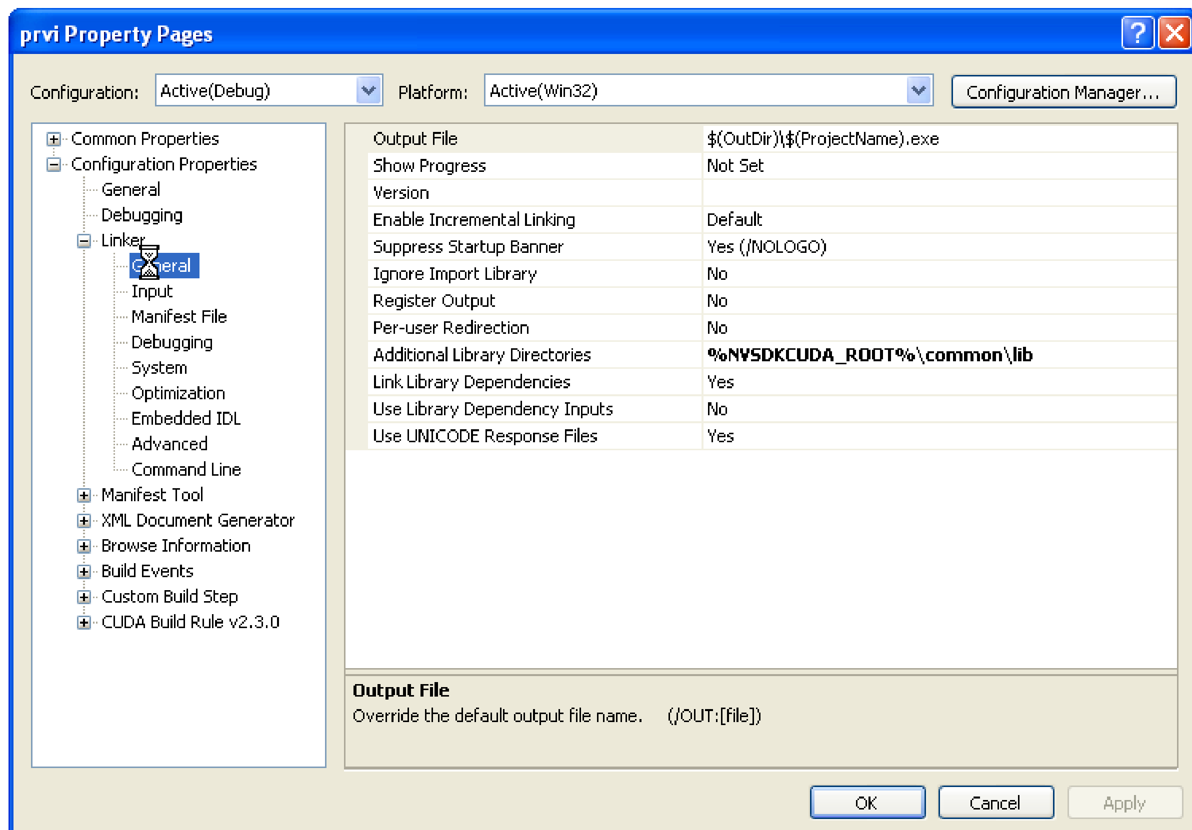
Ko je enkrat programska koda spisana, jo je potrebno prevesti. CUDA se sama po sebi ne integrira v Visual Studio 2008 (VS2008), zato jo je potrebno pripraviti na delo z VS2008. VS2008 ni brezplačen, zato bomo v našem primeru koristili licenco MSDNAA[2].

Najprej je potrebno končnico izvorne kode programa preimenovati v '*.cu*', kajti v nasprotnem primeru bo VS2008 enostavno zavrnil nepoznane CUDA funkcije. Nato je potrebno projektu spremeniti določene atribute, dodati poti za dodatne CUDA knjižnice ter nastaviti povezovalnik. Navodila veljajo za 32-bitno verzijo Windows XP.

Desni klik na projekt s CUDA programom odpre nov meni, iz katerega povsem na dnu izberemo *Properties*. V novem oknu na levi strani izberemo *Linker* in ga razširimo. V podmeniju izberemo *General*, nato pa na desni strani okna poiščemo atribut *Additional Library Directories*. V prostor desno vpišemo oz. dopišemo vrednost

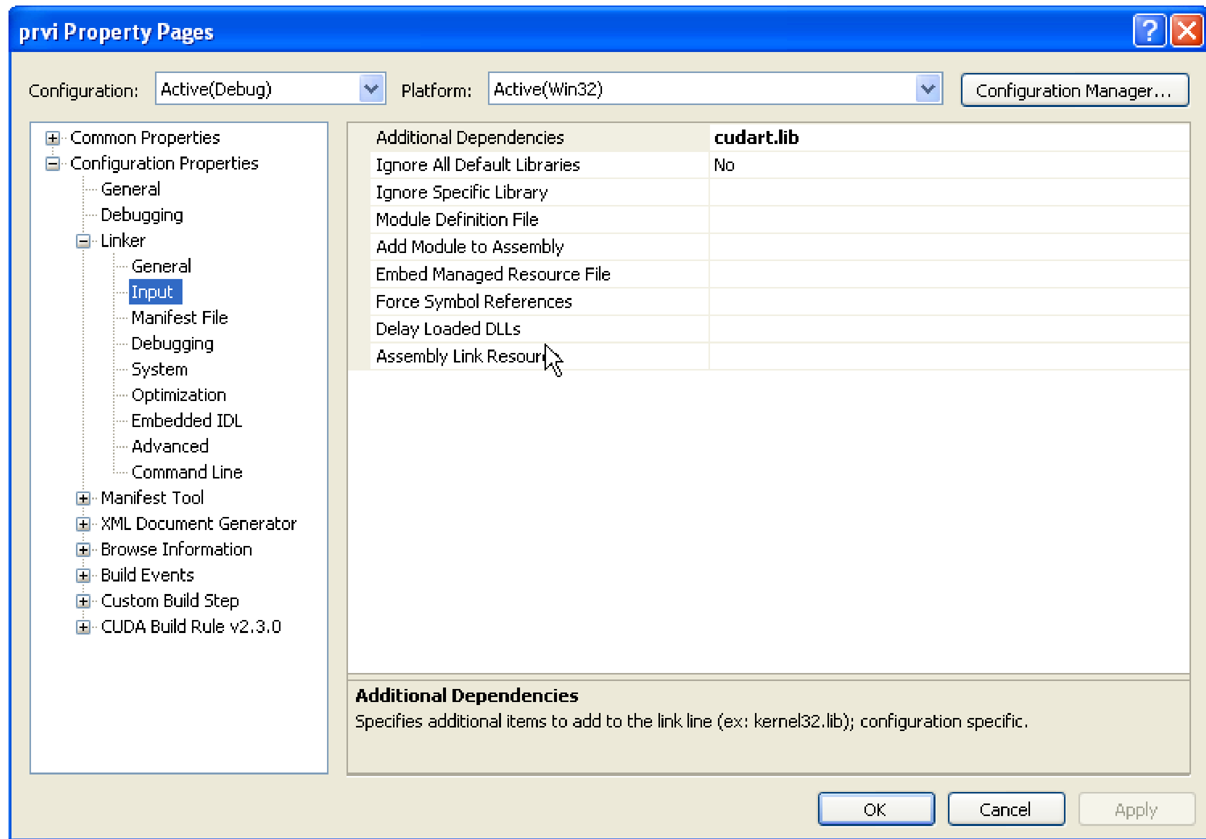
```
"%NVSDKCUDA_ROOT%\common\lib",
```

kot prikazuje Slika 28.



Slika 28: Vnos poti do CUDA knjižnic.

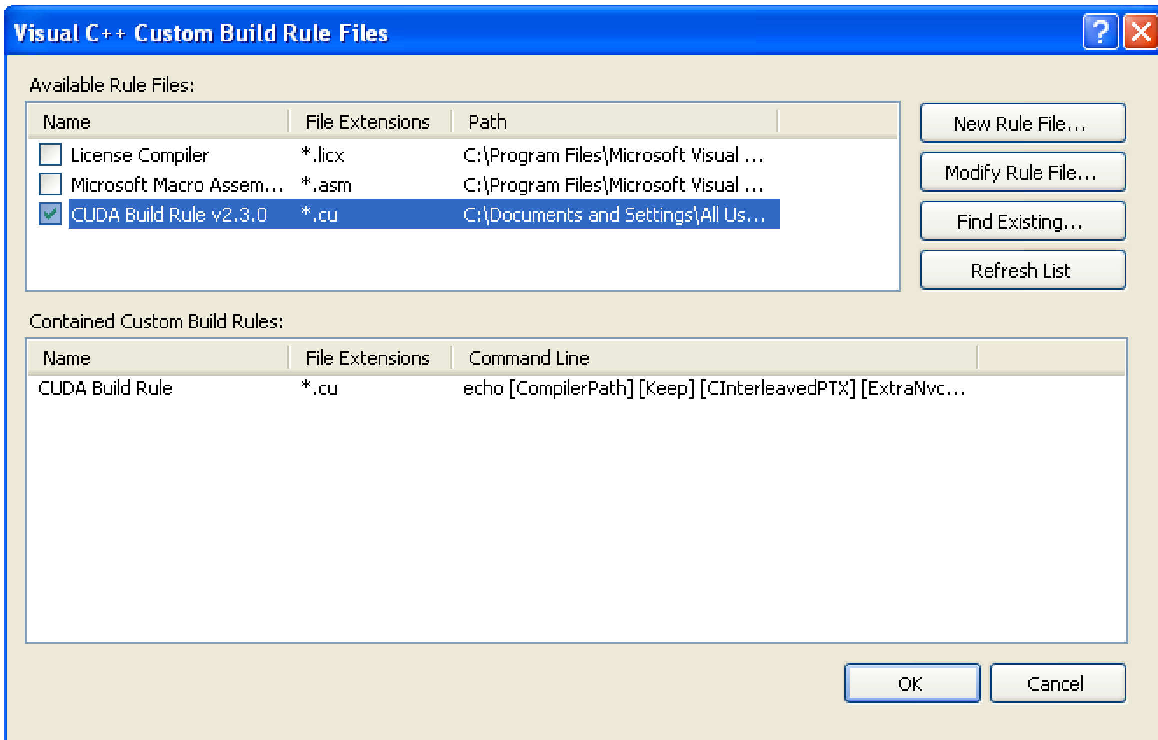
V naslednjem koraku izberemo *Input*(takoj pod *General*), ter nato na desni strani okna atribut *Additional Dependencies*. Za vrednost mu vpišemo knjižnico `cudart.lib`, kot prikazuje Slika 29. Potrdimo z *OK*.



Slika 29: Vnos knjižnice za CUDA Runtime.

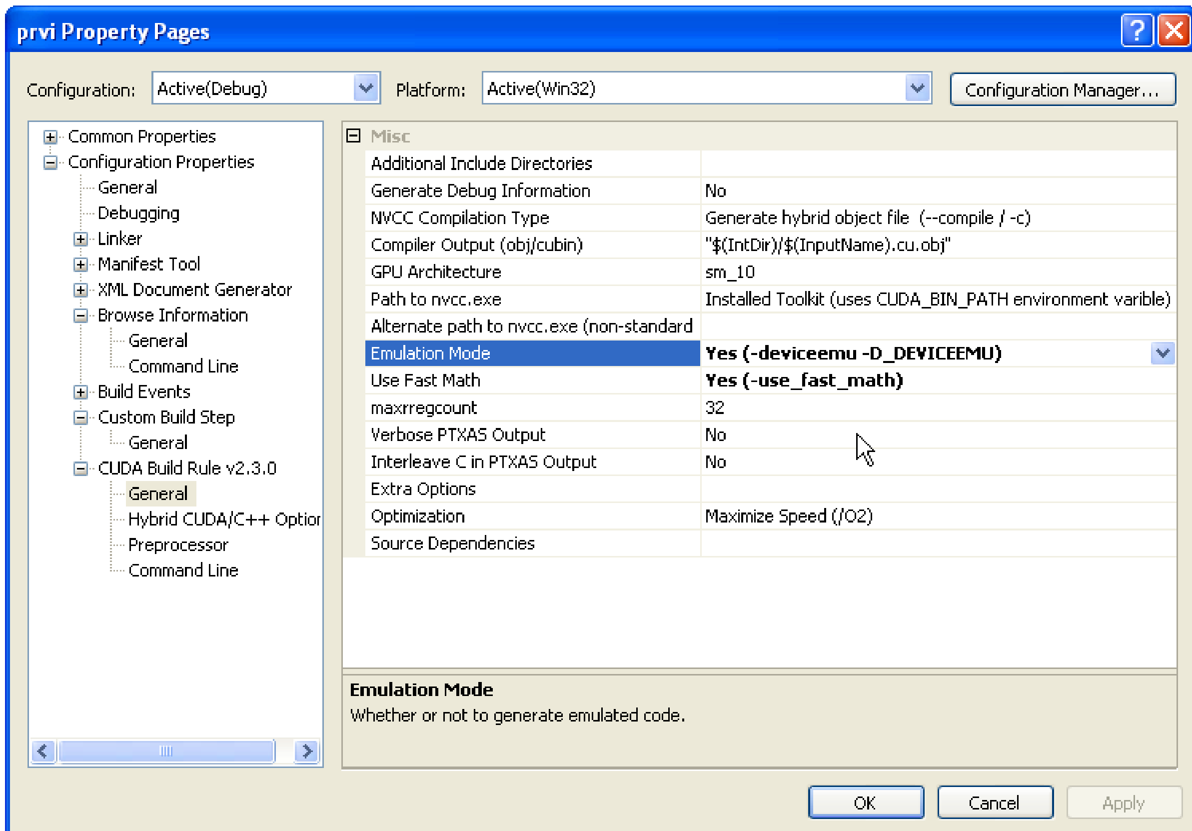
Sedaj je potrebno VS2008 omogočiti uporabo CUDA prevajalnika. Kopijo datoteke `"cuda.rules"` je potrebno prenesti iz `"%NVSDKCUDA_ROOT%\common"` v `"VS2008_install_dir\VC\VCProjectDefaults\"`.

To velja le za VS2008, za starejše oz. novejše različice se pot lahko razlikuje. Ko smo s kopiranjem končali, sledi desni klik na projekt in izberemo *Custom Build Rules...* V oknu ki se odpre, obkljukamo *CUDA Build Rule v2.3.0*, kot prikazuje Slika 30. Potrdimo z *OK*.



Slika 30: Vklop CUDA pravil za pravilno prevajanje aplikacije s strani VS2008.

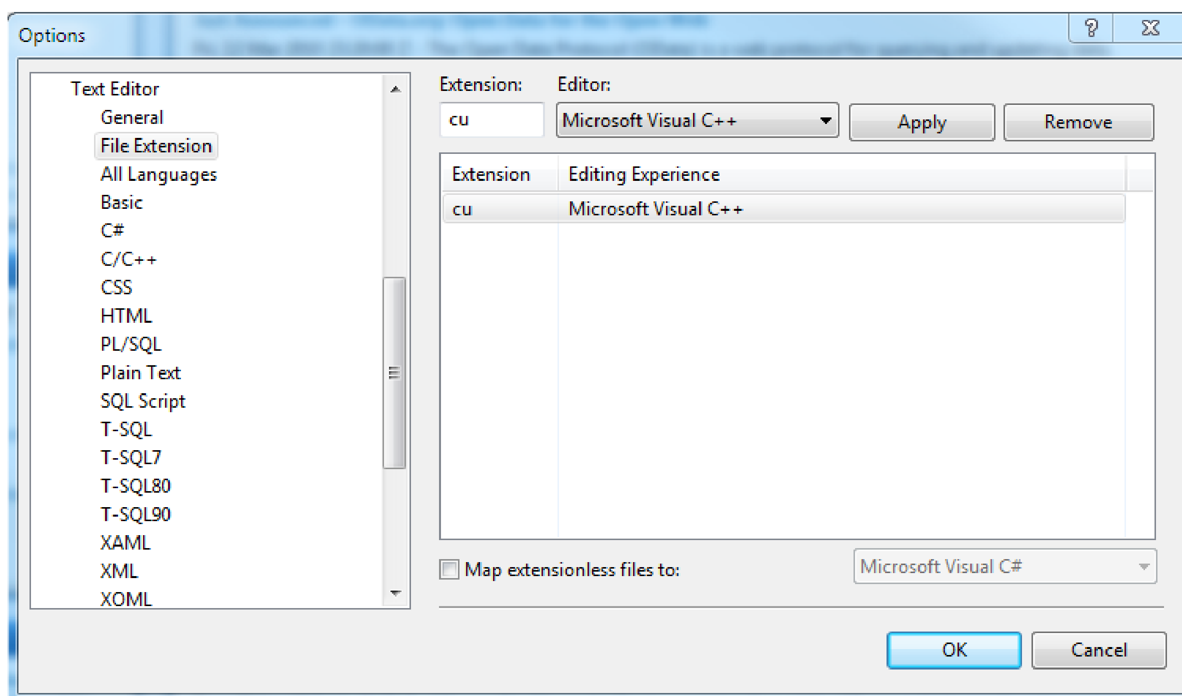
Ostane še zadnji korak, nastavitev na novo uvoženih pravil. Desni klik na projekt, na dnu izberemo *Properties*. Na levi strani okna izberemo *CUDA Build Rule v2.3.0*, ter nato poiščemo atribut *Use Fast Math*. Vrednost nastavimo na *Yes (-use_fast_math),* kot prikazuje Slika 31. Ta izbira sicer ni kritičnega pomena pri prevajanju, je pa hitrostnega pomena pri izvajanju programa. Z njim pridobimo na hitrosti izvajanja matematičnih ukazov, z minimalno izgubo pri natančnosti računanja. V praksi se ta natančnost redko pozna, ker vpliva le na določene ukaze, ki uporabljajo plavajočo vejico z enojno natančnostjo (*float*).



Slika 31: Vklop dodatnih stikal za prevajanje.

Če si dobro ogledamo Sliko 31, opazimo še en atribut, ki je odebeljen (spremenjen glede na privzeto vrednost). *Emulation mode* oz. način emulacije, uporabi CPE za emulacijo CUDA naprave. S tem si olajšamo delo ter programiranje na sistemih brez CUDA naprave. Emulacija ni mišljena za primerjavo zmogljivosti med CPE in GPE, saj je enostavno prepočasna in prekompleksna. Emulacijo naj bi sicer odstranili z verzijo 3.1 (trenutna stabilna 2.3, marec 2010), brez navedenega razloga.

Označevanje kode oz. *text highlighting* je zelo priročno pri pisanju programske kode. Visual Studio 2008 to podpira, vendar le za njemu znane programske jezike. Ker pa CUDA spada pod domeno programskega jezika C/C++, je tudi označevanje zelo podobno. CUDA SDK vsebuje datoteko za pravilno označevanje kode, "`usertype.dat`". Nahaja se v "`%Program Files%\NVidiA Corporation\NVidiA CUDA SDK\C\doc\syntax-_highlighting\visual_studio_8`". Kopijo te datoteke je potrebno prenesti v "`%Program Files%\Microsoft Visual Studio 9.0\Common7\IDE`", v primeru da tam še ne obstaja. V nasprotnem primeru je potrebno vsebino nove datoteke dodati (*append*) obstoječi. Nato odpremo VS2008 in v meniju *Tools* izberemo *Options*. Razširimo razdelek *Text Editor* ter izberemo *File Extension*. V polje *Extension* vpišemo '`cu`' ter v polju *Editor* izberemo `Microsoft Visual C++`, kot kaže Slika 32. Potrdimo z *Apply* ter spodaj z *OK* shranimo nastavitve. Zapremo in ponovno zaženemo VS2008[33].



Slika 32: S pravilno označeno kodo je lažje programirati.

7. Analiza zmogljivosti

V tem poglavju bomo z izbranimi benchmark programi poskušali dokazati zmogljivost CPE in GPE na identičnih algoritmih. Glede na videno teorijo in specifikacije lahko na GPE pričakujemo hitrejše izvajanje kot na CPE.

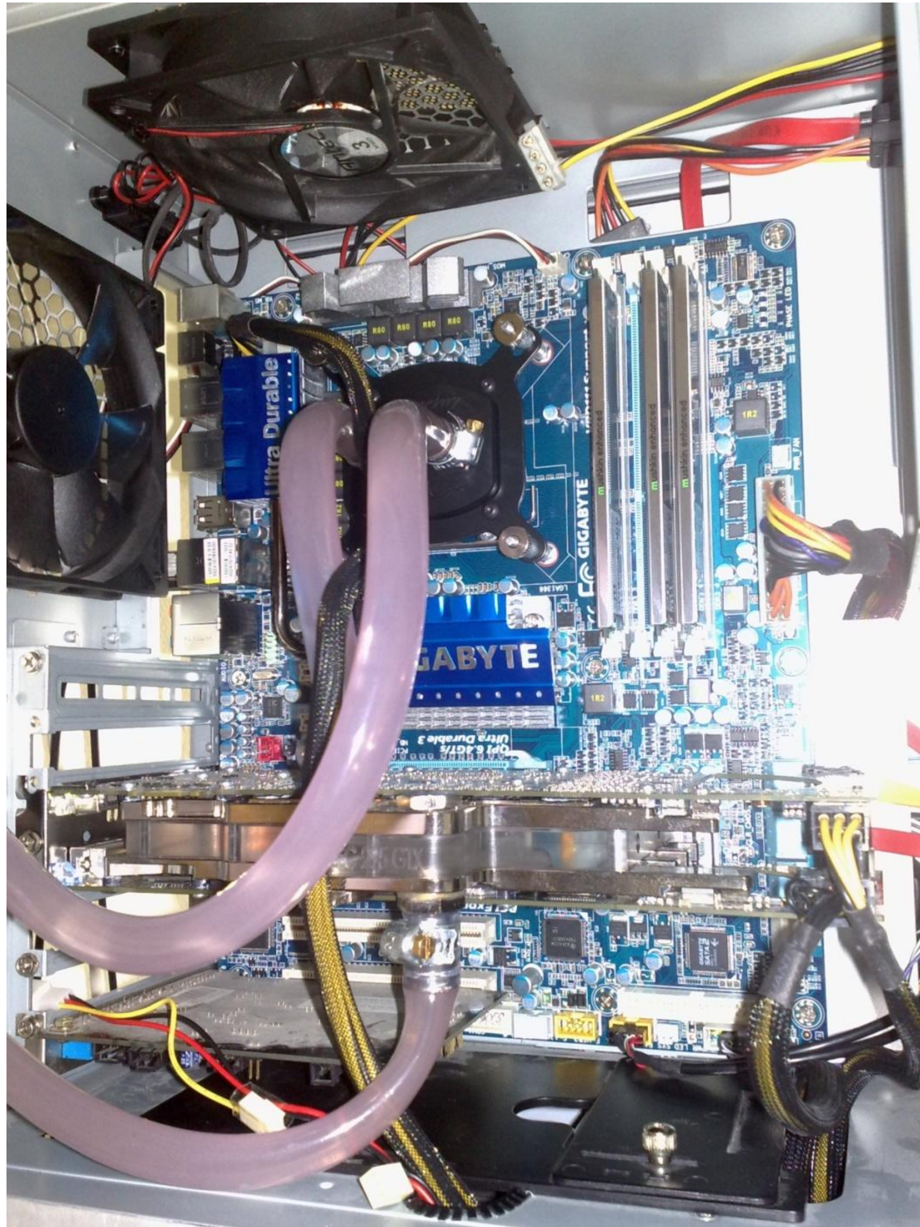
7.1 Strojna oprema

Vsi testi bodo poganjani na treh delovnih postajah, s tremi različnimi CPE ter dvema različnima GPE. Ker bo primerjava potekala med CPE in GPE, bo tako moč dobiti 5 različnih rezultatov. Da pa bo vse skupaj še bolj zanimivo (oz. vroče), bosta oba grafična procesorja testirana tako na privzetih, kot tudi na povišanih frekvencah (ang. *overclocked*, OC). Dva sistema bosta temeljila na 4-jedrnem procesorju, s 6GB pomnilnika DDR2 oz. DDR3 (i7). Šibkejši je klasična HP delovna postaja, model DC7800, s procesorjem Intel Core2 Quad 6600, pri čemer mu računanje na GPE prevzema NVidia Geforce 8400GS. Močnejši je iz zadnje generacije Intelove arhitekture Core i7, model 920. Tretji, najmočnejši, pa bo zasnovan na osnovi najnovejšega 6-jedrnega procesorja družine Core i7 (prvotno mišljen kot Core i9), model 980 Extreme. Oba Core i7 bosta testirana z dvoprocesorsko NVidio GeForce GTX 295, omenjeno že v uvodnih poglavjih, med seboj pa bodo povezani preko osnovne plošče Gigabyte GA-EX58 UD3R. Za lažjo predstavo so v Tabeli 6 zbrani tehnični podatki vseh treh delovnih postaj.

<table>
<tr><th>Komponenta</th><th>PC6 295</th><th>PC4 295</th><th>PC4 8400</th></tr>
<tr><td colspan="4">Procesor</td></tr>
<tr><td>Naziv</td><td>Core i7 980X</td><td>Core i7 920</td><td>Core2 Quad 6600</td></tr>
<tr><td>Čip</td><td>Gulftown</td><td>Bloomfield</td><td>Kentsfield</td></tr>
<tr><td>Proizvodnji proces</td><td>32nm</td><td>45nm</td><td>65nm</td></tr>
<tr><td>Stepping</td><td>A0 (ES)</td><td>D0</td><td>G0</td></tr>
<tr><td>Frekvenca</td><td>3.33 GHz</td><td>2.66 GHz</td><td>2.4 GHz</td></tr>
<tr><td>Število jeder</td><td>6 (12 z HT)</td><td>4 (8 z HT)</td><td>4</td></tr>
<tr><td>Predpomnilnik L2</td><td>4 x 256 KB</td><td>4 x 256 KB</td><td>8 MB</td></tr>
<tr><td>Predpomnilnik L3</td><td>12 MB</td><td>8 MB</td><td>/</td></tr>
<tr><td colspan="4">Grafična karta</td></tr>
<tr><td>Naziv</td><td colspan="2">nVidia GeForce GTX 295</td><td>nVidia GeForce 8400 GS</td></tr>
<tr><td>Procesor</td><td colspan="2">GT200b</td><td>G86</td></tr>
<tr><td>Proizvodnji proces</td><td colspan="2">55nm</td><td>80nm</td></tr>
<tr><td>Število procesorjev</td><td colspan="2">2</td><td>1</td></tr>
<tr><td>Število SPjev</td><td colspan="2">2 x 240</td><td>16</td></tr>
<tr><td>Količina pomnilnika [MB]</td><td colspan="2">2 x 896</td><td>256</td></tr>
<tr><td>Tip pomnilnika</td><td colspan="2">DDR3</td><td>DDR2</td></tr>
<tr><td>Frekvenca jedra [MHz]</td><td colspan="2">576</td><td>459</td></tr>
<tr><td>Frekvenca SPjev [MHz]</td><td colspan="2">1242</td><td>918</td></tr>
<tr><td>Frekvenca pomnilnika [MHz]</td><td colspan="2">999</td><td>400</td></tr>
<tr><td>Frekvenca jedra (OC) [MHz]</td><td colspan="2">700</td><td>500</td></tr>
<tr><td>Frekvenca SPjev (OC) [MHz]</td><td colspan="2">1512</td><td>1242</td></tr>
<tr><td>Frekvenca pomnilnika (OC) [MHz]</td><td colspan="2">1100</td><td>550</td></tr>
<tr><td>Pomnilniško vodilo</td><td colspan="2">448 bit / GPE</td><td>64 bit</td></tr>
<tr><td>jedro OC [%]</td><td colspan="2">21,53</td><td>8,93</td></tr>
<tr><td>SP OC [%]</td><td colspan="2">21,74</td><td>35,29</td></tr>
<tr><td>pomnilnik OC [%]</td><td colspan="2">10,11</td><td>37,50</td></tr>
<tr><td>skupaj OC [%]</td><td colspan="2">17,79</td><td>27,24</td></tr>
</table>

Tabela 6: Tehnične specifikacije testnih komponent.

Delovne postaje so poimenovane po principu jeder CPE ter modela grafične karte. PC4 8400 je tovarniško hlajen zračno, medtem ko sta PC4 295 ter PC6 295 hlajena vodno (CPE in GPE, Slika 33).



Slika 33: Vodno hlajen PC6 295.

7.2 Programska oprema

Ker so nekateri testni programi prirejeni za okolje Linux, bo testiranje potekalo tako na Windows kot tudi na Linux platformi. V Tabeli 7 so zbrani podatki o uporabljeni programski opremi ter njihovemu namenu.

Operacijski sistem	Različica	Pomnilniško naslavljanje	Namen
Microsoft Windows XP Professional	5.1.2600.5512 Service Pack 3	32 bit	programiranje v emulacijskem načinu
Microsoft Windows 7 Ultimate	6.1.7600	64 bit	poganjanje testnih programov
Ubuntu Linux	9.04	64 bit	poganjanje testnih programov
NVidia grafični gonilniki	197.15 (Win), 195.36.15 (Linux)	32 in 64 bit	omogoča delovanje grafične karte
CUDA Toolkit	2.3	32 in 64 bit	prevajalnik, CUFFT, CUBLAS
CUDA SDK	2.3	32 in 64 bit	primeri uporabe, dokumentacija

Tabela 7: Uporabljena programska oprema.

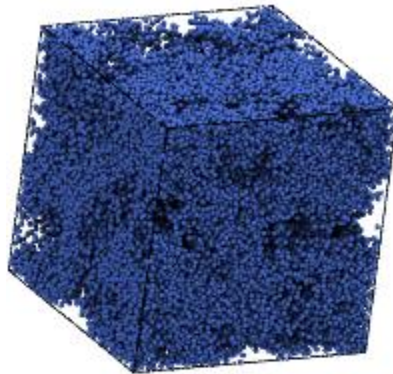
7.3 HOOMD Blue Edition

Prvo testiranje zmogljivosti bo preverjal HOOMD (Highly Optimized Object-oriented Many-particle Dynamics), ki je osnovan na Python ogrodju. Namenjen je uporabi na eni sami delovni postaji, s poudarkom na paralelnem procesiranju preko GPE. HOOMD je namenjen simulaciji dinamike delcev molekul. Na testu bo uporabljena zadnja verzija HOOMD, 0.82. Za pravilno delovanje potrebuje nameščenega Python-a, različico 2.6.

HOOMD podpira naslednje vrste simulacij[34]:

- Pair Potentials
    - Lennard-Jones
    - Gaussian
    - CGCMM
- Bond Potentials
    - FENE
    - Harmonic
- Angle Potentials
    - Harmonic
    - CGCMM
- Dihedral/Improper Potentials
    - Harmonic
- Wall Potentials
    - Lennard-Jones

Za naš test bo aktualna simulacija *Lennard-Jones Liquid MD,* ki je v programskem paketu že pripravljena kot *benchmark* skripta. Skripta v obdelavo pošlje 64,000 delcev, pri čemer je vrednost APF (ang. *Atomic Packing Factor*) enaka 0.2. Zmogljivost se meri z enoto TPS (ang. *Time steps Per Second*). Rezultat zmogljivosti predstavlja povprečje 50,000+ korakov. Slika 34 prikazuje posnetek enega samega TPS med obdelavo s pomočjo orodja VMD (*Visual Molecular Dynamics*).



Slika 34: Vseh 64k delcev, zajetih v delčku sekunde.
(vir: http://codeblue.umich.edu/hoomd-blue/)

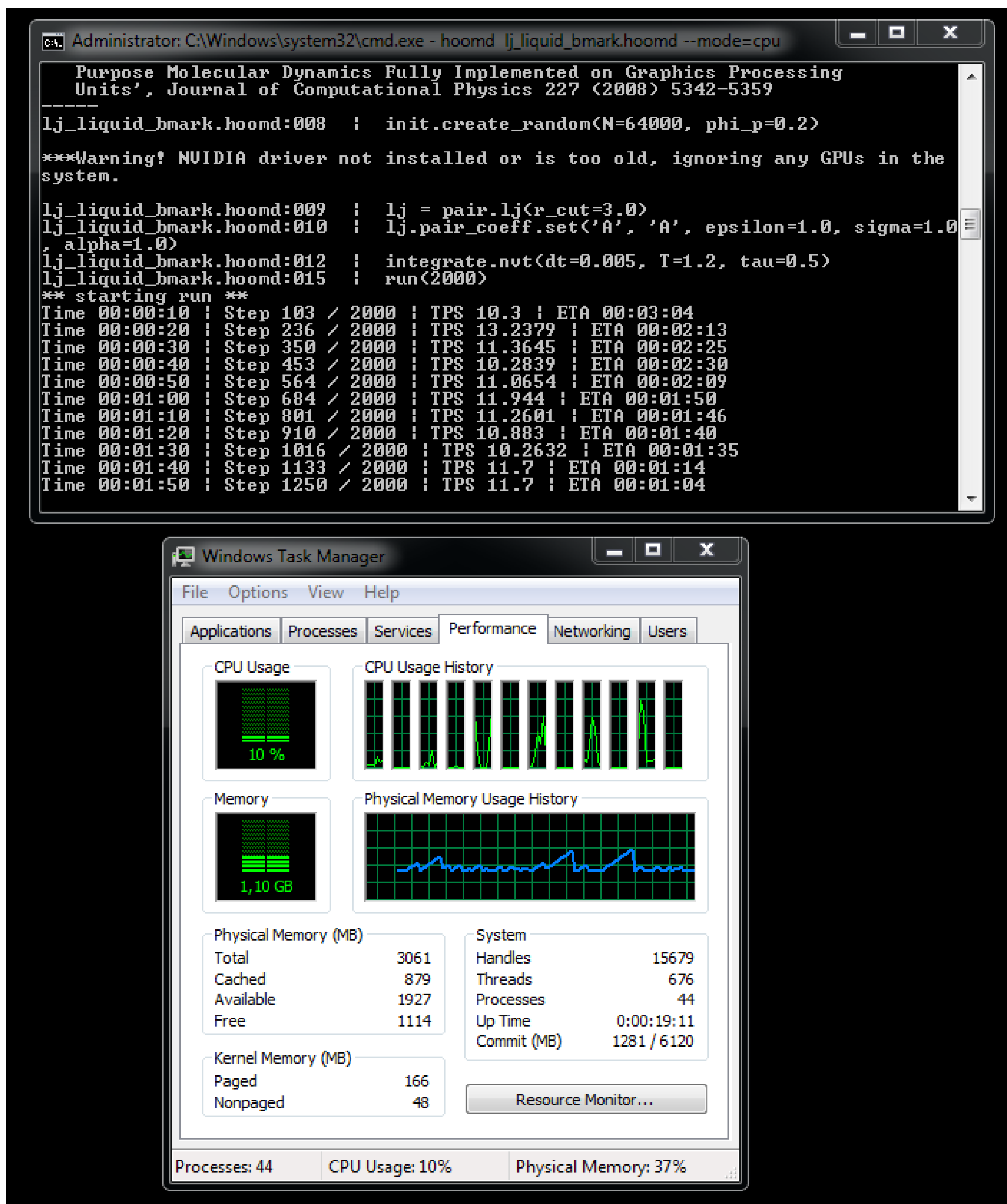
HOOMD je konzolna aplikacija, ki deluje preko stikal. Test bo najprej pognan na CPE, nato pa na GPE. Za zagon benchmark skripte preko CPE je potreben ukaz

```
hoomd lj_liquid_bmark.hoomd --mode=cpu
```

oziroma

```
hoomd lj_liquid_bmark.hoomd --mode=gpu (--gpu=0,1)
```

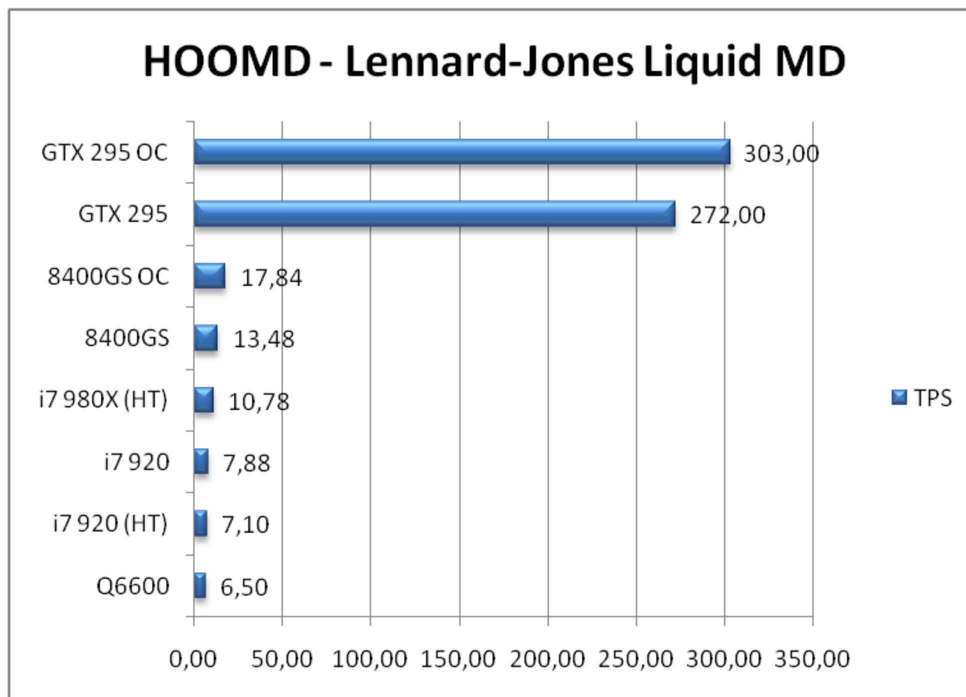
za zagon preko GPE.



Slika 35: HOOMD slabo izkorišča več jeder (Core i7 980X, HT vklopljen).

Zaradi možnih odstopanj je bil vsak test ponovljen trikrat, iz tega pa je bilo narejeno povprečje. Kot je razvidno s Slike 36, je GPE brez konkurence. Nizko cenovna 8400GS (dobrih 20€) prekaša štiri-jedrni Q6600 za 100% pri 5-krat nižji ceni, medtem ko 8400GS OC preračuna kar 126% več TPSjev kot i7 920. Opazi se tudi programsko slabo podprto tehnologijo HT, saj so štiri fizična jedra učinkovitejša od še dodatnih štirih navideznih.

GTX 295 v *singleGPU* načinu konča simulacijo 34-krat hitreje kot i7 920, s povišanimi frekvencami pa kar 38-krat. V najvišjem cenovnem razredu tisočaka evrov vreden šest-jedrnik doseže komaj 4% enkrat cenejše GTX 295.

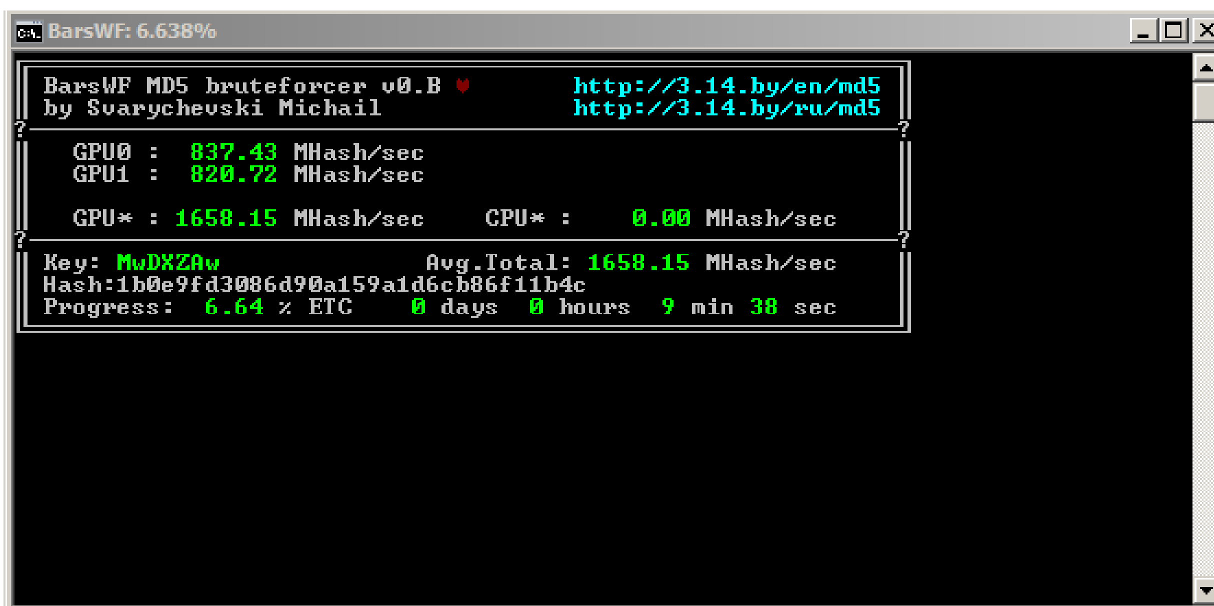


Slika 36: Velike razlike med CPE in GPE.

7.4 BarsWF MD5

MD5 je zgoščevalna (ang. *hash*) funkcija, ki za dane podatke izračuna vrednost fiksne velikosti (128 bitov). Predstavljena je bila leta 1991, kot naslednik MD4. Leta 1996 so našli prvo luknjo v algoritmu, 8 let kasneje pa še nekaj novih, zato se za resne varnostne mehanizme odsvetuje. MD5 ima lastnost, da mu že najmanjša sprememba danih podatkov drastično spremeni vrednost ključa. Najbolj uporabna je za preverjanje integritete datotek. Če se MD5 vrednost ujema na vhodni in izhodni datoteki, se skoraj zagotovo ujema tudi sama vsebina datoteke. V primeru da se MD5 vrednosti ne ujemajo, se prav tako ne ujemajo podatki, katere predstavlja ključ[35].

Ker iskanje MD5 ključa z *brute-force* načinom predstavlja možnost visoke stopnje paralelnega procesiranja, je ta test kot nalašč za preizkus procesorske zmogljivosti. Za programsko rešitev je bil izbran *BarsWF MD5 0.B* (Slika 37), ki je trenutno najhitrejše orodje za odkrivanje ključa. Na voljo so tri različice; CUDA, SSE2 ter Brook[36]. Slednji je namenjen AMD grafičnim kartam. Osredotočili se bomo na SSE2 (CPE) ter CUDA (GPE) različico.



Slika 37: BarsWF CUDA, eden izmed najhitrejših iskalcev MD5 ključa.

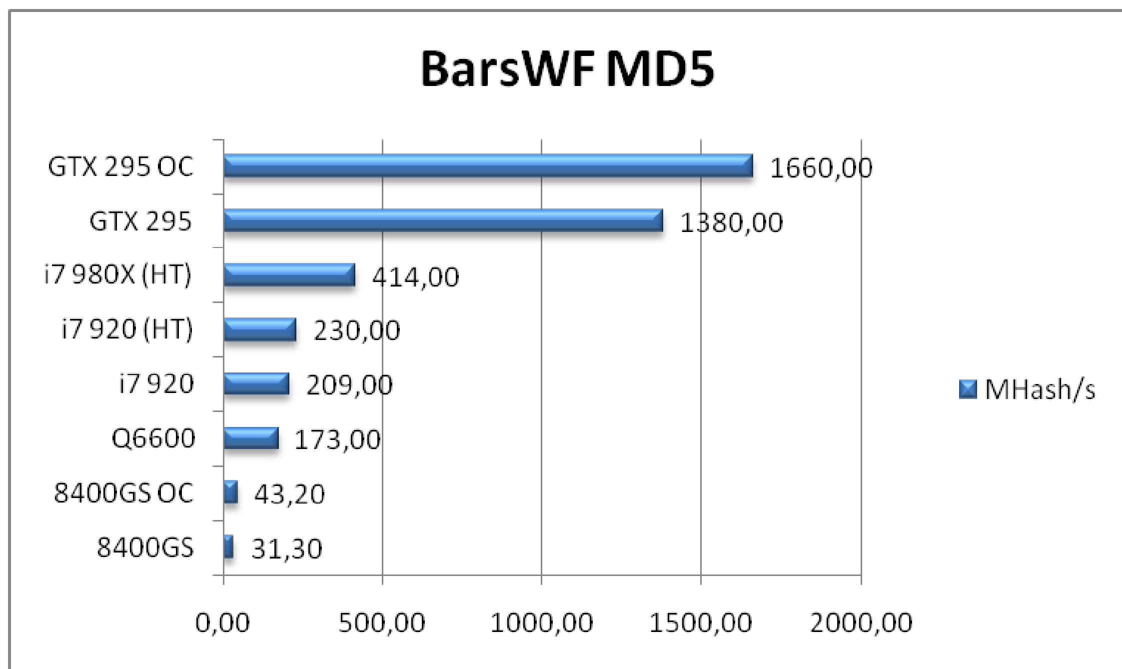
V testu je bil za vse komponente izbran enak ključ. Zmogljivost je merjena v MHash/s, torej koliko milijonov hash funkcij testirana komponenta pregleda v eni sekundi. Za zagon CPE različice je potreben ukaz

```
barswf_SSE2_x64 -h 1b0e9fd3086d90a159a1d6cb86f11b4c -c Aa
```

oz. za GPE ukaz

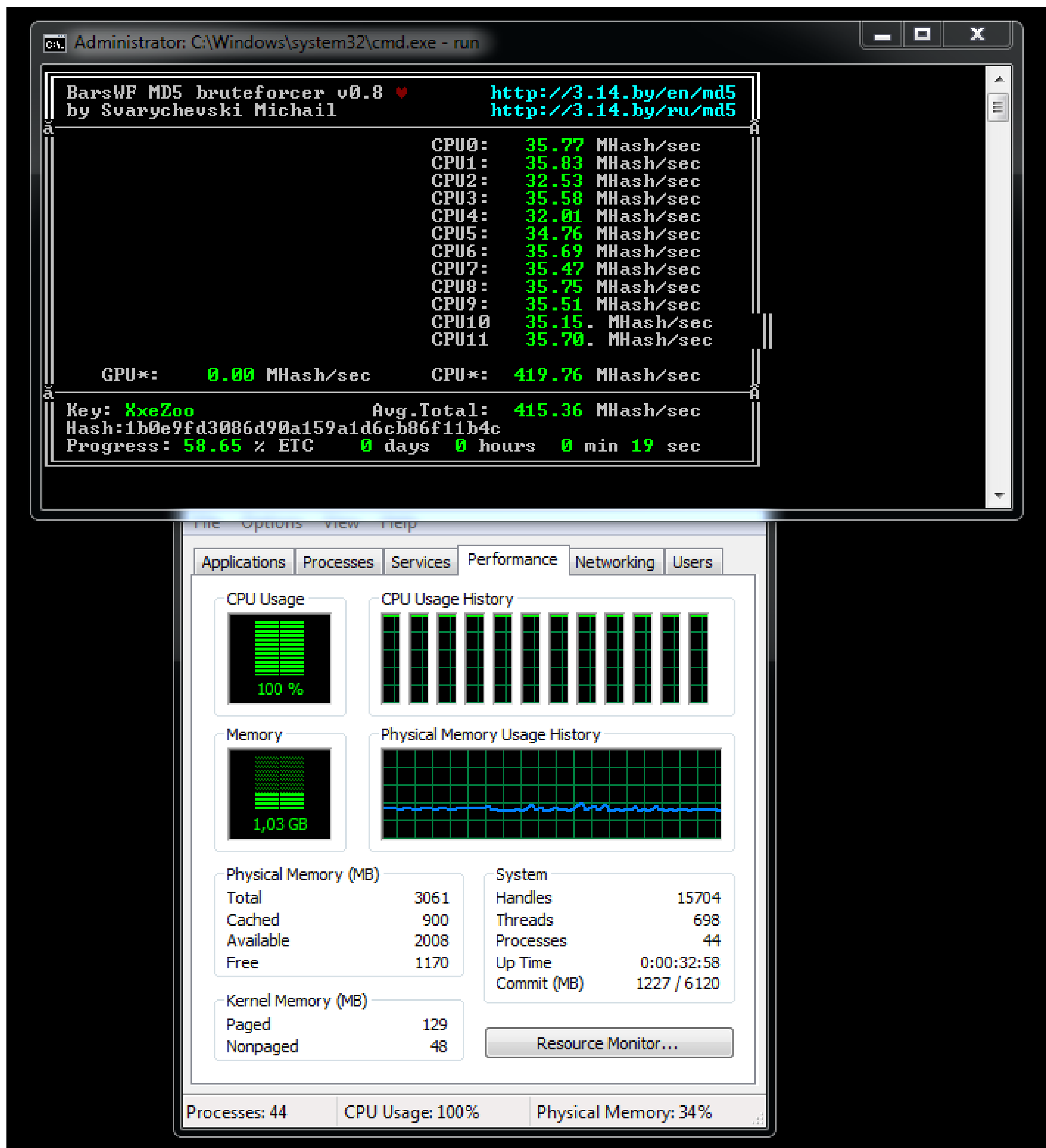
```
barswf_cuda_x64 -h 1b0e9fd3086d90a159a1d6cb86f11b4c -c Aa –
cpu_n=0.
```

Da je GPE resnično boljša izibira pri paralelnem računanju, dokaže tudi iskanje MD5 ključa (Slika 39).



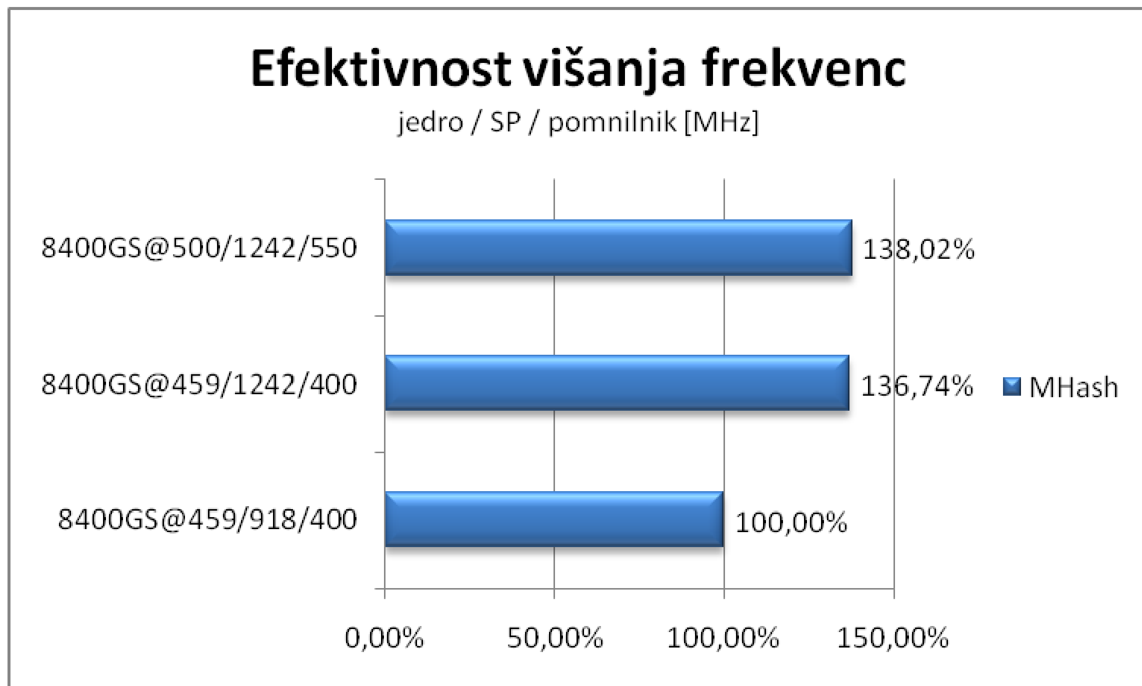
Slika 38: Štirikratna razlika med *GTX 295 OC* in *i7 980X*.

GTX 295 kar 6-krat hitreje najde ključ kot i7 920 z vključenim HT. Da je aplikacija res dobro spisana, potrjuje tudi dejstvo, da so bila ob izvajanju SSE2 različice vsa jedra 100% obremenjena (Slika 38).



Slika 39: BarsWF SSE2 obremeni vseh 12 jeder (HT vključen).

8400GS OC se ob svoji skromni procesorski moči ter predvsem pomanjkanju predpomnilnika lahko kvečjemu primerja z enim samim jedrom procesorja Q6600. Ponovno je zanimivo, kakšen vpliv ima povišanje frekvenc grafične karte. 8400GS OC premelje kar 38% več ključev, pri čemer je frekvenca SPjev povečana za 35,29%. Delovna frekvenca pomnilnika je bila zvišana iz 400 na 550 MHz, kar predstavlja 37,5% višjo pomnilniško prepustnost, ki je pri *low-end* grafičnih kartah največkrat ozko grlo.
V tem primeru se to ni prav nič poznalo, prav tako nam višja frekvenca ure jedra (iz privzetih 459 na 500 MHz) ni nič pomagala (Slika 40).



Slika 40: Najbolj efektivno je višanje frekvenc SPjev.

7.5 Parboil Benchmark Suite

Parboil je skupek konzolnih aplikacij, spisanih v C++ in C for CUDA jeziku[37]. Njihov osnovni namen je s kratkimi testi predstaviti zmogljivost CUDA GPE. Za primerjavo je implementirana tudi CPE različica, ki pa je izpopolnjena za enoprocesorske, enojedrne sisteme. Glede na testno okolje, sestavljeno iz 4- in 6-jedrnika, lahko teoretično vzamemo dobljene rezultate proporcionalno. Primerjava bo izvedena na treh testih Parboil paketa:

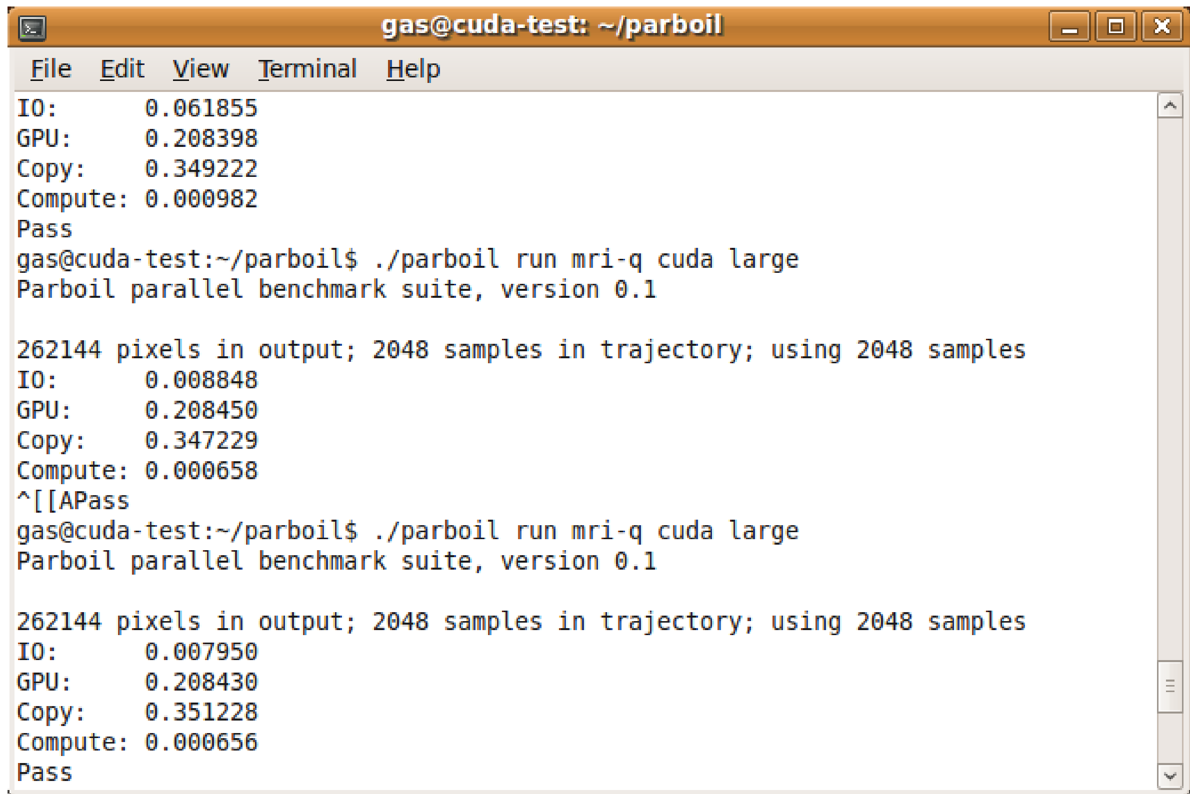
- MRI-Q
- CP
- SAD

Ker je Parboil spisan za operacijski sistem Linux (Slika 41), je bilo potrebno najprej pripraviti testno okolje. Osnovo predstavlja 64-bitna distribucija Linuxa, brezplačni Ubuntu 9.04[38]. Zadnja različica (9.10) zaradi nekompatibilnosti s CUDA SDK ni bila izbrana. V nadaljevanju bodo opisani posamezni testi ter dobljeni rezultati.

Zaradi res hitrega izvajanja na GPE (<1s) je bilo testiranje na povišanih frekvencah brez pomena in je zato izpuščeno iz testa. Pri tem ne smemo mimo dejstva, da testi predstavljajo le majhen delež celotne obdelave, s katerim dobimo okvirno predstavo zmogljivosti. Rezultati v vseh testih so merjeni v sekundah izvajanja, upošteva pa se vsota vseh časov (*IO, GPU, Copy* in *Compute*).

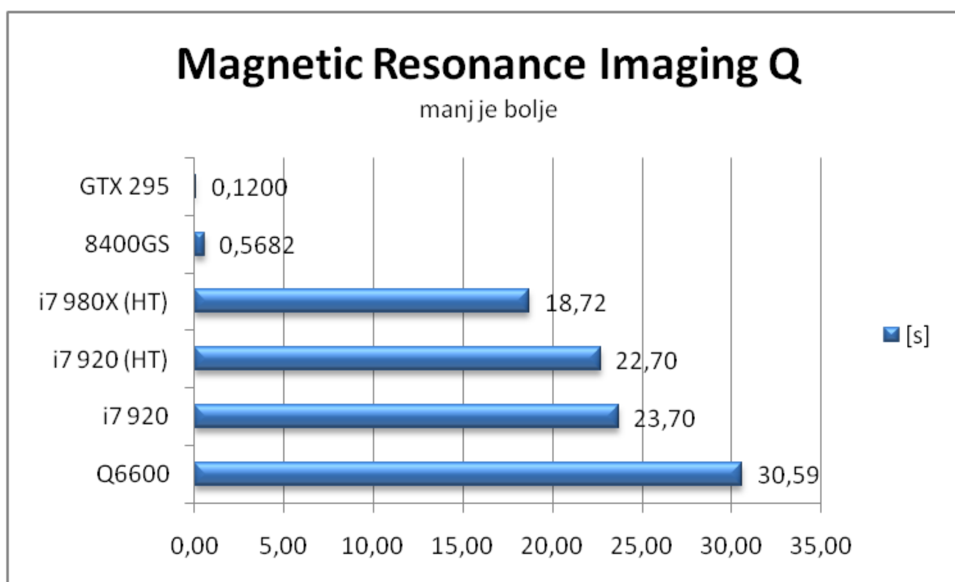
Vsi testi se zaganjajo preko Parboil aplikacije ter dodatnih stikal:

```
./parboil run [test] [cuda/cpu] [dataset]
```



Slika 41: Parboil znotraj Ubuntu terminala.

Magnetic Resonance Imaging Q (MRI-Q) predstavlja izračun Q matrike, ki je del konfiguracije skenerja. Ta je uporabljen v 3D rekonstrukciji magnetno-resonančne slike (MRI). V osnovi preračunava inverzno Fourierjevo transformacijo na vhodnih podatkih.



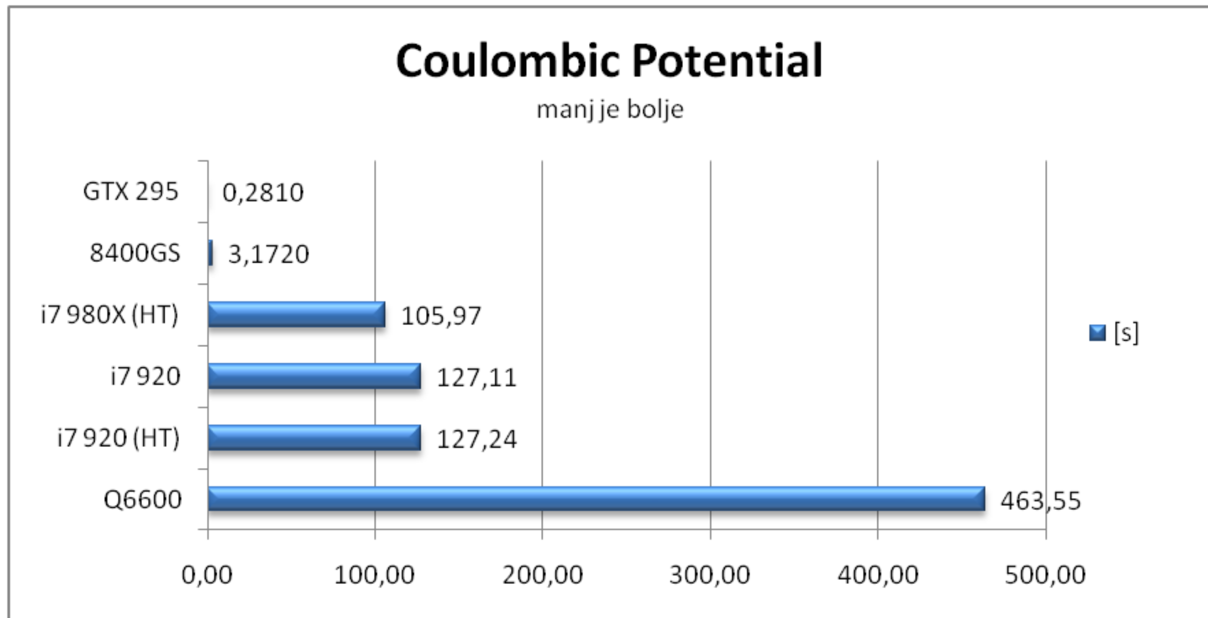
Slika 42: Parboil izkoristi le eno CPE jedro, kar je velik minus.

Kot je razvidno s Slike 42, je 8400GS izračunala Q matriko skoraj 54-krat hitreje kot Q6600, vendar ne smemo pozabiti, da je 100% obremenjeno le eno jedro. Faktor 54 bi lahko teoretično z zelo dobro implementacijo CPE algoritma zmanjšali na približno 15, kar je, glede na ceno obeh primerjanih komponent, še vedno veliko. Podobna zgodba je v najvišjem razredu, kjer GTX295 preračuna Fourierjeve transformacije kar 156-krat hitreje kot i7 980X, ter skoraj 255-krat hitreje od že omenjenega Q6600. CPE različica ponovno izrabi le eno jedro.

Naslednji test iz Parboil paketa je Cuolombic Potential (CP), ki računa iz podatkov na osnovni Cuolombovega zakona, ki pravi, da se sila med dvema točkastima nabojema zmanjšuje z razdaljo. Absolutna vrednost sile je premo sorazmerna produktu obeh nabojev in obratno sorazmerna s kvadratom razdalje med njima. Sila je privlačna, če imata naboja različen predznak (eden pozitivno in drugi negativno), in odbojna, če imata enak predznak[39].

V našem primeru CP izračuna mrežo ter potencialno energijo nekega hipotetičnega naboja vsake točke v tej mreži. Pri tem pošlje pozitiven naboj v globalni minimum, negativen naboj pa v globalni maksimum izhodne mreže.

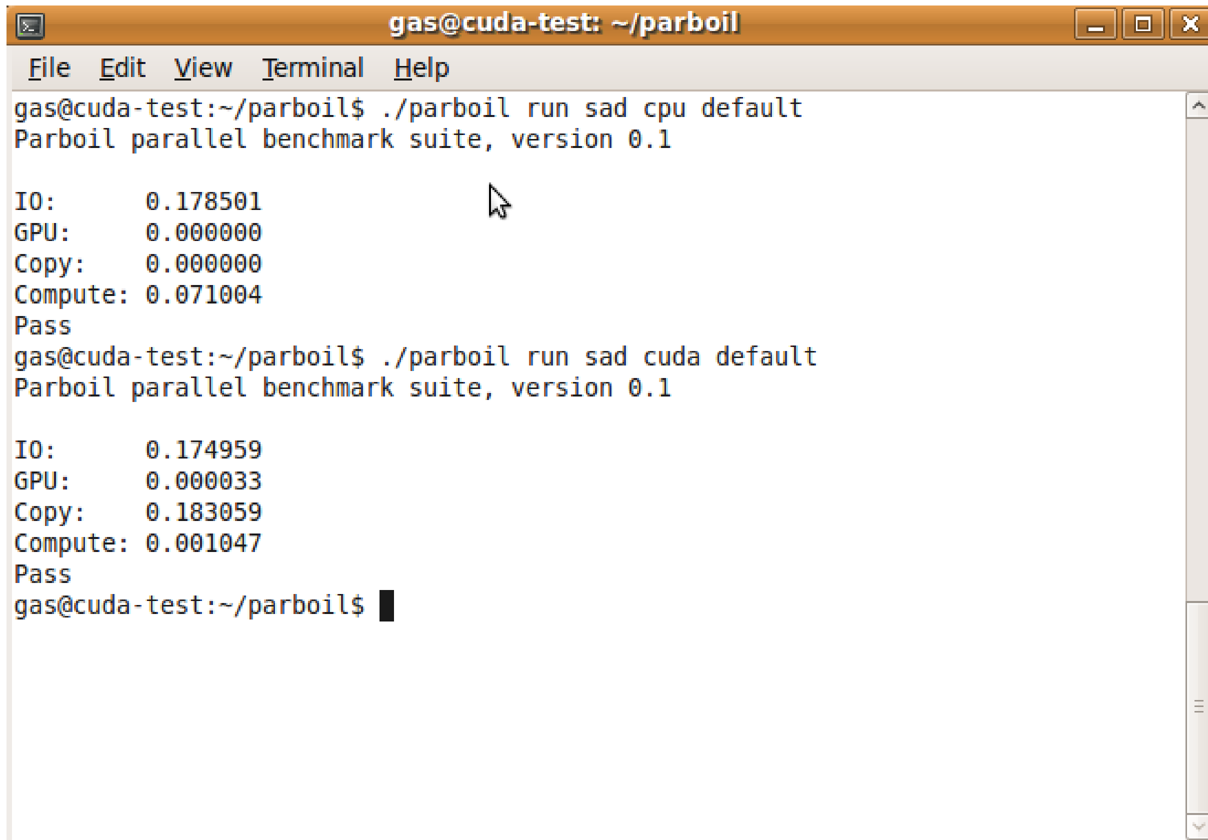
Tu je računska moč CUDA GPE vidna še bolj (Slika 43) kot v prejšnem testu, ker so faktorji zmogljivosti CPE/GPE še večji; 8400GS opravi delo kar 146-krat hitreje od Q6600, medtem ko je faktor med GTX295 in 6-jedrnikom skoraj 380, v dobro GPE. Kot že v prejšnjem primeru, bi ga z dobro implementacijo CPE algoritma (utilizacija vseh 12 jeder) lahko zmanjšali na 32, kar je še vedno veliko.



Slika 43: Občutna razlika že med prejšnjo in sedanjo generacijo štiri-jedrnikov.

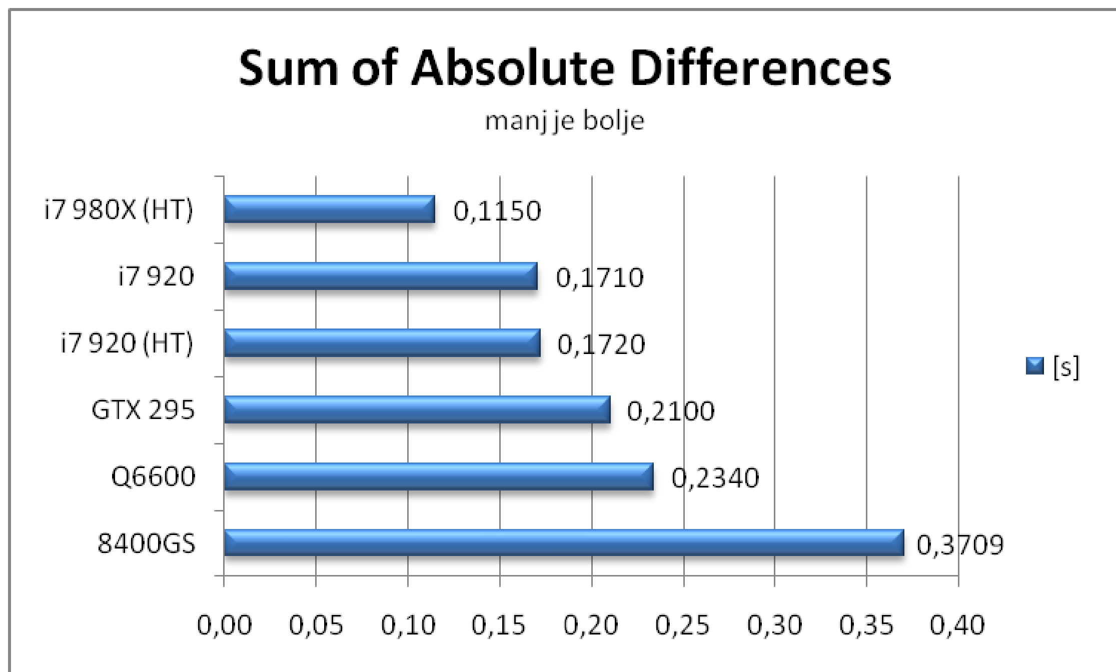
Zadnji test, Sum of Absolute Differences (SAD), pa predstavi zmogljivost procesorja na SAD algoritmu, ki je implementiran znotraj H.264 kodeka za kompresijo HD videa. Ta algoritem predstavlja najbolj procesorsko potraten del dekodiranja videa. Koščke trenutne slike videa (frame) primerja s koščki predhodne slike. Kot rezultat vrne najboljši približek obeh primerjav. Ta predstavlja prehod med posamezno sliko, ki naredi video bolj tekoč, poleg tega pa doseže tudi visoko kompresijo videa.

Za spremembo so rezultati presenetljivi. Prvič je CPE zmagovalec testa, kar gre na račun dobrega algoritma ter zakasnitev s strani prenašanja podatkov na GPE. Slika 44 prikazuje primerjavo med izvajanjem na CPE (Q6600) in GPE (8400GS).



Slika 44: Časi izvajanja SAD testa na CPE (prvi ukaz) in GPE (drugi ukaz).

Opazimo, da je čas računanja (*Compute*) na GPE v vseh primerih bistveno krajši od vseh ostalih časov. GPE največ časa porabi ravno za prenašanje podatkov iz CPE na GPE in obratno. Ker pa CPE prenaša le v svoj predpomnilnik oz. sistemski pomnilnik, so zato tam zakasnitve v primerjavi z GPE, minimalne. Če gledamo iz praktičnega stališča, nas zanimajo skupni časi. Če pa gledamo zgolj na čas obdelave, pa GPE še vedno prehiti CPE za skoraj 70-krat (8400GS vs Q6600). Slika 45 prikazuje končne rezultate računanja.



Slika 45: GPE porabi veliko časa za prenos podatkov.

Za razliko od prejšnjih testov lahko na tem mestu zatrdimo, da enkrat dražji i7 980X dejansko opravi obdelavo enkrat hitreje v primerjavi z GTX 295.

8. Zaključek

V diplomski nalogi smo si najprej ogledali delovanje grafične karte s strojnega in programerskega vidika. Skozi arhitekturo GPE smo se seznanili z njeno veliko zmogljivostjo, potrebno za izvajanje kompleksnih grafičnih operacij. V ta namen smo si ogledali tehnike mehčanja robov, ki veljajo za procesorsko najbolj zahtevne operacije. Sistem vzporedne vezave, SLI, nas je prepričal s svojo efektivnostjo, ki pa je zelo odvisna od programske implementacije in podpore izvajanja na sistemih z dvema in več GPE. Sistem vezave dveh GPE prinese največ zmogljivosti glede na ceno, medtem ko se 3-Way SLI in QuadSLI še uveljavljata, tako med kupci kot tudi med razvijalci (problem izrabe zmogljivosti).

V nadaljevanju smo se usmerili v programerske vode in spoznali osnove okolja CUDA, ki je trenutno najbolj razširjen in najbolj podprt API vmesnik za programiranje grafičnih procesorjev za splošne namene (GPGPU). Ker je CUDA last NVidie, je izvajanje teh aplikacij omejeno le na njihove grafične procesorje. NVidiin najresnejši konkurent ATI ima svoj GPGPU API vmesnik, imenovan Brook, razvit že nekaj časa (celo pred CUDA), na žalost pa (še) ni tako izpopolnjen ter razširjen med širšo množico. Po drugi strani se odprti standard OpenCL že nekaj časa trudi vzpostaviti enoten API za programiranje na GPE, ki bi bil neodvisen od proizvajalca grafičnih procesorjev. Vsekakor mu »problem« predstavlja ravno CUDA, ki je zelo dobro sprejeta med množico uporabnikov in razvijalcev programske opreme.
Osnova programskega modela CUDA je C / C++, zato izkušenim C programerjem ne bo povzročala veliko težav. Če hočemo doseči čim boljšo efektivnost algoritma, nam bo največ časa vzela optimizacija pomnilniških sredstev, izbira pravega tipa pomnilnika in določanje ustreznega števila blokov ter njihove velikosti.
Na primeru CUDA programa smo si ogledali deklaracijo kernel funkcije, upravljanje s pomnilnikom, potek prenosa podatkov iz CPE v GPE, obdelavo podatkov na GPE in prenos rezultatov nazaj v pomnilnik CPE.

Cilj diplomske naloge je bila analiza zmogljivosti CPE in GPE, s katero smo dokazali moč GPE na področjih, specifičnih za CPE. Problem tovrstne analize je predvsem v implementaciji algoritmov na obeh platformah. Ker uradnih benchmark programov za primerjavo CPE in GPE še ni, se je potrebno znajti z bolj eksperimentalnimi programi. Najboljši primer je zagotovo iskanje MD5 ključa, kjer se je BarsWF izkazal z odlično podporo več-jedrnim CPE in tudi z izpopolnjeno SLI implementacijo. Razlika med najhitrejšima predstavnikoma CPE in GPE je le štirikratna. Kaj pomeni slaba implementacija algoritma, nam prikaže Parboil-ov test CP, kjer je razlika več kot 370-kratna! Če bi bil algoritem enako učinkovit tudi na preostalih 11 jedrih, bi 6-jedrnik izračunal rezultat le 32-krat počasneje, kar bi bilo še vedno veliko.

GPE je zelo učinkovit pri paralelnem procesiranju, učinkovitost CPE pa je predvsem odvisna od kakovosti algoritma. V GPE enostavno »namečemo« čim več opravil, in s tem spravimo utilizacijo vseh jeder na čim višji nivo. Pri CPE je ta postopek bolj zapleten, ker imamo opravka z le nekaj jedri, zato mora biti njihova učinkovitost na kar se da visokem nivoju. To pa pomeni rabo kompleksnejših ukazov ter izdelavo kompleksnejših algoritmov, kar posledično vpliva na čas razvoja aplikacije.

Menim, da se bo v bližnji prihodnosti vse več algoritmov in s tem aplikacij preselilo na GPE, in s tem povzročilo preobrat na področju opravljanja osnovnih procesorskih funkcij. Vse bo odvisno od razvoja arhitekture CPE, ki se prav tako udejstvuje v paralelnem svetu. Hitrosti ne bodo šle dosti višje, saj frekvenca CPE ure že nekaj časa stagnira. Prihodnost je v vzporednem izvajanju več niti naenkrat (SIMT), kar je pred tremi leti dokazal tudi Intel s prototipom 80-jedrnega procesorja. Povsem možna je združitev CPE in GPE v nov procesor za splošne namene, na kar delno nakazuje že najnovejša Intelova arhitektura *Clarkdale*, ki združuje 32nm CPE jedro in 45nm GPE jedro v enem čipu.

Priloge

Compute Capability

Model	# SMjev	Različica
GeForce GTX 295	2x30	1.3
GeForce GTX 285, GTX 280	30	1.3
GeForce GTX 260	24	1.3
GeForce 9800 GX2	2x16	1.3
GeForce GTS 250, GTS 150, 9800 GTX, 9800 GTX+, 8800 GTS 512	16	1.1
GeForce 8800 Ultra, 8800 GTX	16	1.0
GeForce 9800 GT, 8800 GT, GTX 280M, 9800M GTX	14	1.1
GeForce GT 130, 9600 GSO, 8800 GS, 8800M GTX, GTX 260M, 9800M GT	12	1.1
GeForce 8800 GTS	12	1.0
GeForce 9600 GT, 8800M GTS, 9800M GTS	8	1.1
GeForce 9700M GT	6	1.1
GeForce GT 120, 9500 GT, 8600 GTS, 8600 GT, 9700M GT, 9650M GS, 9600M GT, 9600M GS, 9500M GS, 8700M GT, 8600M GT, 8600M GS	4	1.1
GeForce G100, 8500 GT, 8400 GS, 8400M GT, 9500M G, 9300M G, 8400M GS, 9400 mGPU, 9300 mGPU, 8300 mGPU, 8200 mGPU, 8100 mGPU	2	1.1
GeForce 9300M GS, 9200M GS, 9100M G, 8400M G	1	1.1
Tesla S1070	4x30	1.3
Tesla C1060	30	1.3
Tesla S870	4x16	1.0
Tesla D870	2x16	1.0
Tesla C870	16	1.0
Quadro Plex 2200 D2	2x30	1.3

Quadro Plex 2100 D4	4x14	1.1
Quadro Plex 2100 Model S4	4x16	1.0
Quadro Plex 1000 Model IV	2x16	1.0
Quadro FX 5800	30	1.3
Quadro FX 4800	24	1.3
Quadro FX 4700 X2	2x14	1.1
Quadro FX 3700M	16	1.1
Quadro FX 5600	16	1.0
Quadro FX 3700	14	1.1
Quadro FX 3600M	12	1.1
Quadro FX 4600	12	1.0
Quadro FX 2700M	6	1.1
Quadro FX 1700, FX 570, NVS 320M, FX 1700M, FX 1600M, FX 770M, FX 570M	4	1.1
Quadro FX 370, NVS 290, NVS 140M, NVS 135M, FX 360M	2	1.1
Quadro FX 370M, NVS 130M	1	1.1

Tabela 8: CUDA podprte GPE.

Programska koda

```
#include <stdio.h>
#include <cuda.h>

// Kernel funkcija, ki se izvede na GPE
__global__ void kvadriraj(float *a, int N)
{
	// izracun indeksa v polju elementov
	int idx = blockIdx.x * blockDim.x + threadIdx.x;
	// kvadriranje vrednosti elementa
	if (idx<N) a[idx] = a[idx] * a[idx];
}
// Metoda, ki se izvede preko CPE
int main(void)
{
	// kazalca na polja CPE in GPE
	float *A_h, *A_d, b;
	// stevilo elementov znotraj polja
	const int N = 10;
	size_t size = N * sizeof(float);
	// rezervacija pomnilnika CPE
	A_h = (float *)malloc(size);
	// rezervacija pomnilnika GPE
	cudaMalloc((void **) &A_d, size);
	// dodelitev vrednosti elementom
	for (int i=0; i<N; i++) A_h[i] = (float)i;
	// prenos podatkov iz CPE v GPE
	cudaMemcpy(A_d, A_h, size, cudaMemcpyHostToDevice);
	// stevilo niti znotraj bloka
	int block_size = 4;
	// koliko blokov potrebujemo za N elementov
	int n_blocks = N/block_size + (N%block_size == 0 ? 0:1);
	// klic kernel funkcije
	kvadriraj <<< n_blocks, block_size >>> (A_d, N);
	// prenos rezultatov v pomnilnik CPE
	cudaMemcpy(A_h, A_d, sizeof(float)*N, cudaMemcpyDeviceToHost);
	// izpis rezultatov
	for (int i=0; i<N; i++) printf("%d.element: %.2f\n", i, A_h[i]);
	// sprostitev CPE in GPE pomnilnika
	free(A_h); cudaFree(A_d);
}
```

Kazalo slik

Slika 1: Teoretična primerjava zmogljivosti CPE in GPE [GFLOP/s]. 4
Slika 2: Primerjava pasovne širine pomnilnika CPE in GPE [GB/s]. 5
Slika 3: GPE ima na voljo več aritmetično-logičnih enot za obdelavo podatkov. 6
Slika 4: GeForce GTX 295, sestavljen iz dveh tiskanin (*DualPCB*)... 9
Slika 5: Manj pregrevanja, nižji stroški izdelave. GTX 295 *SinglePCB*. 10
Slika 6: Arhitektura (SPA) grafičnega čipa GT200(b). 11
Slika 7: Arhitektura grafičnega čipa GF100. 12
Slika 8: Voodoo2 pospeševalnika v SLI načinu. 14
Slika 9: QuadSLI. Za debele denarnice. 15
Slika 10: Od desne proti levi: kronološki pregled nad SLI mostički... 17
Slika 11: Poenostavljeno prikazana tehnika mehčanja robov. 18
Slika 12: Gigabyte GV-3D1, prvi SLI sistem na eni tiskanini. Veliko denarja in veliko težav.... 19
Slika 13: Primerjava med prvo uradno dvoprocesorsko karto 7900GX2 (zgoraj) in njeno naslednico 7950GX2 (spodaj). 20
Slika 14: 3-Way SLI v praksi 21
Slika 15: Quadro Plex, za profesionalno rabo. 23
Slika 16: štirikratni SuperSampling.. 24
Slika 17: Računanje vrednosti barve. 24
Slika 18: FSAA je s stališča procesorske moči potraten. 25
Slika 19: MSAA gleda le na Z vrednost.. 25
Slika 20: MSAA za razliko od FSAA obdela le pike, ki so potrebne mehčanja. 26
Slika 21: "Navadni" način izračuna barvo iz štirih, "kvalitetni" pa iz osmih vzorcev. 27
Slika 22: Organiziranost niti znotraj blokov mreže.. 30
Slika 23: Učinkovita izraba množice niti v primerjavi s CPE. 31
Slika 24: Hierarhija pomnilnika. 32
Slika 25: Tipi pomnilnika CUDA naprave ter dostopi do njega. 35
Slika 26: Primer brezkonfliktne (levo) ter konfliktne (desno) situacije dostopa.......................... 38
Slika 27: Rezultat na standardnem izhodu v ukazni konzoli... 45
Slika 28: Vnos poti do CUDA knjižnic. 46
Slika 29: Vnos knjižnice za CUDA Runtime. 47
Slika 30: Vklop CUDA pravil za pravilno prevajanje aplikacije s strani VS2008. 48
Slika 31: Vklop dodatnih stikal za prevajanje. 49
Slika 32: S pravilno označeno kodo je lažje programirati. 50
Slika 33: Vodno hlajen PC6 295. 53
Slika 34: Vseh 64k delcev, zajetih v delčku sekunde. 55
Slika 35: HOOMD slabo izkorišča več jeder (Core i7 980X, HT vklopljen). 56
Slika 36: Velike razlike med CPE in GPE. 57
Slika 37: BarsWF CUDA, eden izmed najhitrejših iskalcev MD5 ključa.................................... 58
Slika 38: Štirikratna razlika med *GTX 295 OC* in *i7 980X*. 59
Slika 39: BarsWF SSE2 obremeni vseh 12 jeder (HT vključen). 60
Slika 40: Najbolj efektivno je višanje frekvenc SPjev. 61
Slika 41: Parboil znotraj Ubuntu terminala. 63
Slika 42: Parboil izkoristi le eno CPE jedro, kar je velik minus. 64
Slika 43: Občutna razlika že med prejšnjo in sedanjo generacijo štiri-jedrnikov. 65

Slika 44: Časi izvajanja SAD testa na CPE (prvi ukaz) in GPE (drugi ukaz). 66
Slika 45: GPE porabi veliko časa za prenos podatkov... 67

Kazalo tabel

Tabela 1: Uradne specifikacije modelov, osnovanih na čipu GT200b in GF100. 7
Tabela 2: Faktor CSAA načina lahko uporabnika zavede. 26
Tabela 3: Tipi pomnilnika GPE. 36
Tabela 4: Programske omejitve Compute Capability v1.3. 40
Tabela 5: Vgrajene spremenljivke za klic kernel funkcije.. 41
Tabela 6: Tehnične specifikacije testnih komponent. 52
Tabela 7: Uporabljena programska oprema. 54
Tabela 8: CUDA podprte GPE... 71

Viri

[1] (2010) MSDN Academic Alliance. Dostopno na: http://msdn.microsoft.com/en-us/academic/default.aspx
[2] (2010) CUDA Home. Dostopno na: http://www.NVidia.com/object/what_is_cuda_new.html
[3] (2010) GPGPU. Dostopno na: http://gpgpu.org/
[4] (2010) Hetergenous Computing. Dostopno na: http://en.wikipedia.org/wiki/Heterogenous_computing
[5] (2010) ATI Radeon R600. Dostopno na: http://en.wikipedia.org/wiki/Radeon_R600#Radeon_HD_3800
[6] (2010) GeForce 200 Series. Dostopno na: http://en.wikipedia.org/wiki/GeForce_200_Series
[7] (2008) NVIDIA's GTX 280: G80, Evolved. Dostopno na: http://arstechnica.com/hardware/reviews/2008/06/NVidia-geforce-gx2-review.ars
[8] (2010) NVIDIA's GF100: Architected for Gaming. Dostopno na: http://www.anandtech.com/show/2918
[9] (2010) Quad Data Rate SRAM. Dostopno na: http://en.wikipedia.org/wiki/Quad_Data_Rate_SRAM
[10] (2010) 3DFX. Dostopno na: http://en.wikipedia.org/wiki/3dfx#SLI
[11] (2010) NVidia Riva TNT. Dostopno na: http://en.wikipedia.org/wiki/RIVA_TNT
[12] (2010) Scan-Line Interleave. Dostopno na: http://en.wikipedia.org/wiki/Scan-Line_Interleave
[13] (2010) NVidia. Dostopno na: http://en.wikipedia.org/wiki/NVidia
[14] (2010) Scalable Link Interface. Dostopno na: http://en.wikipedia.org/wiki/Scalable_Link_Interface
[15] (2002) Artefacts in Image and Video Systems: Classification and Mitigation. Dostopno na: http://www-ist.massey.ac.nz/dbailey/sprg/pdfs/2002_IVCNZ_197.pdf
[16] (2010) Scalable Link Interface, Caveats. Dostopno na: http://en.wikipedia.org/wiki/Scalable_Link_Interface#Caveats
[17] (2010) Scalable Link Interface, Implementation. Dostopno na: http://en.wikipedia.org/wiki/Scalable_Link_Interface#Implementation
[18] (2008) GeForce 9800 GX2 Launch On March 11th. Dostopno na:

http://vr-zone.com/articles/geforce-9800-gx2-launch-on-march-11th/5560.html
[19] Nvidia Quadro Plex. Dostopno na: http://en.wikipedia.org/wiki/Quadro_Plex
[20] (2010) Anti-Aliasing. Dostopno na: http://en.wikipedia.org/wiki/FSAA#Full-scene_anti-aliasing
[21] (2001) Multi-Sampling Anti-Aliasing Explained. Dostopno na: http://www.firingsquad.com/hardware/multisamp/page3.asp
[22] (2009) Advanced Control Panel for nVidia Cards. Dostopno na: http://www.nhancer.com/?dat=d_AA
[23] NVidia CUDA Programming Guide 2.3, str. 7
[24] NVidia CUDA Programming Guide 2.3, str. 16
[25] NVidia CUDA Programming Guide 2.3, str. 18
[26] NVidia CUDA Programming Guide 2.3, str. 26
[27] NVidia CUDA Programming Guide 2.3, str. 37
[28] (2010) GPGPU Concepts and CUDA. Dostopno na: http://www.evl.uic.edu/aej/525/lecture05.html
[29] NVidia CUDA Programming Guide 2.3, str. 90
[30] NVidia CUDA Programming Guide 2.3, str. 98
[31] NVidia CUDA Programming Guide 2.3, str. 109
[32] NVidia CUDA Programming Guide 2.3, str. 118
[33] (2009) How to Enable Syntax Highlighting for CUDA files in Visual Studio 2005? Dostopno na: http://codereflect.com/2008/09/04/how-to-enable-syntax-highlighting-for-cuda-files-in-visual-studio-2005
[34] (2010) HOOMD-blue. Dostopno na: http://codeblue.umich.edu/hoomd-blue/index.html
[35] (2010) MD5. Dostopno na: http://en.wikipedia.org/wiki/MD5
[36] (2010) World Fastest MD5 cracker BarsWF. Dostopno na: http://3.14.by/en/md5
[37] (2010) Parboil Benchmark suite. Dostopno na: http://impact.crhc.illinois.edu/parboil.php
[38] (2010) Ubuntu Home. Dostopno na: http://www.ubuntu.com/
[39] (2010) Coulombov zakon. Dostopno na: http://sl.wikipedia.org/wiki/Coulombov_zakon
[40] NVidia CUDA Programming Guide 2.3, str. 114
[41] (2010) GeForce GTX 470 & 480 review. Dostopno na: http://www.guru3d.com/article/geforce-gtx-470-480-review/1