UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Ivan Surina

PRIMERJAVA GRAFIČNIH PROCESNIH ENOT IN CENTRALNIH PROCESNIH ENOT

DIPLOMSKO DELO NA VISOKOŠOLSKEM STROKOVNEM ŠTUDIJU

Mentor: mag. Igor Škraba

Ljubljana, 2009

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: 00443/2009
Datum: 05.04.2009

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

Kandidat: **IVAN SURINA**

Naslov: **PRIMERJAVA GRAFIČNIH PROCESNIH ENOT IN CENTRALNIH PROCESNIH ENOT**

COMPARISION OF GRAPHICS PROCESSING UNITS AND CENTRAL PROCESSING UNITS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Zmogljivost grafičnih procesorjev v zadnjih letih v primerjavi s centralnimi procesorji skokovito narašča. Opišite razlike v arhitekturi centralnih in grafičnih procesorjev na primeru dveh modernih procesorjev in preglejte možnosti uporabe grafičnih procesorjev pri izvajanju računsko zahtevnejših negrafičnih obdelav. Na testnem sistemu primerjajte zmogljivosti CPE in GPE pri množenju matrik.

Mentor:
pred. mag. Igor Škraba

Fakulteta za računalništvo in informatiko
2
UNIVERZA V LJUBLJANI

Dekan:
prof. dr. Franc Solina

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: 00443/2009
Datum: 05.04.2009

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

Kandidat: **IVAN SURINA**

Naslov: **PRIMERJAVA GRAFIČNIH PROCESNIH ENOT IN CENTRALNIH PROCESNIH ENOT**

COMPARISION OF GRAPHICS PROCESSING UNITS AND CENTRAL PROCESSING UNITS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija

Tematika naloge:

Zmogljivost grafičnih procesorjev v zadnjih letih v primerjavi s centralnimi procesorji skokovito narašča. Opišite razlike v arhitekturi centralnih in grafičnih procesorjev na primeru dveh modernih procesorjev in preglejte možnosti uporabe grafičnih procesorjev pri izvajanju računsko zahtevnejših negrafičnih obdelav. Na testnem sistemu primerjajte zmogljivosti CPE in GPE pri množenju matrik.

Mentor:
pred. mag. Igor Škraba

Dekan:
prof. dr. Franc Solina

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

diplomskega dela

Spodaj podpisani Ivan Surina,

z vpisno številko 63030294,

sem avtor diplomskega dela z naslovom:

PRIMERJAVA GRAFIČNIH PROCESNIH ENOT

IN CENTRALNIH PROCESNIH ENOT

S svojim podpisom zagotavljam, da:

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

  prof. mag. Igor Škraba

  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: 09.10.2009 Podpis avtorja:

Zahvala

Moj mentor, profesor mag. Igor Škraba, je zaradi svojega načina predavanja pri nas študentih vzbudil veliko zanimanja za snov, ki jo je poučeval. Že v prvem letniku sem pri predmetu osnove računalniške arhitekture ugotovil, da me zanima strojna oprema. Na koncu šolskega leta sem ravno zaradi tega razloga izbral študijsko smer logika in sistemi. Tudi za diplomsko nalogo sem izbral temo, ki je usmerjena v raziskovanje računalniške arhitekture.

Mentorju bi se rad zahvalil za ideje in nasvete. Omogočil mi je, da sem lahko nekaj tednov v laboratoriju raziskoval področje, o katerem sem pisal. Na voljo sem imel tudi strojno opremo, na kateri sem izvajal teste in meritve. Hvala tudi Damjanu Šoncu, ki mi je bil v laboratoriju vedno na voljo za nasvete in mi je s svojim strokovnim znanjem tudi precej pomagal.

Zahvaljujem se tudi dobrim prijateljem in sošolcem, ki so me v času študija spodbujali k rednemu obiskovanju predavanj in skupnemu učenju za izpite.

Predvsem pa bi se rad zahvalil svoji družini, ki me je v času študija podpirala in mi omogočila, da sem prišel do tega, kar imam.

Kazalo

Seznam kratic

CPE – Centralna Procesna Enota

GPE – Grafična Procesna Enota

GPGPU – General-Purpose computation on Graphics Processing Units

CUDA – Compute Unified Device Architecture

SLI – Scalable Link Interface

HD – High Definition

PC – Program Counter

SIMD – Single Instruction Multiple Data

SIMT – Single Instruction Multiple Thread

API – Application programming interface

SP – Streaming Processor

SM – Streaming Multiprocessor

ROP – Raster Operation Processor

MFLOPS – Mega Floating Point Operations Per Second

NVCC – NVIDIA C Compiler

1. Povzetek

Procesorji na grafičnih karticah postajajo vedno hitrejši. V primerjavi s centralnimi procesorji se računska moč skokovito povečuje, zato se vedno pogosteje porajajo ideje, da bi jih bilo smiselno uporabiti za poganjanje negrafičnih aplikacij. Kadar ne igramo iger, so bolj ali manj neobremenjeni, to pa je največkrat večino časa. Vedno bolj se širi zanimanje za izkoriščanje te ogromne računske moči. Tukaj je prostor za razvoj paralelnega procesiranja ter za sodelovanje grafičnih procesorjev s centralnimi.

Bolj kot sem raziskoval to področje, bolj zanimivo mi je postajalo. Izkoriščanje grafičnih procesorjev za splošno uporabo (GPGPU) je relativno novo področje, ki se elo hitro razvija in je zanimivo za širše množice.

V tem delu sem poskušal predstaviti ozadje in razvoj zgodovine tako grafičnih kot centralnih procesorjev. Predstavil sem arhitekturo na splošnih in tudi na konkretnih primerih, torej na procesorjih, ki se uporabljajo v današnjih osebnih računalnikih.

Celoto sem zaokrožil z meritvami izvajalnega časa programa za množenje matrik. Za primer sem vzel Intelov procesor Core 2 Duo E7400 in grafično kartico NVIDIA GTX260. Program se je izvajal na centralni procesni enoti (CPE) in na grafični procesni enoti (GPE), kjer so bile do 300-kratne pohitritve. Uporabil sem prosto dostopno orodje CUDA, ki ga je razvila NVIDIA. CUDA je okolje, ki temelji na programskem jeziku C. Z njegovo uporabo lahko izkoristimo računsko moč GPE in rešujemo kompleksne računsko intenzivne probleme. Orodje se lahko integrira z Microsoft Visual Studiom in je enostavno za uporabo.

Ključne besede: Paralelno procesiranje, CPE, GPE, GPGPU, CUDA, množenje matrik.

2. Abstract

Graphic processors are becoming faster and faster. Computational power within graphic processing units (GPUs) is growing rapidly compared to central processing units (CPUs). Usage of this power is becoming very interesting in many areas. Programmers try to use this power. They are developing new algorithms for non-graphic applications. When we do not play games, GPUs are idle and this is most of the time. Parallel processing algorithms which exploit both GPUs and CPUs takes place here.

More I was researching this area, more interesting it was getting. General purpose computing on graphic processors (GPGPU) is relatively new and it is available to everyone. This is the main reason why it is developing so fast.

In this thesis I tried to represent background and developing of both graphic and main processors through time. I presented architecture on general examples and on specific processors which are widely used in personal computers.

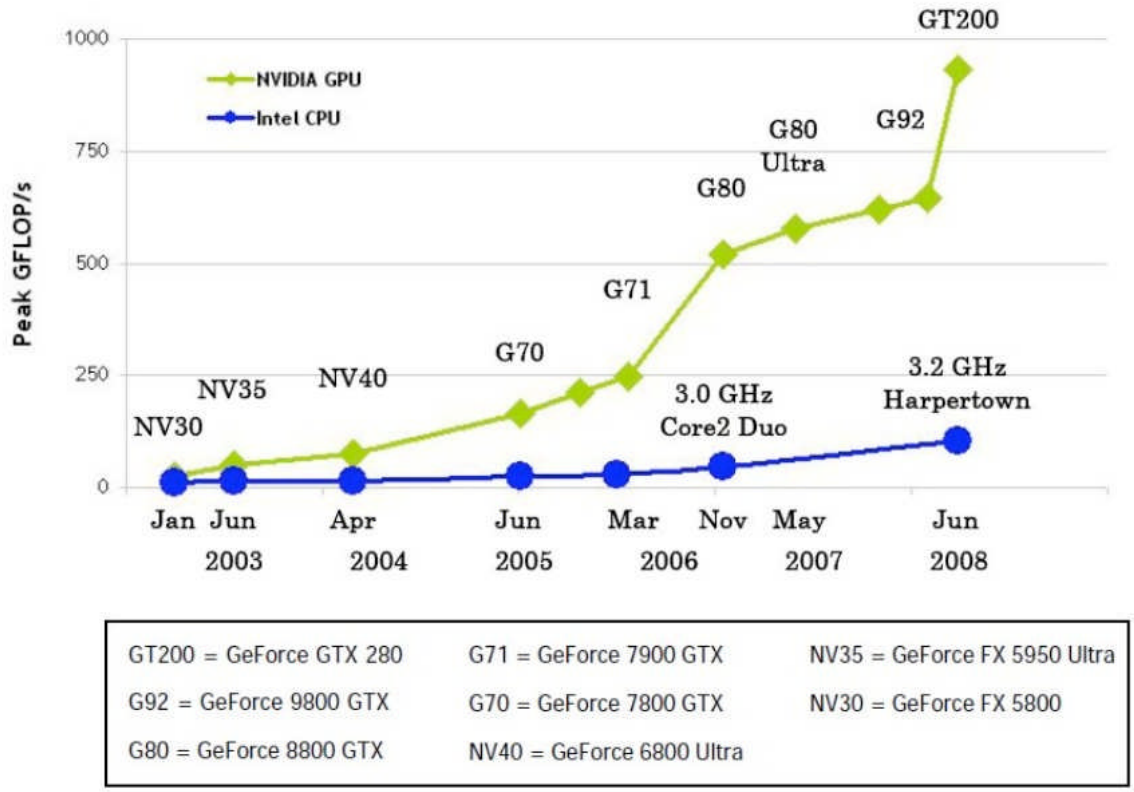
I brought this theme to the close with analyzing execution of matrix multiplication program. I measured time needed for execution of the program on CPU and on the GPU. As example I used Intel's Core 2 Duo processor E7400 and NVIDIA's graphic card GTX260. Speedups in applications were up to 300 times on GPU. I worked with NVIDIA’s environment CUDA, based on C programming language. With CUDA, it is possible to unlock the processing power of the GPU to solve complex compute-intensive problems. Environment is easy to integrate with Microsoft Visual Studio and is easy to use.

Keywords: Parallel processing, CPU, GPU, GPGPU, CUDA, matrix multiplication.

3. Uvod

Ko sem izbiral temo za diplomsko nalogo, sem poskušal najti zanimivo in uporabno področje, ki mi je poznano, a bi ga rad še bolj raziskal. Tako sem prišel do zamisli, da opišem procesorsko arhitekturo grafičnih kartic in jo predstavim skupaj z arhitekturo računalnika kot celote. Primerjal sem centralne procesorje z grafičnimi. Omejil sem se na najbolj prodajane osebne računalnike, ki jih najdemo v številnih gospodinjstvih. Pri konkretnejših opisih sem se omejil na v današnjem času najbolj prodajane dvojedrne procesorje Core 2 in na nove modele grafičnih kartic NVIDIA.

ATI in NVIDIA sta vodilna proizvajalca grafičnih kartic in že vrsto let medsebojno tekmujeta za tržni delež. Mnenja glede proizvajalcev grafičnih kartic in procesorjev so s strani kupcev največkrat deljena. To pa zato, ker si večina ljudi mnenje ustvari na podlagi izkušenj. Tako se tudi pri nadaljnih nakupih največkrat odločajo na podlagi svojih preteklih izkušenj. Sam mislim, da je treba biti pri izbiri objektiven. Veliko testov in mnenj uporabnikov najdemo na internetu. Najbolj zanesljivi se mi zdijo testi priznanih revij ali spletnih strani, kjer najdemo tako opise specifikacij, kot tudi izmerjene rezultate pri različnih aplikacijah in za raznovrstno strojno opremo.



Slika 1: Primerjava zmogljivosti grafičnih procesorjev in centralnih procesorjev. Računska moč grafičnih kartic, izražena v GFLOP/s, se v primerjavi z močjo centralnih procesorjev v zadnjih letih skokovito povečuje.

Grafični procesorj na grafičnih karticah so se v primerjavi s centralnimi procesorji v preteklih letih razvijali precej hitreje. Na sliki 1 je razvidno razmerje računske moči procesorjev dveh vodilnih proizvajalcev grafičnih in centralnih procesorjev.

Bolj kot sem raziskoval to področje, bolj zanimivo je postajalo delo. Izkoriščanje grafičnih procesorjev za splošno uporabo je relativno novo, zanimivo je širšim množicam in zato se razvija z veliko hitrostjo. Računsko moč grafičnih kartic se izkorišča na številnih področjih: od statističnih obdelav, virtualizacije, vzorčenja, operacij nad matrikami ... Več primerov najdemo v poglavju 8.

Cilj tega dela je predstaviti možnosti za razvoj aplikacij, ki izkoriščajo računsko moč grafičnih kartic. Zato sem na splošno opisal centralne in grafične procesorje in predstavil osnovno arhitekturo, kar pripomore k razumevanju nadaljnjega dela. Bolj podrobno sem opisal arhitekturo centralnih procesorjev Intel Core 2 ter grafičnih procesorjev NVIDIA serije GTX200. Razložil sem delovanje programa za množenje matrik, ki je tudi najboljši primer za predstavitev razlik v izvajalnih časih med centralnim in grafičnim procesorjem.

4. Centralna procesna enota

Procesor ali centralna procesna enota (CPE) je osrednji del računalnika, ki izvaja ukaze in nadzoruje ter upravlja ostale enote. Mikroprocesor je procesor, ki je realiziran na enem samem integriranem vezju (čipu). Danes to velja za skoraj vse procesorje, zato ponavadi predpono mikro izpustimo.

Hitrost procesorja je v največji meri odvisna od nabora ukazov, števila bitov, ki jih lahko procesor naenkrat obdela, in od frekvence njegove ure.

4.1. Zgodovina

Prvi komercialni procesor je bil 4-bitni Intel 4004, izdelan leta 1971. V tistem času so bili med vodilnimi tudi inženirji iz podjetja Texas Instruments. Razvijali so procesor TMS 1000 in leta 1973 patentirali mikroprocesor na enem čipu.

Intel je kasneje razvil prvi 8-bitni procesor z oznako 8008. Podjetje Motorola je istočasno razvilo procesor 6800 in kasneje 6502. Zaradi teh procesorjev so v 80-ih letih osebni računalniki s ceno pod 100 dolarji postali dostopni navadnim ljudem.

4.2. Lastnosti

Dve glavni lastnosti procesorja sta frekvenca njegove ure in število bitov, ki jih procesor obdela v eni operaciji. S frekvenco delovanja je povezano število operacij, ki jih procesor izvede vsako sekundo. Število bitov vpliva na naslovni prostor in omejuje velikost operandov za procesiranje.

Danes je število bitov v najbolj prodajanih procesorjih za osebne računalnike 32 ali 64, frekvenca pa se giblje od 800MHz (Intel Atom) do 3.33GHz (Intel Core i7) in več. V novejšem času so zaradi doseganja večje procesorske moči začeli vgrajevati več jeder v en procesor. Tako so danes najbolj prodajani dvo- in štiri-jedrni procesorji, na tržišču pa so tudi šest-jedrni (Xeon) procesorji, kmalu lahko pričakujemo tudi osem- in šestnajst-jedrne.

Višje hitrosti povzročajo nestabilnost sistemov in pregrevanje, zato se je razvoj usmeril v realizacijo več jeder na enem čipu. To omogoča tudi boljšo večopravilnost. Na več procesorjih se lahko izvaja več aplikacij hkrati (vzporedno). Večopravilnost je možna tudi pri enojedrnih sistemih. Dosežemo jo s časovno delitvijo procesorskega časa, kar povzroča zakasnitve.

4.2.1. Zgradba

Glede na ukazno arhitekturo procesorje v grobem delimo na dve kategoriji: RISC in CISC. Procesorji tipa RISC (Reduced Instruction Set Computer) imajo majhno število preprostih ukazov, ki se izvršujejo približno v enakem času. Najbolj znani RISC procesorji so Alpha, ARM, PIC MIPS in SPARC. Procesorji tipa CISC (Complex Instruction Set Computer) pa

imajo velik nabor kompleksnejših ukazov, ki naredijo več operacij hkrati. Vodilna proizvajalca teh procesorjev sta AMD in Intel.

4.3. Centralni procesorji AMD

Advanced Micro Devices, Inc. – AMD je ameriško podjetje v Kaliforniji, ki razvija procesorje in z njimi povezane računalniške tehnologije za komercialni in potrošniški trg. Glavni produkti so mikroprocesorji, vezni nabori čipsetov na matičnih ploščah, vgrajeni procesorji in grafični procesorji. Poleg omenjenih AMD izdeluje tudi namenske procesorje, to so procesorji za rabo v telefonih, avtomobilih, igralnih konzolah in podobnih sistemih.

AMD je drugi največji izdelovalec mikroprocesorjev, takoj za Intelom in tretji največji izdelovalec grafičnih procesorjev, takoj za Intelom in Nvidio. AMD je bil ustanovljen leta 1969. Začeli so z izdelavo logičnih čipov in kasneje, leta 1975, razširili razvoj na čipe RAM. Pozneje so se usmerili na arhitekturo RISC in razvili procesor AMD 29K, še pozneje so začeli izdelovati grafične in zvočne procesorje ter pomnilnike EPROM in modeme. Najbolj uspešni so bili na področju procesorjev, zato so se usmerili samo na mikroprocesorje, kompatibilne Intelovim in na pomnilnike flash. Tako so postali konkurent Intelu z x86 kompatibilnimi procesorji.

Leta 2006 je AMD kupil podjetje ATI Technologies, ki se ukvarja z izdelavo grafičnih kartic in je sedaj del podjetja AMD. AMD tekmuje po eni strani z Nvidio, po drugi pa z Intelom.

4.3.1. Zgodovina

Leta 1982 je AMD podpisal pogodbo s podjetjem INTEL in postal drugi licenciran proizvajalec procesorjev 8086 in 8080. IBM je hotel uporabiti Intel 8080 v svojem IBM PC računalniku, vendar je moral po pogodbi zagotoviti vsaj dva različna proizvajalca za svoje čipe. Tako je AMD pozneje proizvajal procesorje AM286 preko iste pogodbe, vendar je Intel od te pogodbe leta 1986 odstopil in ni hotel razkriti tehnične dokumentacije za arhitekturo i386. AMD se je pritožil na odločitev Intela in tako se je začel pravni spor, ki se je leta 1994 končal v prid AMD-ju. Zaradi nejasnosti pri pravicah uporabe Intelovega mikroprograma so morali pri AMD-ju razviti svoje verzije mikroprograma. Uporabljali so reverse engineering, to je izdelavo nekakšne kopije naprave, v tem primeru procesorja. Delovanje čipa so analizirali in poskušali narediti čip z enako funkcionalnostjo, a svojim mikroprogramom.

Tako so leta 1991 predstavili AM386, kopijo Intelovega procesorja 386. V manj kot letu dni je podjetje prodalo milijon teh procesorjev. Pozneje so razvili procesor AM486, ki je bil uporabljen v številnih originalno opremljenih računalnikih, med njimi tudi v Compaqovih. Ti procesorji so se izkazali za zelo uspešne. AM5x86 je nadaljeval razvoj pri nizko-cenovnih alternativah. Razvoj procesorjev je pospešeno napredoval in zato je bila reverse engineering strategija vse manj zanimiva za uporabo, saj jo je bilo vse težje implementirati.

Leta 1996 je AMD predstavil procesor K5, leto pozneje pa še K6. Leta 1999 so razvili procesor K7 sedme generacije procesorjev x86, poimenovali so ga Athlon. Pozneje so se iz tega razvili procesorji Athlon XP. Procesor K8 je temeljil na arhitekturi K7, s to razliko, da so dodali 64-bitne ukaze. Razvili so tudi HyperTransport tehnologijo za visoko zmogljive povezave točka v točko (point-to-point). Tehnologijo so uporabili v procesorjih Opteron,

namenjenih za strežniške sisteme. Pozneje so to tehnologijo uporabili tudi za procesorje za namizne računalnike – Athlon 64. Leta 2005 so predstavili prva dvojedrna procesorja Opteron in Athlon 64 X2. Naziv Athlon 64 X2 so zamenjali z Athlon X2. Leta 2008 je AMD razvil tudi dvojedrne procesorje Sempron, ki imajo za razliko od Athlonov manj predpomnilnika L2 in počasnejše vodilo HyperTransport.

Leta 2007 so razvili štiri-jedrne procesorje, ki so jih najprej uporabili za strežniške sisteme (Opteron), pozneje pa še za namizne računalnike (Phenom). Procesorji Phenom imajo tri ali štiri jedra na enem čipu. Danes imamo drugo generacijo teh procesorjev, ki nosijo oznako Phenom II. Leta 2007 so predstavili procesorje Turion 64 X2, narejene v 65nm tehnologiji. Namenjeni so uporabi v prenosnih računalnikh in imajo zelo nizko porabo. So odziv na Intelove procesorje Pentium M, Core ter Core 2.

4.4. Centralni procesorji INTEL

Podjetje je bilo ustanovljeno leta 1968 kot Integrated Electronics Corporation v Kaliforniji. Prvi komercialni mikroprocesor so izdelali leta 1971 in danes je proizvodnja mikroprocesorjev glavni poslovni segment tega podjetja. Izdelali so prvi mikroprocesor z arhitekturo x86, kakršne najdemo v večini današnjih računalnikov. Intel izdeluje tudi vezne nabore čipov za matične plošče, mrežne vmesnike, pomnilnike flash, grafične kartice in druge naprave, povezane s komunikacijami in računalništvom.

Intel je po letu 1990 postal vodilni proizvajalec procesorjev za osebne računalnike in ima danes 80% tržni delež. Podjetje je veliko vlagalo v razvoj novih mikroprocesorjev. To je pozitivno vplivalo na rast prodaje osebnih računalnikov po celem svetu, saj so le-ti postali dostopnejši vsakemu posamezniku.

4.4.1. Arhitektura Core

Core 2 označuje Intelove eno-, dvo- in štiri-jedrne procesorje, izdelane v 64-bitni arhitekturi. Eno- in dvo-jedrni modeli so na enem vezju, medtem ko so štirijedrni modeli na dveh vezjih, od katerih vsako vsebuje dve jedri, obe vezji pa sta realizirani na enem čipu. Procesorje Core 2 so največ uporablja pri namiznih in prenosnih računalnikih. Nadomestili so procesorje Pentium 4, Pentium D in Pentium M.

Mikroarhitektura Core je ponovno uporabila nižje frekvence ure a izboljšano uporabo urinih ciklov in moči v primerjavi z NetBurst tehnologijo, uporabljeno pri procesorjih Pentium 4, Pentium D. Arhitektura Core zagotavlja bolj učinkovite načine dekodiranja ukazov, izvajanja ukazov, uporabo predpomnilnika in vodil. Zmanjšana je poraba energije ob povečanju procesorske moči. Procesorji Core 2 s porabo moči 65W so dvakrat bolj učinkoviti od procesorjev Pentium D, ki imajo porabo moči 130W.

Procesorji Core 2 imajo več novih tehnologij, ki jih ločijo od predhodnikov. Nekatere izmed njih so:
- Virtualization Technology
- Execute Disable Bit
- SSE3

Poleg naštetega so predstavili tudi nove ukaze SSSE3, Trusted Execution Technology, Enhanced SpeedStep in Active Management Technology (iAMT2).

4.4.2. Procesorji Core i7

Naslednik procesorjev Core 2 je Core i7, ki temelji na arhitekturi Nehalem. Core i7 je bil predstavljen leta 2008 kot družina treh štirijedrnih procesorskih modelov. Nehalem ima glede na mikroarhitekturo Core številne izboljšave. Vodilo FSB je nadomeščeno z vodilom QuickPaht in procesor ima vgrajen krmilnik za komunikacijo z glavnim pomnilnikom, ki se tako ne nahaja več na posebnem čipu na matični plošči. Vsako od štirih jeder podpira tehnologijo HyperThreading, ki vsakemu jedru dodeljuje niti procesov in jih tako bolj enakomerno izkorišča. Vsako od štirih jeder je razdeljeno na dve navidezni jedri. Operacijski sistem vidi s svojega stališča osem jeder. Tehnologija izdelave je 32nm, leta 2011 pa bo Intel predstavil procesorje, ki bodo izdelani v 22nm tehnologiji.

Procesorji Core i7 imajo podnožje LGA1366, medtem ko imajo procesorji Core 2 podnožje LGA775. Številka označuje število nožic, s katerimi je procesor pritrjen na matično ploščo.

Za strežniške sisteme so na tržišču že šest-jedrni procesorji Xeon. Predvideni pa so tudi procesorji z osmimi jedri, po napovedih Intela bodo prišli na tržišče konec leta 2009.

5. Grafična procesna enota

Grafična procesna enota ali GPE (GPU - Graphics Processing Unit, včasih tudi VPU -Visual Processing Unit) je poseben procesor, namenjen prikazovanju grafike pri računalnikih, delovnih postajah in igralnih konzolah. Sodobne GPE so zelo učinkovite pri prikazovanju slike in pri delu z računalniško grafiko. Grafični procesor ima mnogo jeder. Visoka raven paralelizma jih za izračun grafičnih operacij naredi veliko bolj učinkovite od CPE. Pri obdelavi slike gre za veliko količino razmeroma preprostih računskih operacij. GPE se nahaja na grafični kartici, lahko pa je tudi integrirana na osnovni plošči. Več kot 90% osebnih računalnikov in prenosnikov, ki so danes na tržišču, ima integrirano GPE. Takšna izvedba je praviloma manj zmogljiva od GPE čipov, ki so na grafični kartici.

GPE izvršuje številne osnovne grafične operacije nad biti. Vse sodobne grafične kartice podpirajo 3D računalniško grafiko, ki tipično vključuje funkcije, povezane z obdelavo digitalnega videa.

Grafični procesorji so realizirani v obliki cevovodov (angleško pipeline). V enem urinem ciklu se s pomočjo cevovodov lahko obdela ena slikovna točka (angleško pixel). Sodobne grafične kartice imajo od 8 do 32 cevovodov, torej je vsak ukaz nad slikovno točko sestavljen iz od 8 do 32 operacij.

Veliko podjetij je izdelovalo grafične procesorje z različnimi poimenovanji. Danes so na tržišču najbolj razširjeni Intel s 50%, NVIDIA z 28% in AMD/ATI z 20% tržnim deležem. Številke vključujejo manj zmogljive rešitve grafičnih procesorjev proizvajalca Intel, ki so realizirani na osnovnih ploščah. Če ne upoštevamo teh integriranih grafičnih realizacij, potem NVIDIA in AMD/ATI predstavljata skoraj 100 odstotni delež celotnega tržišča. Ostala večja proizvajalca sta še Matrox (grafične kartice, namenjene profesionalni rabi) in VIA Technologies/S3 Graphics.

Grafične kartice pospešujejo 2D in 3D grafiko, strojno podpirajo funkcije za predvajanje digitalnega videa, sodobne GPE pa celo strojno dekodirajo HD video in tako razbremenijo CPE.

5.1. Grafični pospeševalniki

Grafični pospeševalniki so posebne naprave, namenjene izboljšanju grafičnega prikazovanja. Sliko ne prikazujejo samostojno, ampak so kot dopolnilo osnovni enoti, ki skrbi za grafično prikazovanje. Danes se jih ne uporablja pogosto. Grafični pospeševalniki imajo namensko narejene mikročipe. Ti čipi so namenjeni izvajanju takšnih matematičnih operacij, ki so pri izračunih slike najbolj pogosti. Učinkovitost mikročipa zato določa učinkovitost grafičnega pospeševalnika. Največ se jih uporablja za igranje 3D iger ali za visoko ločljivo 3D prikazovanje.

5.2. Računske funkcije GPE

Sodobne GPE uporabljajo večino tranzistorjev za izvajanje operacij, povezanih s 3D računalniško grafiko. Grafične kartice so bile prvotno zasnovane za doseganje boljših rezultatov pri operacijah, kjer je komunikacija s pomnilnikom intenzivna. Pri teh operacijah se naenkrat obdeluje veliko število podatkov, zato je pasovna širina komunikacije s pomnilnikom zelo pomembna. Takšne operacije so preslikave tekstur, senčenje likov ali geometrični izračuni, kot naprimer rotacije in premiki koordinatnih sistemov. Sodobni razvoj je poleg naštetega usmerjen še v izboljšave pri operacijah vzorčenja, antialiasinga in natančnih operacij z barvami. Antialiasing pri grafiki je postopek minimiziranja napak, ki nastanejo pri preslikavi iste slike iz ene resolucije v drugo.

Večina takih računskih operacij vključuje operacije z matrikami. Sodobne GPE so prilagojene za učinkovito izvajanje takšnih operacij, zato so sodobne GPE zanimive inženirjem in znanstvenikom tudi za uporabo pri ne-grafičnih obdelavah.

5.3. Oblike GPE

Bolj zmogljive GPE tipično komunicirajo z ostalimi enotami računalnika preko vodila PCI Express ali AGP (Accelerated Graphics Port), ki se nahaja na matični plošči. Tako je mogoče relativno enostavno zamenjati grafično kartico, seveda v primeru, ko osnovna plošča podpira drugo grafično kartico. Nekaj grafičnih kartic uporablja vodilo PCI (Peripheral Component Interconnect), vendar je prepustnost tega vodila v primerjavi z vodili PCIe in AGP manjša.

Tehnologiji SLI od NVIDIA in CrossFire od ATI omogočata povezovanje več grafičnih kartic za izrisovanje ene slike, kar poveča razpoložljivo računsko moč. Več o tem najdemo v nadaljevanju, na koncu poglavja 7.1.

Integrirane grafične kartice so grafični procesorji, ki uporabljajo glavni pomnilnik računalnika, in torej nimajo svojega pomnilnika. Takšne rešitve so cenejše, vendar so manj zmogljive. Današnje rešitve integriranih GPE so relativno zmogljive in z njimi lahko poganjamo 3D igre in grafično zahtevne programe, kot so Adobe Flash ali Adobe Photoshop. Številne osnovne plošče z integriranimi grafičnimi karticami imajo razširitveno mesto PCIe ali AGP in tako omogočajo poznejšo nadgradnjo sistema z vgradnjo grafične kartice na to vodilo. Takšna kartica ima svoj pomnilnik in je zato hitrejša. V primeru takšne nadgradnje sistem prikazuje sliko preko dodane kartice, integrirana grafična kartica pa ostane neuporabljena.

Glavni pomnilnik ima hitrost prenosa podatkov od 2Gbit/s do 12,8Gbit/s, medtem ko imajo grafične kartice pomnilnik s hitrostjo prenosa od 10Gbit/s do 100 Gbit/s in več, odvisno od modela grafične kartice. Integrirane GPE komunicirajo z glavnim pomnilnikom in za dostop do njega tekmujejo s CPE.

V splošnem velja, da je grafične kartice, ki so realizirane na posebni kartici, mogoče zamenjati, integriranih pa ni mogoče zamenjati. Obstajajo pa tudi grafične kartice, ki imajo vse značilnosti grafičnih kartic, ki so realizirane na posebni kartici, so pa fizično integrirane na matično ploščo. Takšne grafične kartice imajo torej svoj pomnilnik in so hitrejše od integriranih grafičnih kartic. Uporablja se jih predvsem pri prenosnikih. Logične povezave z

matično ploščo so enake kot pri vodilu AGP ali PCIe, niso pa to standardni priključki AGP oziroma PCIe. Takšnih kartic ni mogoče zamenjati.

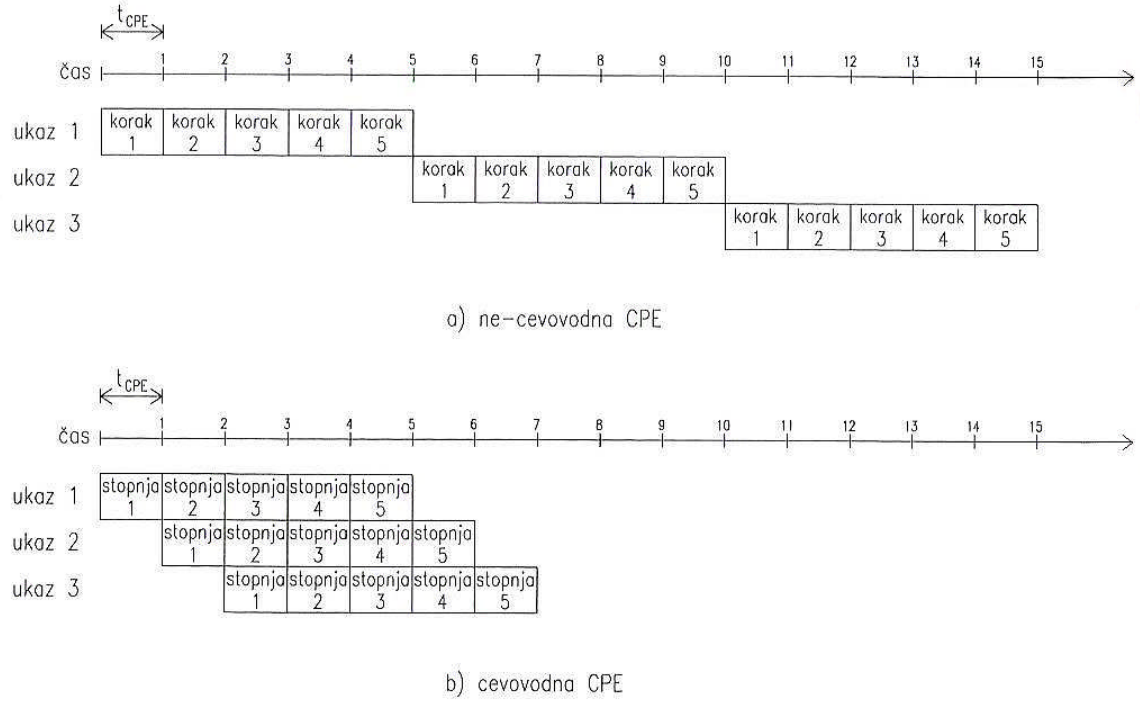
Danes imamo na trgu tudi hibridne grafične kartice, ki tekmujejo z integriranimi. Te imajo nekaj svojega pomnilnika, ki je zelo hiter, če pa potrebujejo več pomnilnika, si sposodijo glavni pomnilnik, ki si ga delijo s CPE. Takšne kartice se uporablja predvsem pri prenosnih računalnikih. Najbolj razširjene implementacije so ATI HyperMemory in NVIDIA TurboCache. Hibridne grafične kartice so nekoliko dražje, toda precej hitrejše od integriranih.

6. Arhitektura centralne procesne enote

Arhitektura vsakega računalnika je v največji meri določena s številom in vrsto strojnih ukazov. Na zmogljivost računalnika najbolj vpliva CPE, ki izvršuje ukaze. Poleg CPE imamo v računalniku tudi druge procesorje, ki niso centralni. Uporabljeni so v vhodno/izhodnih enotah računalnika in skrbijo za delovanje teh enot.

Procesorji so danes zgrajeni iz digitalnih elektronskih vezij in so najpogosteje realizirani na enem čipu, to so mikroprocesorji. Ker so danes skoraj vsi procesorji mikroprocesorji, predpono mikro ponavadi izpustimo.

Zaporedno izvajanje ukazov je osnovna značilnost von Neumannovega modela računalnika. To pa je hkrati tudi njegova omejitev. Večjo zmogljivost CPE dosežemo z izvajanjem več funkcij istočasno ali paralelno. Treba je določiti, katere funkcije se lahko izvršujejo hkrati, to pa ni preprosto. Eno možnost predstavlja drugačno programiranje, pri katerem da odgovor programer. Pri takem pristopu morajo biti programi prilagojeni za CPE. To pomeni ponovno prevajanje programov in spremembe v programih za določene CPE. Rešitev tega problema lahko dosežemo z vzporednim izvrševanjem ukazov, ki so med seboj neodvisni. To je paralelizem na nivoju ukazov. Najpogosteje uporabljani način za njegovo izkoriščanje je cevovod (pipeline).

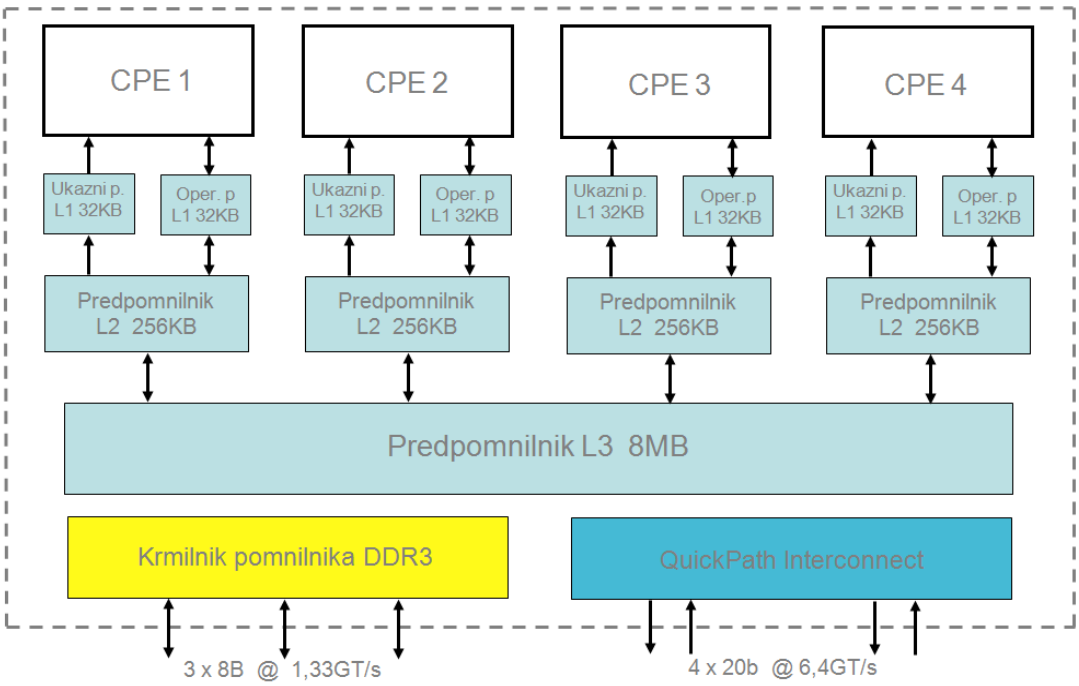


Slika 2: Izvrševanje ukazov pri ne-cevovodni in cevovodni CPE.

Z besedo cevovod se označuje način realizacije CPE, pri katerem se naenkrat izvršuje več ukazov tako, da se posamezni koraki izvrševanja prekrivajo. Cevovod izvršuje ukaze na način, ki je podoben tekočemu traku pri proizvodnji avtomobilov in drugih izdelkov. Posamezne operacije se razdelijo na manjše dele ali podoperacije. Vsako od podoperacij

opravi točno določen del cevovoda, ki mu pravimo stopnja cevovoda (pipeline stage) ali segment cevovoda. Stopnje so povezane tako, da tvorijo nekakšno cev, skozi katero potujejo ukazi. Na sliki 2 vidimo razliko pri izvrševanju ukazov pri ne-cevovodni in cevovodni CPE. Zaradi enostavnosti smo vzeli, da uporabljata obe CPE enake podoperacije in izvrševanje vsake traja eno urino periodo. Čeprav je trajanje izvrševanja vsakega ukaza enako pri obeh CPE, se pri cevovodni v danem času izvrši n-krat več ukazov (n je število stopenj cevovoda). Pri resničnih CPE so zaradi shranjevanja vmesnih rezultatov, začetnih zakasnitev in cevovodnih nevarnosti pohitritve nekoliko manjše.

Paralelizem lahko dosežemo tudi na druge načine, naslednji ki ga bom opisal, je paralelizem niti. Nit lahko opišemo kot zaporedje ukazov in operandov, ki se lahko izvršuje neodvisno od morebitnih drugih ukazov in operandov. Nit je lahko del programa, ki ga sestavlja več procesov, lahko pa je tudi samostojen program. Razlikovanje med nitmi in procesi ali opravili (task) srečujemo pri operacijskih sistemih, kjer so definicije pogosto neenotne. Za večnitno procesiranje je pomembno samo, da je nit definirana in zapisana. Z izrazom večnitnost (multithreading) se označuje način delovanja, pri katerem si več niti deli funkcijske enote enega procesorja. Tako delovanje zahteva, da ima vsaka nit v procesorju svoje stanje, ki je ločeno in neodvisno od stanj drugih niti. Stanje niti vsebuje svojo kopijo registrov, svoj programski števec, svoje tabele strani in del programsko nevidnih registrov. Vse niti pa si delijo predpomnilnik in glavni pomnilnik. Vsaka nit vidi, kot da ima celoten procesor na razpolago sama – temu pravimo, da je en fizični procesor videti kot več logičnih procesorjev [2].



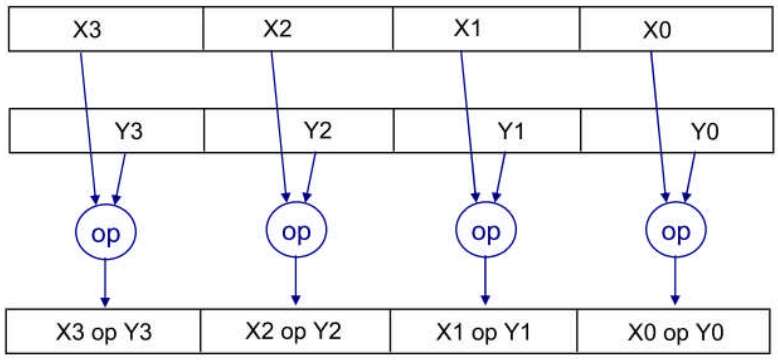
Slika 3: Zgradba 4-jedrnega procesorja Intel Core i7 (Nehalem)

Naslednji korak k izkoriščanju paralelnosti na nivoju niti so tako imenovani večjedrni procesorji (multicore). To so procesorji, ki imajo na istem čipu več neodvisnih jeder. Tipično ima vsako jedro svoj predpomnilnik nivoja 1 (L1), medtem ko si predpomnik nivoja 2 (L2) in višje nivoje delijo. Pri nitih, ki se izvršujejo vsaka na svojem jedru, imamo pravo paralelnost, čeprav je zaradi skupnega pomnilnika in predpomnilnika medsebojni vpliv še prisoten.

Pri večini današnjih večjedrnih CPE je vsako jedro tudi večnitno. Na sliki 3 vidimo zgradbo Intelovega procesorja Core i7 s 4 jedri, od katerih je vsako 2-nitno in tako se lahko hkrati izvršuje 8 niti. Operacijski sistem vidi ta procesor kot 8-jedrno CPE.

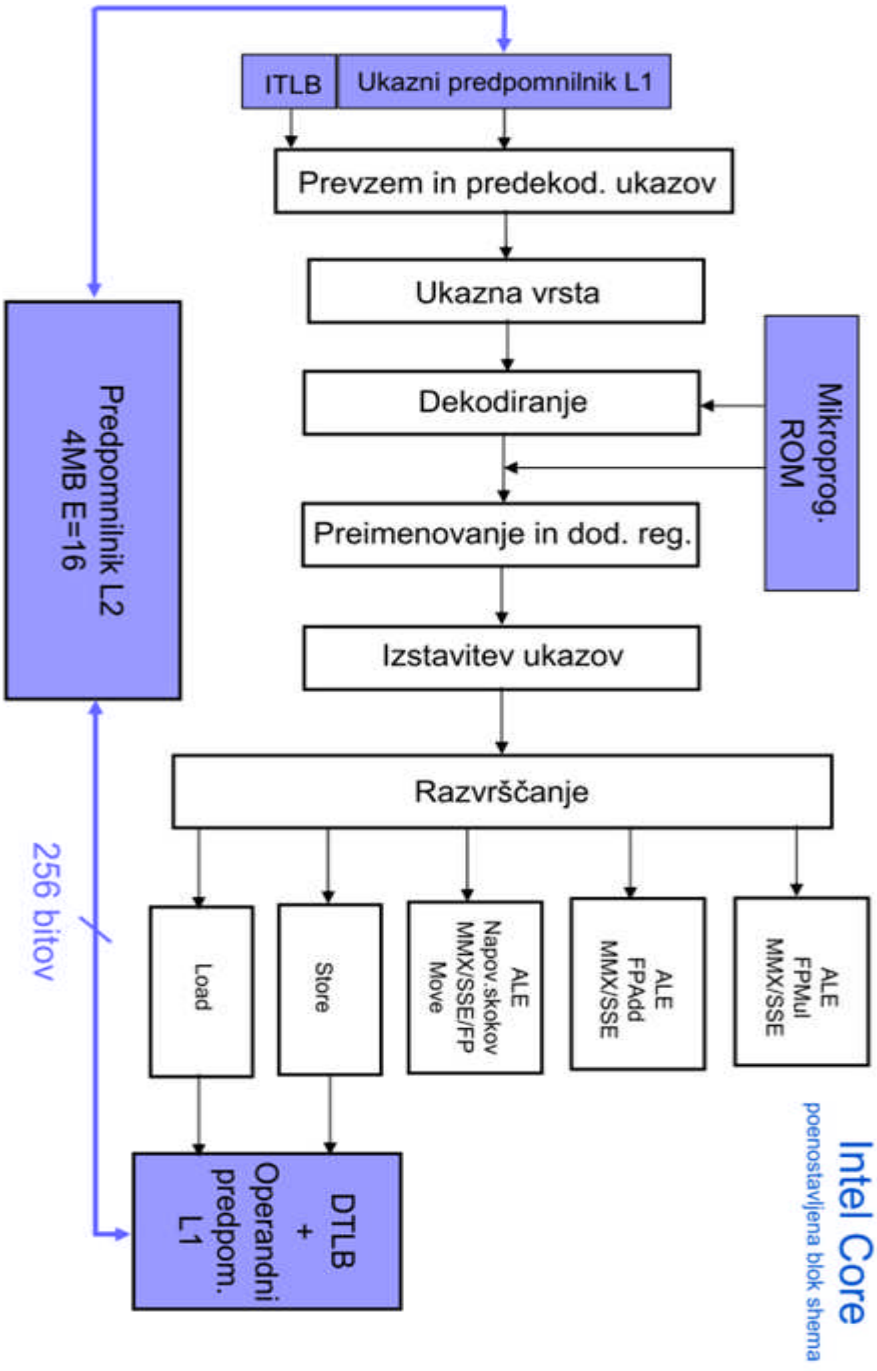
6.1. Arhitektura procesorjev Intel Core

Novo arhitekturo procesorjev (pomlad 2006) so pri Intelu najprej poimenovali NGMA (Next Generation Micro-Architecture), nato pa je dobila ime Core Microarchitecture.
Procesorji s Core arhitekturo so 64-bitni superskalarni cevovodni procesorji vrste SIMD (Single Instruction Multiple Data). Ta oznaka velja za MMX in SSE ukaze. Ti ukazi so bili dodani za izboljšanje zmogljivosti multimedijskih programov.
Ti ukazi omogočajo hkratno delovanje več ALE oziroma delitev ene široke (npr. 64-bitne) ALE na več paralelno delujočih ALE (npr. 2x32-bitni, 4x16-bitne, ...).



Slika 4: Primer tipičnega SSE ukaza.

To je eden od cenejših načinov uvajanja paralelizma za celoštevilčne operacije in tudi operacije v plavajoči vejici. Ker pa je večina procesorjev v Core arhitekturi večjedrnih, jih lahko označimo enako kot MIMD procesorje. Procesorji so bili ob predstavitvi izdelani v 65nm tehnologiji, od leta 2008 dalje jih izdelujejo v 45nm tehnologiji.



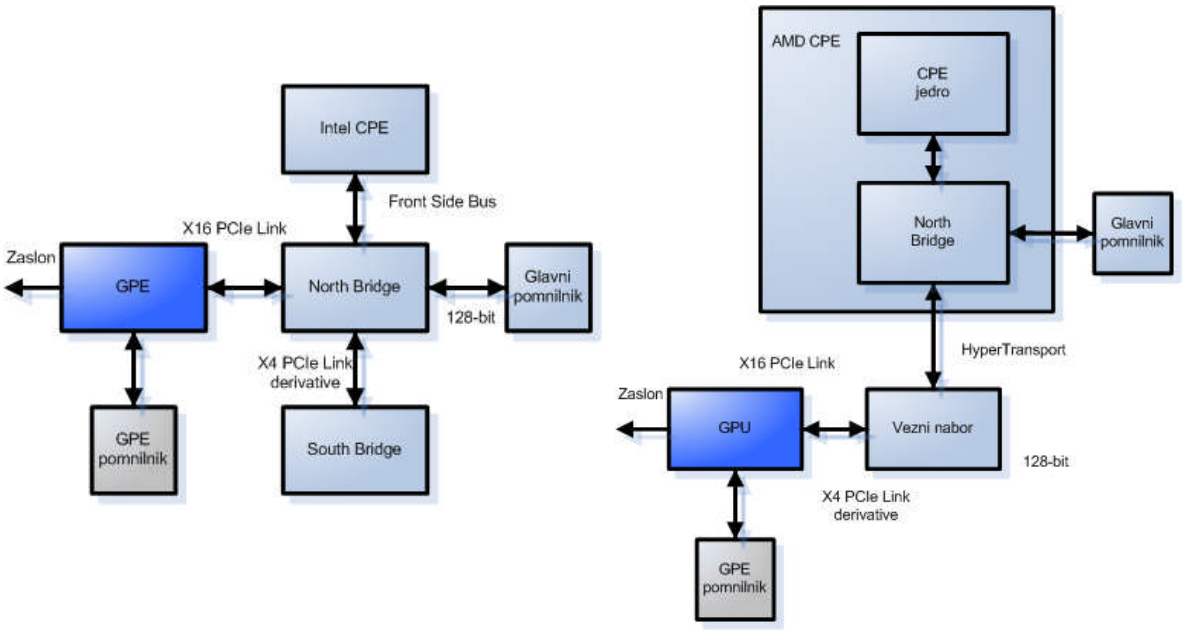
Slika 5: Zgradba procesorja z Intel Core.

7. Arhitektura grafične procesne enote

V tem poglavju sem opisal arhitekturo sodobnih grafičnih kartic. Najprej sem predstavil, kako je GPE povezana z ostalimi enotami v računalniškem sistemu. Nato je predstavljena arhitektura GPE na splošnem primeru in na koncu poglavja na konkretnem primeru grafične kartice novejše generacije NVIDIA serije GTX200.

7.1. GPE v računalniku

Pri današnjih osebnih računalnikih se najpogosteje uporablja takšna sistemska konfiguracija, kjer imata GPE in CPE vsaka svoj pomnilnik. Grafična kartica je danes v večini primerov povezana s CPE prek vodila PCI-Express x16, ki omogoča hitrost prenosa do 16GB/s. CPE lahko dostopa do pomnilnika GPE in obratno, vendar je pasovna širina nižja kot pri komunikaciji z lastnim pomnilnikom.



Slika 6: Sistemska konfiguracija sodobnih osebnih računalnikov. Na levi strani vidimo sistemsko arhitekturo Intel Core, na desni pa AMD sistemov. Grafična procesna enota ima v obeh primerih lasten pomnilnik.

Vezje North Bridge skrbi za komunikacijo med pomnilnikom in centralnim procesorjem. Poimenujemo ga tudi severni most ali krmilnik pomnilnika. South Bridge ali južni most pa skrbi za komunikacijo z ostalimi enotami.

Pri Intelu komunicira procesor s pomnilnikom prek vezja North Bridge, z GPE in s krmilnikom South Bridge. Pri AMD sistemih pa North Bridge komunicira s pomnilnkom DDR2 in z veznim naborom čipov, ki komunicira z GPE ter ostalimi enotami. Vezje North Bridge je pri AMD vgrajeno v centralni procesor.

Integrirane grafične kartice nimajo svojega pomnilnika, zato uporabljajo glavni pomnilnik. Takšni sistemi imajo pri prikazovanju grafike slabše zmogljivosti, saj je hitrost dostopa do glavnega pomnilnika omejena z razpoložljivostjo glavnega pomnilnika. Omejena je s kapaciteto in s pasovno širino dostopa.

Grafične kartice z lastnim pomnilnikom omogočajo večjo pasovno širino in manjše zakasnitve pri dostopu do pomnilnika, zato so bolj zmogljive.

Sistemi, zasnovani za visoko grafično zmogljivost uporabljajo dve ali več GPE enot, ki so med seboj povezane. Kot primer naj omenim SLI (Scalable Link Interconnect). To rešitev je razvila Nvidia za povezovanje dveh ali več grafičnih kartic, ki na enem izhodu prikazujejo sliko. Rešitev uporablja dodatno logiko pri povezovanju kartic in posebne algoritme, ki računsko obremenitev enakomerno porazdelijo na dve ali več grafičnih kartic. Vse novejše NVIDIA grafične kartice podpirajo to tehnologijo. NVIDIA SLI omogoča do dvakratne pohitritve v zmogljivosti pri uporabi dveh grafičnih kartic in 2.8-kratno povečanje zmogljivosti pri uporabi treh v primerjavi z eno grafično kartico [8]. Idealno je povezovanje enakih grafičnih kartic. V primeru, ko imamo različne, počasnejši procesor ali pomnilnik počasnejše omejuje delovanje drugih grafičnih kartic na to (manjšo) hitrost.

ATI CrossFire (znan tudi kot CrossFireX) je zelo podobna tehnologija, ki jo je razvil ATI in podpira ATI/AMD grafične kartice. Namen je isti kot pri NVIDIA SLI, to je izboljšanje grafične zmogljivosti. Pri ATI CrossFire je pomembno, da so grafične kartice iz iste generacije, ni pa nujno da so enaki modeli.

7.2. Osnovna enotna arhitektura GPE

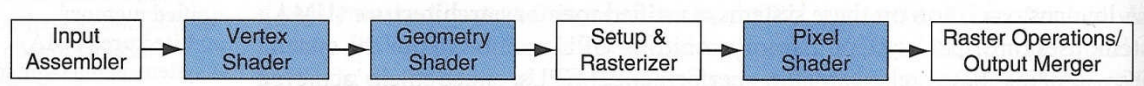
Današnje GPE komunicirajo s CPE preko vodila PCI-Express (PCIe). Prejšnje generacije računalnikov so uporabljale vodilo AGP. Grafične aplikacije kličejo OpenGL ali Direct3D funkcije, ki se izvajajo na GPE. Aplikacija pošilja ukaze, programe in podatke na GPE preko grafičnih gonilnikov, optimiziranih za določeno arhitekturo GPE.

Arhitektura GPE temelji na paralelnih poljih več procesorjev. Ti procesorji omogočajo poenoteno obdelavo točk, geometrijske operacije nad točkami in večnitnost. Včasih so imele GPE ločene enote za različne vrste obdelav, danes pa so te enote bolj splošne in na njih se lahko izvajajo različne funkcije. Te funkcije so filtriranje tekstur, vzorčenje točk, operacije nad točkami, glajenje robov, stiskanje, raztezanje, prikazovanje, dekodiranje videa in obdelovanje visoko kakovostnega videa.

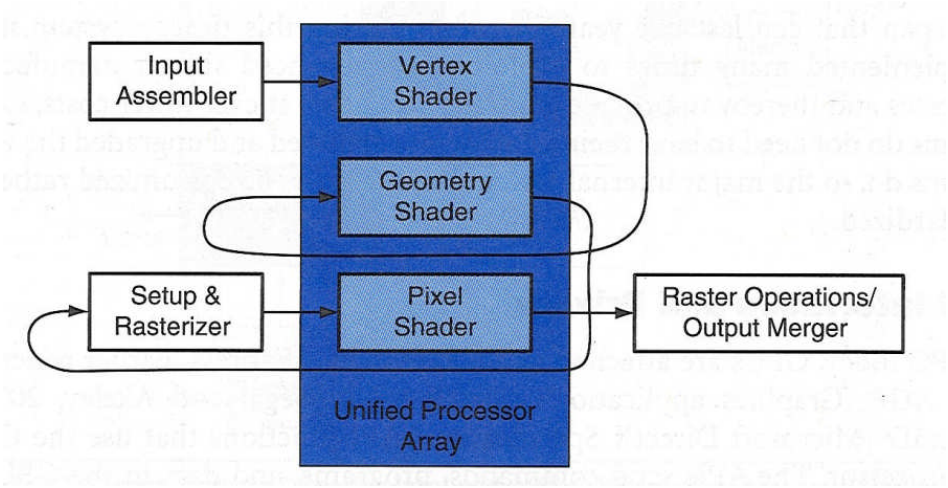
7.2.1. Logični grafični cevovod

Programski vmesniki imajo pomembno vlogo pri razvoju GPE in CPE procesorjev. Za programiranje grafike sta najbolj razširjena dva standarda, to sta OpenGL in Direct3D, ki je del Microsoftovega DirectX programskega vmesnika. OpenGL je odprt standard, razvilo ga je podjetje Silicon Graphics, sedaj pa skrbi za njegov razvoj industrijski konzorcij Khronos. Za razvoj Direct3D skrbi predvsem Microsoft. Oba standarda imata podobno strukturo. Definirata logično strukturo cevovoda, ki je strojno realizirana na GPE.

Slika 7 prikazuje stopnje logičnega cevovoda grafične kartice. Modro obarvani kvadratki so najpomembnejši deli logičnega grafičnega cevovoda. To so stopnja za obdelavo vozlišč (Vertex Shader), stopnja za obdelavo likov (Geometry Shader) in stopnja za obdelavo točk (Pixel Shader).

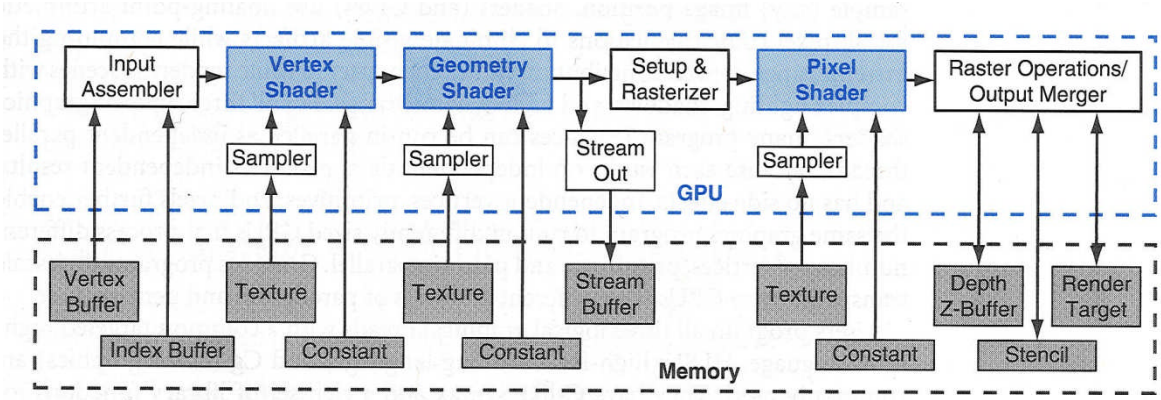


Slika 7: Logični grafični cevovod.



Slika 8: Preslikava logičnega grafičnega procesorja v fizične procesorje. Modro obarvane stopnje se izvajajo na fizičnih procesorjih.

Na začetku cevovoda prva stopnja zbira vhodne podatke (Input Assembler). Stopnja za obdelavo vozlišč te podatke že obdela, kar vključuje preslikavo položaja prostorskih geometrijskih točk v položaj na zaslonu, njihovo osvetlitev in določanje barve. Tretja stopnja za obdelavo likov (Geometry Shader) izvaja operacije nad osnovnimi liki in jih lahko dodaja ali odstranjuje. Naslednja stopnja (Setup in Rasterizer) tvori skupine točk, s katerimi so liki prekriti. Stopnja za obdelavo točk (Pixel Shader) skrbi za prikazovanje vzorcev in za barve. V zadnji stopnji cevovoda (Raster Operations) računamo globino slike v 3D prostoru in delamo majhne popravke, kjer je to potrebno. Nato prikažemo sliko na zaslonu. Grafični vmesnik (API) in grafični cevovod skrbita za vhodne in izhodne podatke, pomnilniške objekte ter vse operacije, ki so potrebne za prikazovanje slike.



Slika 9: Logični grafični cevovod in Direct3D 10.

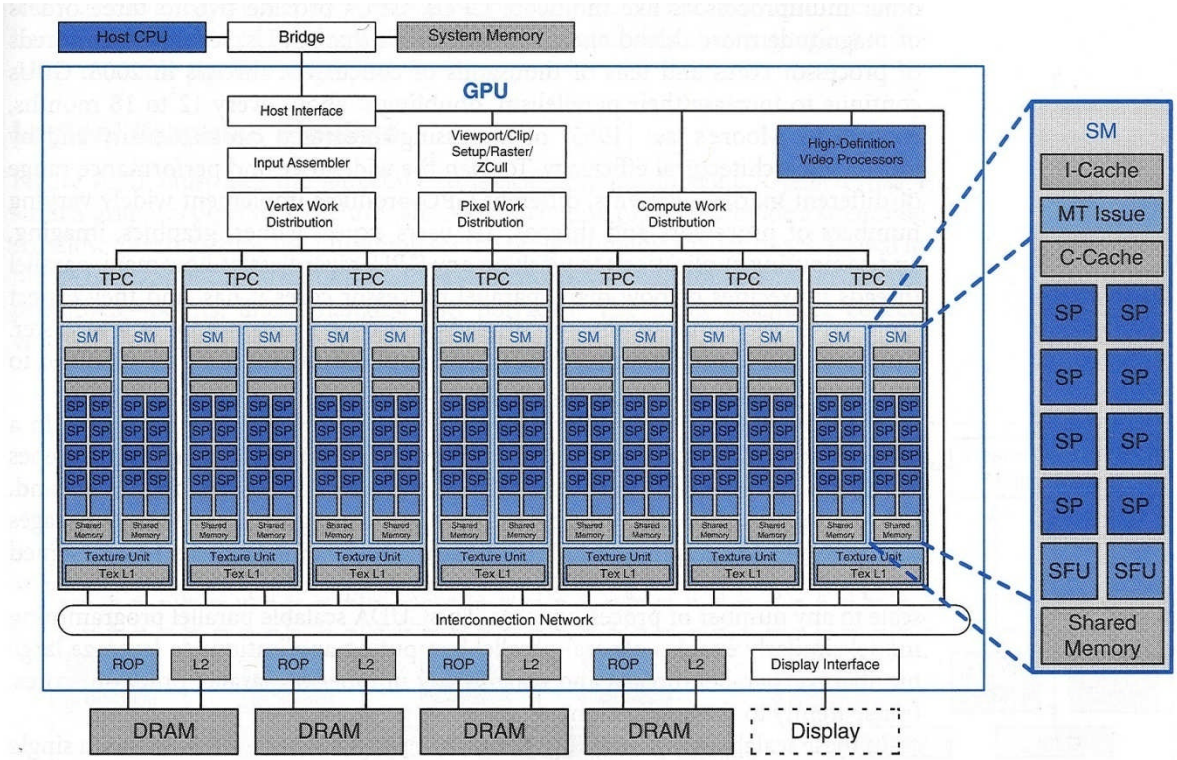
Na sliki 9 je opisan cevovod Direct3D 10. Za izvajanje vsake stopnje tega cevovoda skrbi strojna oprema na grafični kartici. To so lahko posebne enote, namenjene točno določeni stopnji, ali pa procesor, ki ga lahko sprogramiramo tako, da skrbi za izvajanje določene stopnje. Modro so obarvane stopnje, ki jih običajno izvaja procesor. Fiksne funkcije, za katere običajno skrbijo posebne enote, so bele barve, sivo pa so obarvani različni pomnilniki in predpomnilniki, ki jih ostale enote potrebujejo za svoje delovanje [1].

Danes imajo skoraj vse grafične kartice arhitekturo SIMT (Single Instruction Multiple Thread). Ta arhitektura ustvari, upravlja, razvršča in izvaja istočasne niti v skupine paralelnih niti, (angleško warp, ime izhaja iz tkanja, ki je prva tehnologija obdelave paralelnih niti).

7.2.2. Procesorsko polje

Današnje GPE vsebujejo veliko procesorskih jeder, ki so organizirana kot večnitni procesorji. Slika 10 prikazuje Tesla arhitekturo proizvajalca NVIDIA, ki temelji na arhitekturi GPE Geforce 8800. Ta grafična kartica ima 112 jeder, ki podpirajo večnitnost, zato jim pravimo podatkovno pretokovni procesorji (Streaming Processor, SP). Osem od teh jeder je združenih v celoto, ki ji rečemo podatkovno pretokovni multiprocesor (streaming multiprocessor, SM). Na tej GPE imamo 14 (večjedrnih) podatkovno pretokovnih multiprocesorjev (SM). Vsak SM ima osem jeder, dve enoti za posebne funkcije, predpomnilnik, enoto za večnitno izvajanje ukazov in deljen pomnilnik. Dva SM sta združena in si delita enoto za obdelovanje tekstur in predpomnilnik L1. Procesorji si delijo pomnilnik, do katerega dostopajo po štirih kanalih. Do vsake od štirih enot pomnilnika (slika 10) proceosr dostopa preko 64-bitnega vodila. Skupno imamo torej 112 jeder (8 x 14), od katerih vsako podpira večnitnost in omogoča do 96 niti. Grafična kartica tako podpira hkratno izvajanje več kot 10.000 niti.

Takšna arhitektura je razširljiva in primerna za bolj ali manj zahtevne GPE. Takšno arhitekturo lahko uporabimo pri manj in bolj zmogljivih grafičnih čipih, tako da prilagodimo število procesorjev, multiprocesorjev in količino pomnilnika.



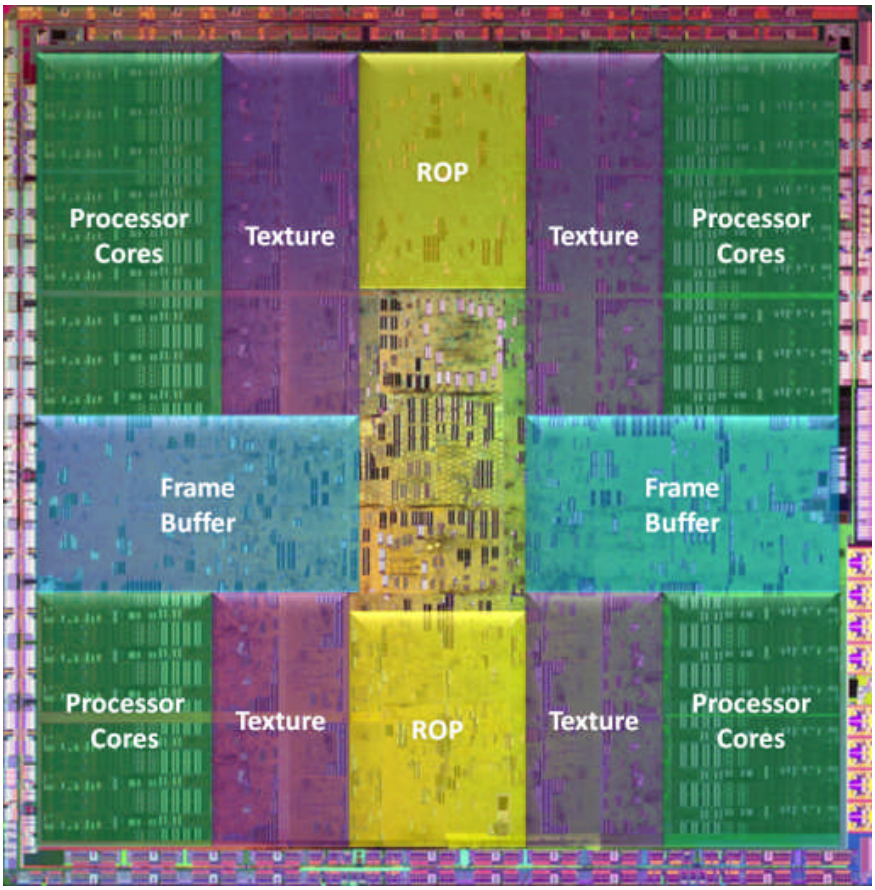
Slika 10: Osnovna arhitektura Tesla, ki temelji na GPE Nvidia 8800. Procesorska jedra so združena v skupine po 8 in tako združena tvorijo streaming multiprocesor (SM). Le-teh je 14. Skupaj imamo 8 x 14 = 112 jeder.

Procesorji v današnjih GPE imajo enotno arhitekturo in na njih se lahko izvajajo različne funkcije. Pri prejšnjih generacijah grafičnih kartic je bil vsak procesor namenjen samo za izvajanje določenih funkcij.

7.3. Grafični čip NVIDIA GTX200

NVIDIA Geforce 8800 GTX je bila vodilna kartica v letu 2007. Pozneje je ATI razvil jedro R600, poznano kot Radeon HD2000 in HD3000. Nato je NVIDIA predstavila grafično kartico 8800 Ultra z nekoliko višjimi frekvencami procesorske ure in s hitrejšim pomnilnikom. Ker pa to ni bilo nič revolucionarnega, je nekaj mesecev kasneje izšla 9800 GTX z jedrom G92, ki pa v primerjavi z 8800 GTX in Ultra tudi ni imela pričakovane izboljšane zmogljivosti. Jedro G92 je temeljilo na jedru G80, ki ga imajo grafične kartice serije 8000 [3].

Konec leta 2008 je NVIDIA končno predstavila povsem novo jedro GT200 z grafično kartico GTX280. Ta kartica predstavlja velik napredek v primerjavi s prejšnjimi modeli. Ima 1.4 milijarde tranzistorjev, ki so bili na začetku izdelani v 65nm tehnologiji, danes pa imamo na tržišču grafične kartice serije GTX 200 s čipi, izdelanimi v 55nm tehnologiji.



Slika 11: Na sliki vidimo kolikšen del procesorja se uporablja za teksture, za procesorska jedra in za pomnilnik in kje se te enote nahajajo na čipu.

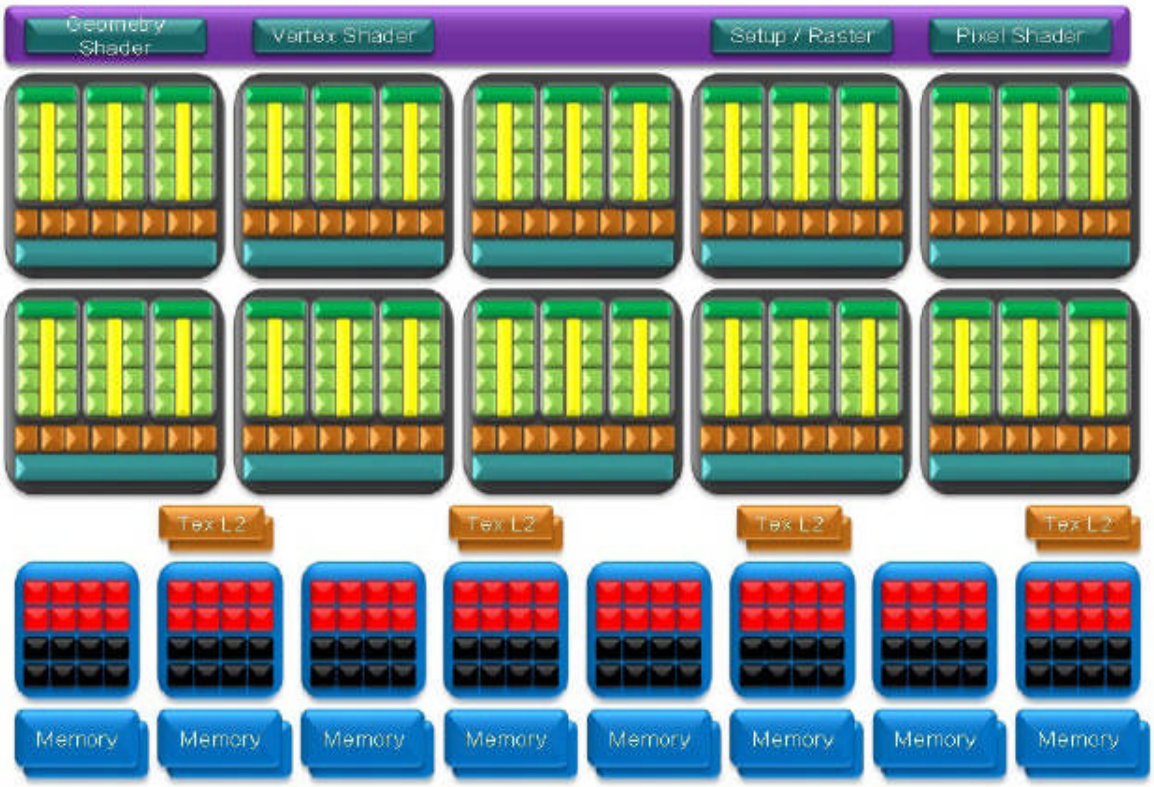
8800 GTX je imela računsko moč v plavajoči vejici 518 GFLOPS, GTX280 pa se približuje TFLOPS-u, računsko moč ima namreč 933 GFLOPS.

NVIDIA in AMD sta enotna pri tem, da bodo igre v prihodnosti zahtevale znatno večjo zmogljivost GPE pri aritmetičnih operacijah v primerjavi z močjo, potrebno za obdelovanje tekstur. Zato je pri razvoju grafičnih kartic poudarek na učinkovitosti izvajanja aritmetičnih operacij.

Pri jedru G80 (Geforce 8000 serije) je bilo uporabljenih 24 specializiranih procesorjev za obdelavo točk, ROP (Raster Operation Processor). Pri novem jedru GT200 se je število ROP povečalo na 32. AMD/ATI medtem ni predstavil revolucionarnega jedra, zato dosegajo primerljive rezultate tako, da pri novejših karticah višajo frekvence delovanja pomnilnika in procesorske ure.

NVIDIA predstavlja svoje nove grafične kartice (jedro GT200) kot GPE, ki imajo podatkovno pretokovne multiprocesorje tipa SIMT. Tehnologija je opisana na koncu poglavja 7.2.1. V primerjavi s SIMD arhitekturo pri SIMT nimamo omejitve pri velikosti vektorjev, s katerimi procesiramo. Točke, ki jih obdelujemo, so predstavljene kot vektorji s štirimi vrednostmi v plavajoči vejici z enojno natančnostjo.

Slika 12 prikazuje arhitekturo jedra GT200, ki ga imajo grafične kartice GTX280. Že na prvi pogled je podobna prej opisani Tesla arhitekturi, ki temelji na grafičnem čipu Geforce 8800, le da ima več jeder. V jedru GT200 imamo deset TPC-jev (Texture Processor Clusters). Vsak TPC tvorijo tri SM, od katerih ima vsak osem jeder.



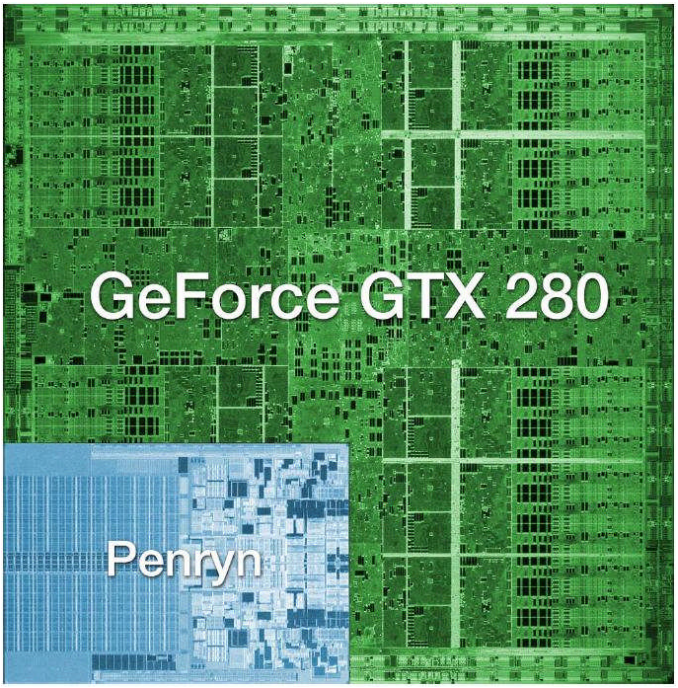
Slika 12: Organizacija procesorskih jeder pri grafični kartici Geforce GTX280.

Znotraj vsakega TPC se nahaja enota, ki skrbi za izračune tekstur in tri podatkovno pretokovne multiprocesorje (SM, Streaming Multiprocessor). Na sliki 12 vidimo TPC od blizu. Vsak od treh SM ima osem jeder in skupni pomnilnik, ki si ga delijo jedra. Grafična kartica GTX280 ima 240 jeder (8 x 3 x 10).

Multiprocesorji lahko naenkrat izvajajo do 1024 niti. To je posebej uporabno pri operacijah s teksturami in pomeni manjše zakasnitve. Na grafični kartici GTX280 lahko hkrati poganjamo do 30.720 niti.

Slika 13: Arhitektura GPE GTX280. TPC ima tri SM. Vsak SM ima osem jeder.

Vsak od multiprocesorjev ima kar 16.384 registrov. Vsaka nit ima tako na razpolago do 16 registrov.



Slika 14: Primerjava velikosti jedra procesorja Quad-Core in jedra grafične kartice GTX280. Grafični procesor NVIDIA GTX280 je v primerjavi s CPE Intel Quad-Core precej večji.

8. Programiranje grafičnih procesorjev za splošne namene

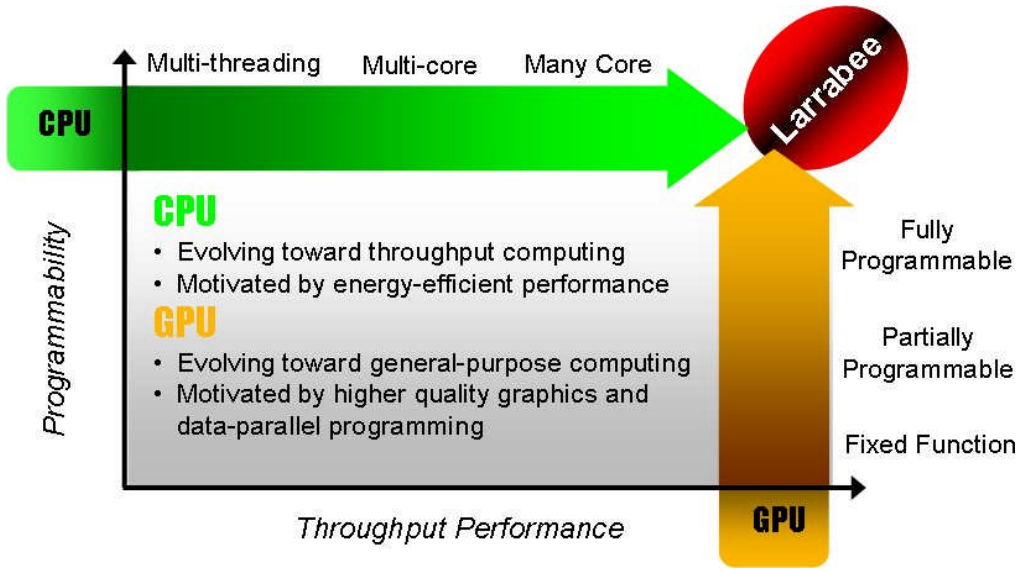
GPGPU (General-Purpose computing on Graphics Processing Units) označuje uporabo GPE za izvajanje splošnih aplikacij, ki jih običajno izvaja CPE. Z GPGPU se že več let ukvarjajo razvijalci iger in predvsem v zadnjem času se uporaba razširja na aplikacije, ki niso nujno povezane z računalniško grafiko. Orodja postajajo vedno bolj dostopna in enostavna za uporabo. Nekaj področij, na katerih se uporablja računska moč grafičnih procesorjev, je naštetih v nadaljevanju:

- digitalno procesiranje signalov: Fourierove transformacije, analize signalov,
- računalniški vid: AdaBoost, Houghove transformacije,
- enkripcija: RSA algoritmi,
- bio-informatika: gensko vzorčenje,
- optimizacija: optimizacija vezij, razvrščanja procesov, planiranja kritičnih poti.
- obravnavanje podatkov: o vremenu, geološke raziskave,
- virtualizacija,
- statistične obdelave,
- antivirusni programi,
- prepoznavanje v realnem času: sledenje obrazu, detektorji gibanja,
- folding@home: porazdeljeno računanje,
- računanje verjetnosti: vremenske napovedi, simulacije, generator naključnih števil, genetsko programiranje,
- urejevalniki slik v realnem času: CellSec Photoshop,
- simulacije v realnem času: obnašanje tekočin v 3D prostoru, mehanične simulacije predmetov, širjenje zvoka v prostoru, valovanje na vodi, kemijske reakcije,
- algoritmi za histograme,
- vzorčenje,
- paralelizacija stohastičnih algoritmov pri raziskovanju kemijskih reakcij, procesov,
- elektrodinamika,
- izračuni fizikalnih enačb v igrah (interakcija objektov v prostoru z upoštevanjem mase, gravitacije, pospeškov),
- operacije z matrikami,
- astrofizika,
- molekularne raziskave,
- finančne analize,
- razno.

8.1. Orodja za programiranje grafičnih procesorjev

Orodja za programiranje grafičnih procesorjev so usmerjena v izkoriščanje podatkovno-paralelnega procesiranja, ki ga GPE omogočajo. Z njimi lahko programiramo cevovode GPE. Najbolj razširjeno orodje za GPGPU je CUDA, ki ga bom podrobneje opisal v nadaljevanju. Poleg tega se v današnjem času za GPGPU uporabljata tudi programska vmesnika za programiranje GPE Brook in CAL (Compute Abstraction Layer). CAL je nizko-nivojski način programiranja z vmesnikom za AMD-jeve GPE, medtem ko je Brook poseben programski jezik, s katerim programiramo tokove, ki se izvajajo na GPE. Nastal je leta 2004. Naj omenim še orodje Jacket, ki ga je razvilo podjetje AccelerEyes. Programi, napisani v Matlabu, se lahko s pomočjo vmesnika CUDA izvajajo na grafičnih karticah.

Poleg omenjenega lahko GPE programiramo še v OpenGL, OpenCL, Direct 3D ali C programskih jezikih. Alternativa orodju CUDA je ATI FireStream, ki je namenjeno za uporabo na grafičnih karticah ATI. Tudi Intel razvija strojno opremo, ki bo namenjena paralelnemu procesiranju. Kmalu bo izšel čip Larrabee, ki bo nekakšen hibrid med grafičnimi karticami in procesorji. Prve kartice s to tehnologijo naj bi izšle v prvi polovice leta 2010 in bodo konkurirale NVIDIA in AMD/ATI GPE, ki se jih uporablja za GPGPE. Larrabee bo po pomnilniku in po kompatibilnosti z x86 arhitekturo podobna centralnemu procesorju medtem ko bo SIMD arhitektura in enote za vzorčenje tekstur bolj podobne tistim pri GPE.



Slika 15: Izkoriščanje značilnosti GPE in CPE arhitekture za doseganje boljših zmogljivosti pri računanju.

Larrabee bo podpirala 3D grafiko (DirectX/OpenGL), kar bo uporabno za igre. Hibridne lastnosti CPE/GPE bodo ta čip naredile uporaben za GPGPU, možna pa bo tudi uporaba v superračunalnikih.

8.2. Zgodovina GPGPU

Paralelno procesiranje se je začelo razvijati v 80-ih in je v začetku 90-ih let je doživelo pravi razcvet. Pri tem načinu programiranja so običajno enake operacije izvršene nad velikim številom različnih podatkov. Pogosto so to operacije nad posameznimi elementi velikih tabel kot podatkovnih struktur. V tem obdobju so bile raziskane in narejene številne arhitekture. Razvijali so super-računalnike, ki so bili zelo zmogljivi in dragi. Le malo ljudi je imelo dostop do teh specifičnih sistemov, ki so se najpogosteje uporabljali v laboratorijih in velikih izobraževalnih središčih.

Kljub temu, da takšna oprema ni bila dostopna širšemu krogu uporabnikov, je bilo za paralelno procesiranje veliko zanimanja, zato se je o tem veliko pisalo. Razvijali so se tudi programski jeziki, moduli in podatkovni algoritmi za optimizirano paralelno procesiranje, ki so bili namenjeni reševanju širokega spektra različnih računskih problemov. Poleg tega so se razvijale tudi nove arhitekturne rešitve. Število prodanih super-računalnikov je bilo relativno majhno, zato je bilo tudi malo programerjev, ki so se s tem ukvarjali. Te velike super-

računalnike, ki so bili cenovno nedostopni, so pozneje nadomestile skupine poceni osebnih računalnikov. Ti računalniški sistemi so bili povezani v mreže. Računske probleme so reševali porazdeljeno. Takšnemu procesiranju pravimo porazdeljeno procesiranje. Te skupine računalnikov so po moči napredovale hitreje od super računalnikov.

Po letu 2000 se je pri razvoju mikroprocesorjev hitrost večanja računske moči močno zmanjšala. Zmogljivejše skupine računalnikov je bilo treba graditi iz večjega števila računalnikov, kar je oteževalo razširljivost in povečalo porabo prostora in energije. Današnje GPE imajo veliko računsko moč, ki je primerljiva z močjo skupin računalnikov ali super-računalnikov, ki so se uporabljali pred leti. Danes lahko s sodobnimi programskimi orodji na GPE dosežemo takšne rezultate, za kakršne so se včasih trudili znanstveniki na super-računalnikih.

GPE so mnogojedrne, kar pomeni, da lahko imajo na stotine jeder. Večjedrni procesorji, kot jih najdemo v CPE, pa imajo v primerjavi z GPE le dva, tri, štiri, šest ali osem jeder. GPE lahko izvaja tudi 10.000 niti hkrati in ima zmogljivost do 1TFLOP. Današnje grafične kartice s svojimi številnimi procesorji predstavljajo povratek v zlato dobo paralelnega procesiranja. GPE lahko tako izvajajo številne algoritme, ki so bili narejeni za super-računalnike in so se uporabljali v 80. letih. Uporabniki, ki uporabljajo to tehnologijo za znanstvene in inženirske raziskave dosegajo desetkratne in večje pohitritve na GPE v primerjavi s CPE.

Danes imamo v uporabi več kot 80 milijonov grafičnih kartic, ki podpirajo CUDA tehnologijo. NVIDIA proda približno milijon GPE-jev vsak teden, kar pomeni 100 GPE-jev vsako minuto. Vzroki za takšno prodajo se skrivajo v cenovni dostopnosti. Grafično kartico z zmogljivostjo 500GFLOPS lahko dobimo za manj kot 200 EUR. To posledično vpliva na množično uporabo podatkovno-paralelnih računalnikov, ki jih danes najdemo že praktično povsod.

8.3. Programsko okolje CUDA

CUDA je programsko okolje, ki ga je NVIDIA razvila leta 2007. Podpira računanje za splošne namene na strojni opremi NVIDIA. Temelji na programskem jeziku C, kar za programerje pomeni manj učenja, saj ni veliko novega. Dodani so le ukazi, ki omogočajo razširitev paralelnega procesiranja na grafične kartice z mnogo jedri. Programerju omogoča programiranje v C-ju brez prevajanja problemov v grafični koncept. CUDA razbije programe, ki rešujejo številne kompleksne probleme, na majhne dele, ki se izvajajo na CPE, in na dele, ki se izvajajo na GPE. Poleg programiranja v visoko nivojskih jezikih CUDA omogoča razvijalcem uporabo standardnih industrijskih API (Application Program Interface). Primera takšnih API sta Microsoft DirectX in OpenCL, ki se enostavno združita s tem razvijalskim okoljem.

CUDA izkorišča CPE in GPE v delih aplikacij, kjer sta najmočnejša. Del programa, kjer je zahtevano serijsko računanje, se izvaja na CPE-ju, medtem ko se paralelni deli aplikacij prenesejo na GPE. Ta lastnost omogoča znatne pohitritve v aplikacijah, ki se poganjajo z orodjem CUDA, količina truda za te dosežke pa je relativno majhna. Poleg tega CUDA omogoča, da se negrafične ali računske aplikacije izvajajo na GPE.

CUDA izrablja tudi nekaj strojnih izboljšav, ki jih pri programiranju preko grafičnega vmesnika API ni mogoče izkoristiti. Najpomembnejši med njimi je deljen pomnilnik, do katerega lahko dostopamo v paralelnih blokih niti. To omogoča periodično shranjevanje podatkov in lahko znatno pohitri aplikacije.

9. Primerjava izvajanja programa za množenje matrik na CPE in GPE

Za primerjavo sem testiral izvajanje množenja matrik na CPE in na GPE. Program za množenje matrik je relativno preprost, zato ga je mogoče tudi na preprost način optimizirati za paralelno računanje. Razlike v izvajanju med CPE in GPE so tukaj še posebej velike. Najprej sem predstavil sistem, na katerem sem izvajal meritve, nato problematiko programa. Potem sem opisal delovanje programa, ki je ključno za razumevanje razlik in na koncu dodal še rezultate meritev.

Pri sodobnih programskih aplikacijah je precej delov programa takih, da vključujejo veliko paralelizma. CUDA naprave omogočajo pospešeno izvajanje takih aplikacij, saj dobro implementirajo paralelno procesiranje.

9.1. Testni sistem

Program sem poganjal na dveh računalnikih. Prvi računalnik, na katerem sem delal meritve v laboratoriju, ima dvojedrni procesor Intel Core 2 Duo E2200, ki deluje s frekvenco 2.0 GHz in ima 2 MB L2 predpomnilnika. Grafična kartica NVIDIA 8800GT ima 112 procesorjev, ki delujejo s frekvenco ure 600 MHz in 512 MB pomnilnika, ki deluje s frekvenco ure 1500 MHz.

Drugi računalnik, na katerem sem izvajal meritve doma, ima procesor Intel Core 2 E7400, ki deluje s frekvenco 2.8 GHz, in ima 3MB L2 predpomnilnika. Grafična kartica NVIDIA GTX260 ima 216 procesorjev, ki delujejo s frekvenco ure 626 MHz. Kapaciteta pomnilnika je 896 MB in frekvenca delovanja ure je 2106 MHz.

Na domačem računalniku sem pri začetnih meritvah ugotovil, da so časi izvajanja za faktor dva manjši. To je tudi razumljivo, saj je procesor precej hitrejši in ima več predpomnilnika L2. Tudi grafična kartica ima skoraj še enkrat toliko procesorjev in pomnilnika, kot tista na slabšem sistemu. Podatke o sistemih vidimo v tabeli 1.

		Testni sistem (Core 2 Duo E7400, Nvidia GTX260)	**Sekundarni testni sistem (Core 2 Duo E2200, Nvidia 8800GT)**
CPE	Frekvenca	2.8 GHz	2.0 GHz
	L2 predpomnilnik	3 MB	2 MB
GPE	Št. Procesorjev	216	112
	Hitrost jeder	626 MHz	600 MHz
	Pomnilnik	896 MB	512 MB
	Hitrost pomnilnika	2106 MHz	1500 MHz

Tabela 1: Testni sistemi, na kateremu so potekale meritve.

Večino meritev sem izvajal samo na zmogljivejšem sistemu, tudi tukaj zapisani rezultati so nastali na tem sistemu. Meritve sem poganjal večkrat. Pri manjših matrikah, kjer so bili časi izvajanja do nekaj ms, sem meritve ponovil 100-krat. Pri večjih matrikah, kjer so bili izvajalni časi nekaj sekund ali minut, rezultati pa niso veliko odstopali, sem meritve ponovil desetkrat. Pri matrikah, kjer so bili izvajalni časi nad minuto, sem ponovitve izvedel enkrat. Podatki v tabelah v nadaljevanju so povprečne vrednosti rezultatov.

Testiranje sem izvajal z orodjem CUDA, različica 2.1, ki je dostopno na spletni strani NVIDIA. Trenutno je na voljo novejša različica 2.3.

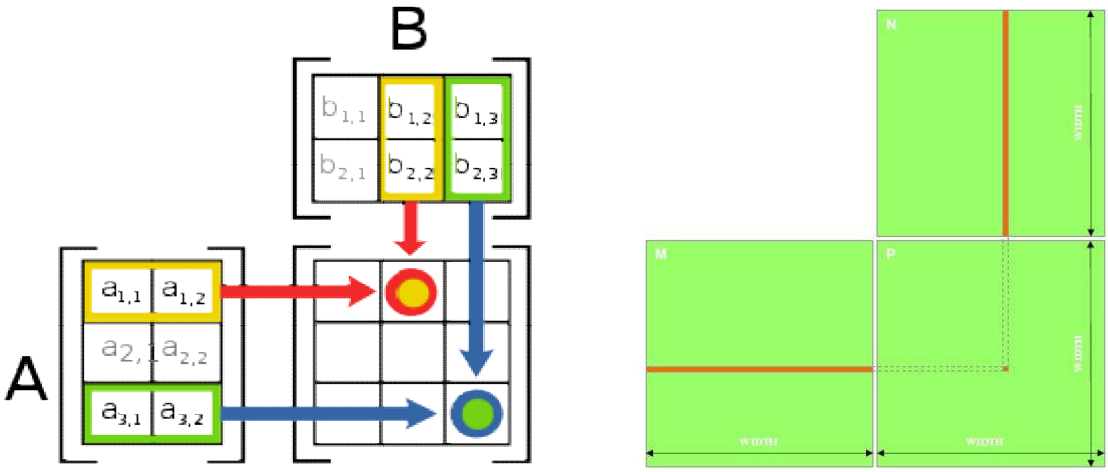
9.2. Predstavitev problema množenja matrik

Prednosti paralelnega procesiranja lepo vidimo na primeru programa za množenje matrik. Ta algoritem namreč lahko enostavno razbijemo na majhne dele, ki se izvajajo paralelno, saj posamezna množenja med seboj niso odvisna. Ko imamo vse zmnožke, rezultate seštejemo in tako dobimo vrednost enega elementa v končni matriki.

Množenje dveh matrik je izvedljivo le, če je število stolpcev prve matrike enako številu vrstic druge matrike. Če je A matrika razsežnosti m-krat-n, (m vrstic, n stolpcev) in je B matrika razsežnosti n-krat-p (n vrstic, p stolpcev), potem je njun zmnožek AB matrika razsežnosti m-krat-p (m vrstic, p stolpcev) definiran kot:

(AB)[i, j] = A[i, 1] * B[1, j] + A[i, 2] * B[2, j] + ... + A[i, n] * B[n, j] za vsak par i in j.

Število vrstic prve matrike je torej enako številu stolpcev druge matrike. Zmnožek dveh matrik dobimo tako, da vsak element vrstice prve matrike pomnožimo z vsakim elementom stolpca druge matrike. Prva vrstica gre s prvim stolpcem, druga z drugim in tako naprej. Dobljene zmnožke seštejemo in rezultat vpišemo v polje tretje matrike.



Slika 16: Množenje matrik – predstavitev principa računanja.

Naj kot primer opišem matriko velikosti 1000x1000 elementov. Pri zmnožku dveh takih matrik dobimo končno matriko enake velikosti, to je 1000x1000 elementov. Za izračun

vsakega od teh potrebujemo 1000 množenj. Ta množenja so med seboj neodvisna, kar pomeni, da lahko računamo vsa hkrati. Ko imamo vse zmnožke, jih seštejemo in dobimo rezultat, ki je en element končne matrike.

Končna matrika ima ravno tako milijon elementov (1000x1000). Za vsakega od teh potrebujemo 1000 zmnožkov, ki jih na koncu seštejemo. Skupno imamo torej milijardo množenj in sešteti moramo milijardo števil. Tu pride do izraza paralelno procesiranje.

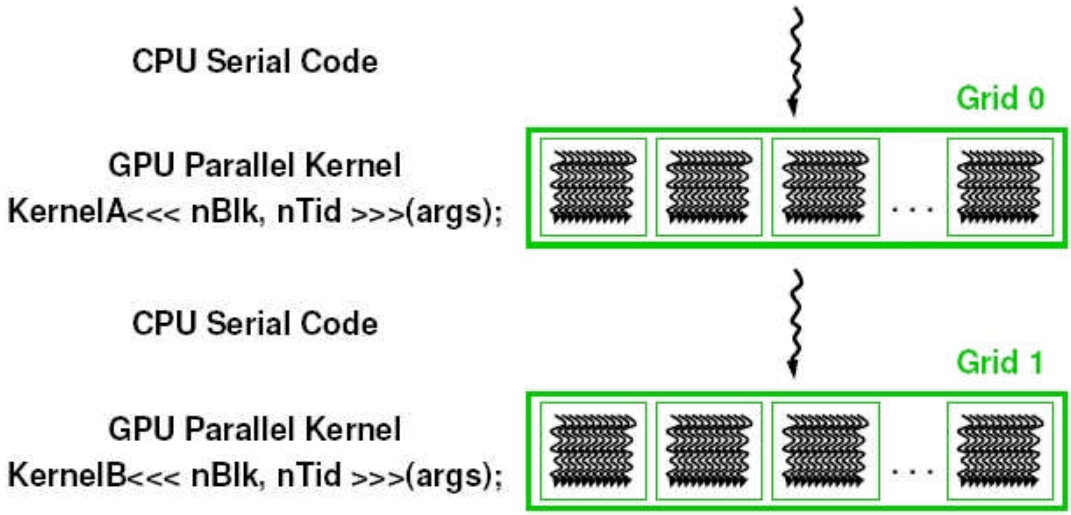
Če imamo dovolj procesorjev, se lahko vsako množenje izvaja na svojem procesorju. Ker vsako jedro večjedrne grafične kartice podpira večnitnost, lahko naenkrat izvajamo veliko število množenj in tako dosežemo enakomerno obremenitev posameznih jeder. Tudi seštevanja lahko izvajamo hkrati, saj med seboj niso odvisna. V našem primeru, kjer imamo 999 seštevanj, lahko z uporabo paralelizma izvedemo vsa seštevanja v 10 korakih. Zaradi naštetega lahko pri izvajanju tega programa na GPE dosežemo velike pohitritve v primerjavi z izvajanjem na CPE.

9.3. Programiranje v okolju CUDA

Programi v okolju CUDA so sestavljeni iz ene ali več stopenj, ki se izvajajo ali na gostitelju (host) ali na napravi (device). Gostitelj je CPE, naprava pa tipično GPE, zato bom v nadaljevanju uporabljal te dve kratici. Naprava ni nujno GPE, lahko je tudi podobna naprava, prilagojena za paralelno procesiranje. Primer takšnih naprav so Tesla kartice. Stopnje, ki vsebujejo malo ali nič paralelizma, so implementirane na CPE, stopnje, ki vsebujejo veliko paralelizma, pa se izvajajo na GPE.

Okolje CUDA kodo programa razdeli na dvoje. Del kode se izvaja na CPE in del na GPE. Koda za CPE je napisana v C-ju in prevede jo standardni C prevajalnik. Izvaja se kot navaden proces. Koda za GPE je napisana v jeziku C, z dodanimi ukazi za označevanje podatkovno-paralelnih funkcij, klicev jeder v GPE in s tem povezanimi podatkovnimi strukturami. Ta koda se prevede s prevajalnikom NVCC (NVIDIA C Compiler) in se izvede na GPE. V primeru, ko GPE ni na voljo ali ko je zahtevana natančnost, kot jo lahko zagotovi le CPE, se koda izvaja na CPE.

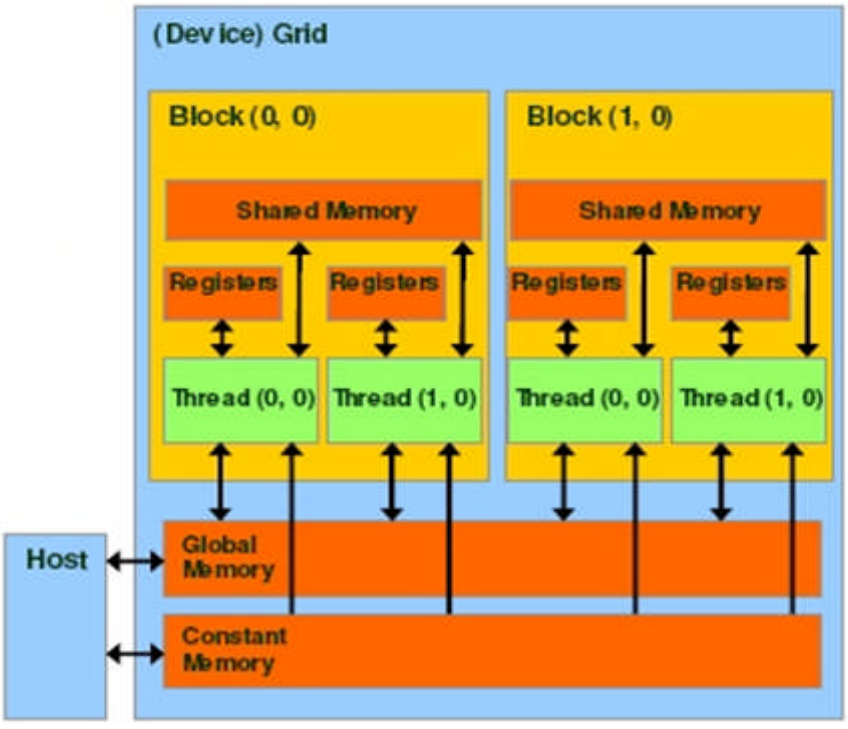
Funkcije, ki se izvajajo na GPE tipično tvorijo veliko število niti z namenom boljšega izkoristka vseh procesorjev. Tem funkcijam rečemo jedrne (kernel) funkcije. Pri programu za množenje matrik je lahko celotno množenje implementirano na jedrih GPE tako, da vsaka nit računa en element izhodne matrike. Število niti je funkcija dimenzij matrike. Pri matriki velikosti 1000 x 1000 elementov se tako tvori milijon niti. Procesorji so zasnovani tako, da porabijo malo urinih ciklov za izvedbo. V primerjavi s CPE so niti, ki se izvajajo na GPE majhne, saj za razvrščanje porabijo le nekaj ciklov, medtem ko tiste na CPE za razvrščanje (scheduling) potrebujejo tipično na tisoče urinih ciklov.



Slika 17: Programi pri CUDA aplikacijah se razdelijo na niti. Pri enem klicu jedra se tvori skupina niti (grid).

Program se začne izvajati na CPE. Ko program na CPE pokliče ustrezno funkcijo, se izvajanje prenese na GPE, kjer se tvori veliko število niti. Vsem nitim, ki nastanejo pri enem klicu jedra, pravimo rešetka (po angleško grid). Ko so vse niti v jedru GPE pripravljene za izvajanje, se prilegajoča rešetka uniči in izvajanje se nadaljuje na CPE, dokler ni poklicana naslednja jedrna funkcija GPE [9].

Pri CUDA programiranju CPE uporablja glavni pomnilnik, GPE pa svoj, grafični pomnilnik. GPE so ponavadi realizirane na grafični kartici in imajo svoj pomnilnik tipa DRAM (Dynamic Random Access Memory).



Slika 18: Pomnilnik naprav CUDA. Za programerja sta najpomembnejša globalni in konstantni spomin.

Slika 18 prikazuje pomnilnik GPE, kot ga vidi programer, ki programira z orodjem CUDA. GPE ima globalni in konstantni pomnilnik. CPE lahko bere in piše v oba izmed teh pomnilnikov, GPE pa lahko iz konstantnega pomnilnika le bere. Do drugih notranjih pomnilnikov in registrov pa ima GPE dostop za branje in pisanje.

9.4. Opis programa za množenje matrik

Izvorna koda programa, s katerim sem izvajal meritve, se nahaja v prilogi. Najprej se izvede MatrixMul.cu, ki pokliče funkcijo runTest, znotraj katere se izvaja množenje matrik in meritve.

Izvajanje programa se začne z izvedbo izvorne kode, ki se nahaja v datoteki matrixMul.cu. Na začetku se naložijo vse potrebne knjižnice, potrebne za izvedbo, vključno z matrixMul_kernel.cu. V tej datoteki so definirane funkcije in konstante, specifične za CUDA okolje.

Najprej se rezervira prostor v glavnem pomnilniku za matriki A in B, ki ju bomo zmnožili,

```
// ALOCIRAMO POMNILNIK NA CPE (host memory) ZA MATRIKI A in B
unsigned int size_A = WA * HA;
unsigned int mem_size_A = sizeof(float) * size_A;
float* h_A = (float*) malloc(mem_size_A);
unsigned int size_B = WB * HB;
unsigned int mem_size_B = sizeof(float) * size_B;
float* h_B = (float*) malloc(mem_size_B);
```

in ta prostor napolnimo z naključnimi vrednostmi.

```
randomInit(h_A, size_A);
randomInit(h_B, size_B);

// v tej funkciji napolnimo matrike z naključnimi vrednostmi
void randomInit(float* data, int size)
{
    for (int i = 0; i < size; ++i)
        data[i] = rand() / (float)RAND_MAX;
}
```

Rezerviramo pomnilnik na GPE za matriki A in B

```
// allocate device memory
float* d_A;
CUDA_SAFE_CALL(cudaMalloc((void**) &d_A, mem_size_A));//v d_A se shrani
  // kje je ta matrika
float* d_B;
CUDA_SAFE_CALL(cudaMalloc((void**) &d_B, mem_size_B));
```

in vrednosti iz glavnega pomnilnika, ki smo jih prej inicializirali, prepišemo v pomnilnik GPE.

```
    // copy host memory to device
    CUDA_SAFE_CALL(cudaMemcpy(d_A, h_A, mem_size_A,
                              cudaMemcpyHostToDevice) );
    CUDA_SAFE_CALL(cudaMemcpy(d_B, h_B, mem_size_B,
                              cudaMemcpyHostToDevice) );
```

Sedaj rezerviramo prostor za matriko C v pomnilniku GPE in v glavnem pomnilniku, saj bomo meritve na CPE in GPE izvajali ločeno.

```
    // allocate device memory for result
    unsigned int size_C = WC * HC;//to je definirano v matrixMul.h (80 x
128)
    unsigned int mem_size_C = sizeof(float) * size_C;//4x80x128 byte-ov
    float* d_C;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_C, mem_size_C));

    // allocate host memory for the result
    float* h_C = (float*) malloc(mem_size_C);
```

Pred izračuni nastavimo še parametre, ki določajo število niti in koliko niti se bo tvorilo pri enem klicu. Ti parametri se nahajajo v datoteki MatrixMul.h.

```
#define BLOCK_SIZE 10 //originalno 16, ta parameter direktno vpliva na
velikost matrike

// Matrix dimensions
// (chosen as multiples of the thread block size for simplicity)
#define WA (9 * BLOCK_SIZE) // Matrix A width = 48
#define HA (9 * BLOCK_SIZE) // Matrix A height = 80
#define WB (9 * BLOCK_SIZE) // Matrix B width = 128
#define HB WA  // Matrix B height = 48
#define WC WB  // Matrix C width  = 128
#define HC HA  // Matrix C height = 80
```

Konstanta BLOCK_SIZE je pomnožena z nekim številom in tako določa velikosti vseh matrik. S spreminjanjem vrednosti v tej datoteki sem določal velikost matrik, nad katerimi sem izvajal množenja. Zaradi enostavnosti pri prikazu rezultatov sem določil, da so bile matrike dimenzij NxN, kar pomeni, da so imele enako število stolpcev in vrstic.

Potem začnemo meriti čas in pokličemo funkcijo, ki izvede množenje matrik na GPE:

```
matrixMul<<< grid, threads >>>(d_C, d_A, d_B, WA, WB);
```

Števec ustavimo in izpišemo koliko časa se je množenje izvajalo. Potem rezultate iz pomnilnika GPE prepiše v glavni pomnilnik.

Začnemo meriti čas in pokličemo program computeGold, ki izvede množenje matrik na CPE. Ta program je preprost in obsega le nekaj vrstic programske kode. V treh for zankah se premikamo čez elemente matrik in izvajamo množenje:

```
        for (unsigned int i = 0; i < hA; ++i)
            for (unsigned int j = 0; j < wB; ++j) {
                double sum = 0;
                for (unsigned int k = 0; k < wA; ++k) {
```

```
                double a = A[i * wA + k];
                double b = B[k * wB + j];
                sum += a * b;
            }
            C[i * wB + j] = (float)sum;
        }
```

Na koncu merjenje ustavimo in izpišemo čas izvajanja. Rezultate množenj na GPE in CPE primerjamo in v primeru prevelikih odstopanj opozorimo na napako ter rezultate izpišemo. Odstopanje je bilo lahko do 1.0E-6, kar ustreza natančnosti 32-bitnih števil po standardu IEEE 754.

Na koncu izvajanja sprostimo ves zasedeni pomnilnik

```
// SPROSTIMO ZASEDEN POMNILNIK
free(h_A);
free(h_B);
free(h_C);
free(reference);
CUDA_SAFE_CALL(cudaFree(d_A));
CUDA_SAFE_CALL(cudaFree(d_B));
CUDA_SAFE_CALL(cudaFree(d_C));
```

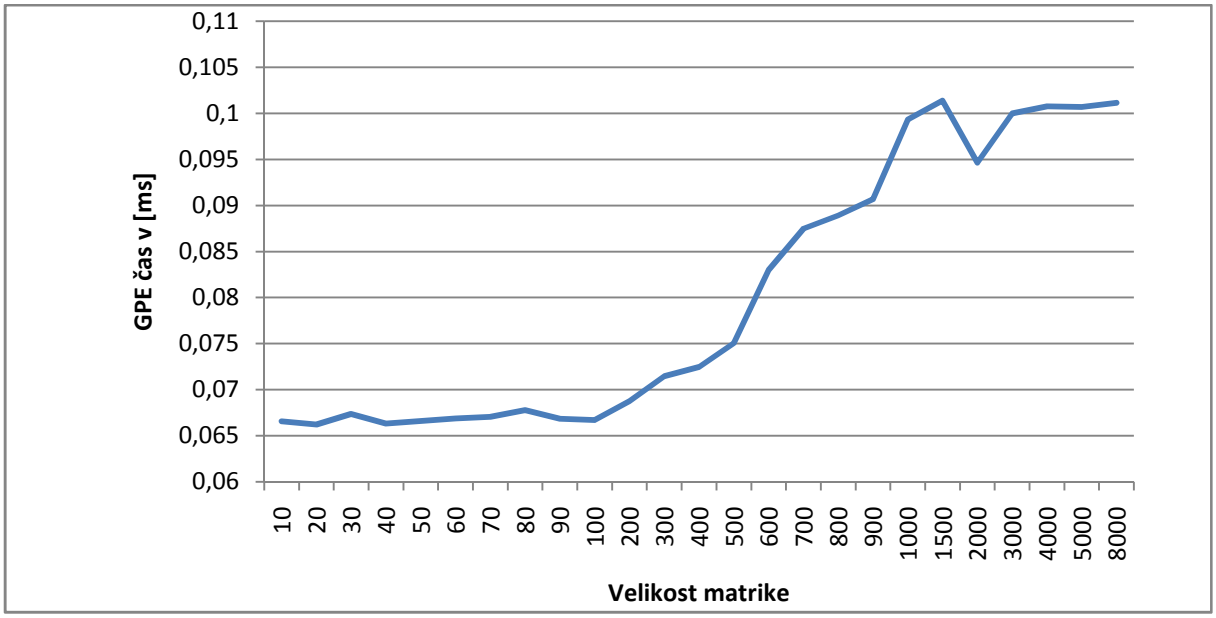
9.5. Rezultati, analiza

Z meritvami sem izgubil precej časa, saj so bili nekateri časi v urah. Naredil sem več ponovitev, število ponovitev sem smiselno izbral v odvisnosti od izvajalnega časa. Kjer so bili izvajalni časi krajši, sem v zanki ponovil izvajanje sto-krat, medtem ko sem meritve pri večjih časih ponovil desetkrat.

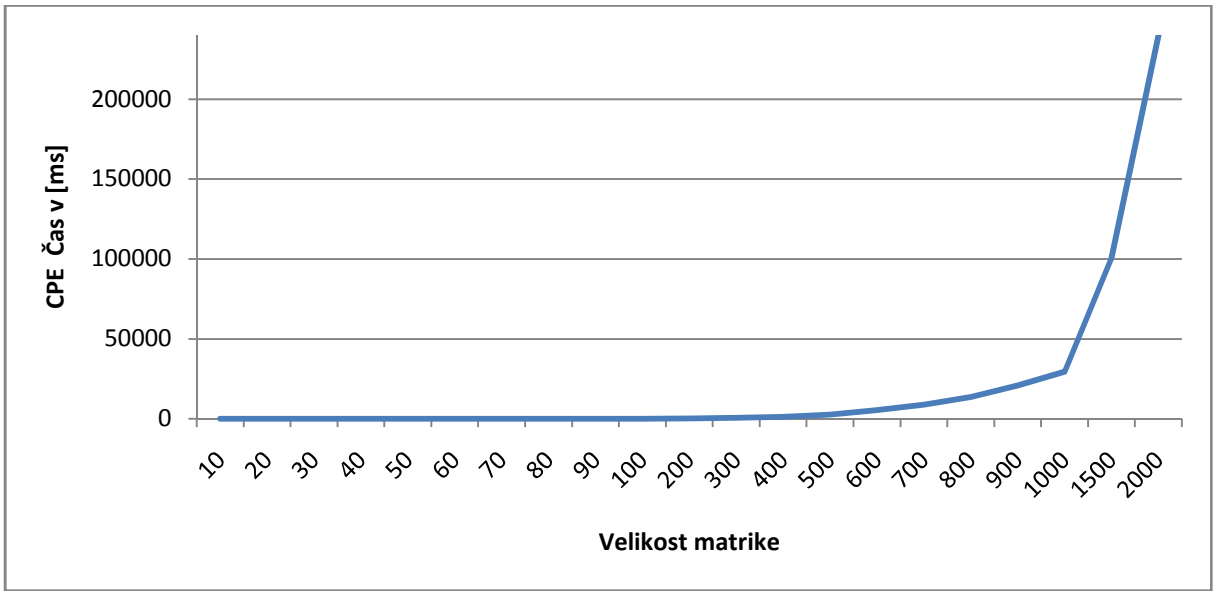
Velikost matrike	izvajanje na GPE (ms)	izvajanje na CPE (ms)
10	0,066574	0,023094
20	0,066213	0,155317
30	0,067364	0,517526
40	0,066315	1,246854
50	0,066593	2,373485
60	0,066886	4,11
70	0,067052	6,592
80	0,067789	9,824
90	0,066838	13,681
100	0,0667	18,857731
200	0,068736	150,2
300	0,071469	508,2
400	0,072458	1210
500	0,075055	2642
600	0,083	5485
700	0,0875	8880
800	0,088893	13637
900	0,090664	20856
1000	0,099322	29409
1500	0,101386	100549
2000	0,094656	240139
3000	0,1000014	10min
4000	0,100746	55min
5000	0,100686	130min
8000	0,101153	/

Tabela 2: Čas izvajanja v odvisnosti od velikosti matrike. Množenje je potekalo na dveh enakih matrikah, ki sta imeli enako število stolpcev in vrstic. Podana je ena od dveh enakih dimenzij dvodimenzionalne matrike.

Iz tabele 2 in grafa 1 lahko razberemo, da so bili časi izvajanja programa na GPE med 0.066 ms za matrike od dimenzij 1 do 100, nato se je izvajalni čas povečal do 0.10 ms za matrike velikosti 8000. Pri večjih matrikah izvajanje ni bilo mogoče, saj je program javil napako zaradi prekoračitve pomnilnika na grafični kartici. Pomnilnika na grafični kartici je bilo 896MB.

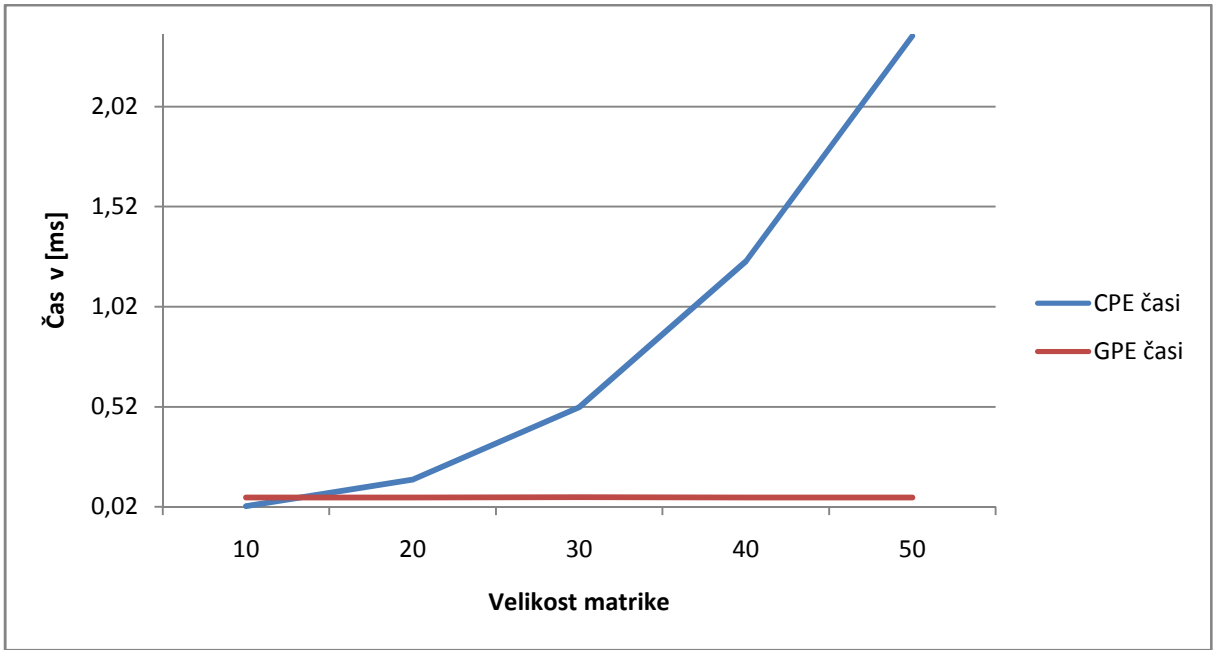


Graf 1: Izvajalni časi na GPE.



Graf 2: Izvajalni časi na CPE.

Na grafu 2 vidimo čase izvajanja programa na CPE. Pri najmanjših matrikah so bili izvajalni časi 0.02ms, kar je hitreje kot na GPE. Pri matriki 100x100 je bil čas izvajanja že 18ms, kar je 280-krat počasneje od izvajalnega časa na GPE pri isti velikosti matrike. Pri večjih matrikah je čas rastel polinomsko in pri matriki 1000x1000 je bil že 30sek. Pri isti velikosti matrike je bil čas izvajanja na GPE 0.099ms. Razlika v časih izvajanja med CPE in GPE je pri velikosti matrik 1000x1000 kar 300.000-kratna.



Graf 3: Izvajalni časi na CPE in na GPE. Pri teh dimenzijah matrik je vidno, da je CPE pri majhnih matrikah hitrejša. Pri CPE se izvajalni časi povečujejo polinomsko, pri GPE pa ostajajo skoraj konstantni.

Na grafu 3 sem prikazal rezultate množenja matrik do velikosti 50x50. Na tem grafu je vidno, da je CPE pri množenju manjših matrik hitrejša od GPE. Če bi prikazal rezultate do 200x200 ali 5000x5000, razlika ne bi bila dobro vidna, saj bi bili časi izvajanja na GPE prikazani skoraj na abscisni osi. Čas izvajanja na GPE se je v primerjavi s časom na CPE spreminjal zanemarljivo malo.

Na takšne velike razlike pri rezultatih vpliva arhitektura procesorja in način, kako je napisan program. GPE NVIDIA GTX260 je močno paralelna, ima 216 procesorjev, od katerih vsak lahko hkrati izvaja 96 niti. Hkrati se lahko izvaja več kot 20.000 množenj. Niti so majhne in za izvajanje potrebujejo le nekaj ciklov, kar pripelje do takšnih rezultatov, da se tudi pri največjih matrikah množenje izvede v desetinki milisekunde. Ko sem povečeval velikost matrik nad 8000x8000, je prišlo do napak, verjetno zaradi prezasedenosti pomnilnika v grafični kartici.

Pri CPE sem naletel na drugačne težave. Časi izvajanja so strmo rasli pri vseh velikostih matrik. Tako sem za izračun največjih matrik potreboval nekaj ur. Program, ki se je izvajal na CPE, je bil zapisan v treh zankah. Program se izvaja zaporedno, kar je vzrok za slabše rezultate. Za razporejanje procesov v glavnem procesorju je zadolžen operacijski sistem in niti so zaradi tega velike. Izvajanje preprostih operacij traja na tisoče ciklov, kar pri velikih matrikah pripelje do zaključka, da potrebujemo drugačen pristop. To je lahko drugačen način programiranja in drugačna zasnova procesorjev.

10. Sklepne ugotovitve

Med izdelavo diplomske naloge sem se srečal s številnimi izzivi. Iskanje podatkov je bilo včasih težavno, ker sem naletel na različne specifikacije. Prav to me je vzpodbudilo k iskanju zanesljivo točnih podatkov. Največ literature sem našel na medmrežju, ostalo v knjigah. Spoznal sem, da je GPGPU uporabna stvar in je relativno lahka za uporabo. Orodja so dostopna in z nekaj osnovnega razumevanja strojne opreme ter programiranja je nadaljnje delo preprosto.

Strojna oprema je zelo dostopna in cene grafičnih kartic nenehno padajo, saj prihajajo na tržišče vedno nove generacije grafičnih kartic z novimi izboljšavami. Zmogljivost se že vrsto let hitro povečuje in kot vse kaže se bo ta trend še nadaljeval. Mislim, da se bo v prihodnje vse bolj izkoriščalo računsko moč grafičnih kartic.

GPGPU se uporablja v številnih aplikacijah in za različne namene. Orodje CUDA je najbolj razširjeno, zato najdemo o njem tudi največ podatkov. Pohitritve pri splošnih aplikacijah so večje od 10 do 300 krat. Pri programu množenja matrik so razlike še izrazitejše. Pri tem algoritmu je kodo enostavno prilagoditi za paralelno procesiranje. Milslim, da je GPGPU smiselno uporabljati za reševanje problemov, saj lahko pri določenih aplikacijah dosežemo velike pohitritve.

Z izdelavo tega dela sem obnovil in razširil svoje znanje programiranja. Pri programu sem spoznal klice, specifične za okolje CUDA. Algoritme sem opremil z zajemom časa, tako da sem lahko primerjal meritve pri različnih parametrih.

Pri našem primeru nastanejo ogromne razlike v času izvajanja programa. To je zato, ker CPE porabi na tisoče ciklov samo za razvrščanje, CUDA pa razbije algoritem na tisoče niti, ki se na GPE izvajajo povsem drugače kot na CPE. Program se na GPE izvaja vzporedno, na CPE pa vzporedno. Niti, ki se izvajajo na GPE so majhne, za njihovo izvajanje se porabi le nekaj urinih ciklov. Poleg tega se lahko hkrati izvaja na stotine niti. To so glavni razlogi, da so razlike v izvajanju programa za množenje matrik tako velike.

Vsem, ki jih programiranje grafičnih kartic zanima, priporočam preizkus orodja CUDA. Namestitev je preprost in vključuje nekaj primerov programov. Tako lahko kmalu vidimo prve rezultate – pohitritve izvajanja na GPE v realnih aplikacijah.

11. Priloge

11.1. Program: MatrixMul.cu

```
// includes, system
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
// includes, project
#include <cutil.h>
// includes, kernels
#include <matrixMul_kernel.cu>
////////////////////////////////////////////////////////////////////////////////
//////
// declaration, forward
void runTest(int argc, char** argv);
void randomInit(float*, int);
void printDiff(float*, float*, int, int);

extern "C"
void computeGold(float*, const float*, const float*, unsigned int, unsigned
int, unsigned int);
////////////////////////////////////////////////////////////////////////////////
// MAIN PROGRAM
////////////////////////////////////////////////////////////////////////////////
int
main(int argc, char** argv)
{
    printf("/////////////////////////////////////////////\n");
    printf("//    Mnozenje matrik - izvajalni casi     //\n");
    printf("/////////////////////////////////////////////\n\n");
    runTest(argc, argv);

    CUT_EXIT(argc, argv);// CUDA function - displays Press ENTER to exit...
}
////////////////////////////////////////////////////////////////////////////////
// Simple CUDA TEST
////////////////////////////////////////////////////////////////////////////////
void
runTest(int argc, char** argv)
{
    CUT_DEVICE_INIT(argc, argv);

    // set seed for rand()
    srand(2006);

    // ALOCATE GPE MEMORY(host memory) for A, B
    unsigned int size_A = WA * HA;
    unsigned int mem_size_A = sizeof(float) * size_A;
    float* h_A = (float*) malloc(mem_size_A);
    unsigned int size_B = WB * HB;
    unsigned int mem_size_B = sizeof(float) * size_B;
    float* h_B = (float*) malloc(mem_size_B);

    // INICIALIZe MAIN MEMORY (host memory) // with z random values
    randomInit(h_A, size_A);
```

```
    randomInit(h_B, size_B);

    // allocate device memory
    float* d_A;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_A, mem_size_A));//pointer of
matrix is in d_A
    float* d_B;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_B, mem_size_B));

    // copy host memory to device
    CUDA_SAFE_CALL(cudaMemcpy(d_A, h_A, mem_size_A,
                              cudaMemcpyHostToDevice) );
    CUDA_SAFE_CALL(cudaMemcpy(d_B, h_B, mem_size_B,
                              cudaMemcpyHostToDevice) );

    // allocate device memory for result
    unsigned int size_C = WC * HC;// defined in matrixMul.h (original 80 x
128)
    unsigned int mem_size_C = sizeof(float) * size_C;//4x80x128 bytes
    float* d_C;
    CUDA_SAFE_CALL(cudaMalloc((void**) &d_C, mem_size_C));

    // allocate host memory for the result
    float* h_C = (float*) malloc(mem_size_C);


    // setup execution parameters
    dim3 threads(BLOCK_SIZE, BLOCK_SIZE);//number of threads (16x16),
defined in matrixMul.h
    dim3 grid(WC / threads.x, HC / threads.y);//velikost bloka (128 / 16,
80 / 16) (8, 5)

    printf("\n   Velikost bloka je %i    \n", BLOCK_SIZE);
    printf("   Velikost koncne matrike je %ix%i    \n", WC, HC);

////////////////////////////////////////////////////////////////////////
/////
    // CREATE AND RUN timer_GPU
    unsigned int timer_GPU = 0; // init of timer
    CUT_SAFE_CALL(cutCreateTimer(&timer_GPU)); // stop timer
    CUT_SAFE_CALL(cutStartTimer(timer_GPU)); // start counting time

    // execute the kernel // main GPU funkction
    //for (int i = 0; i < 100; ++i) //loop for better results
        matrixMul<<< grid, threads >>>(d_C, d_A, d_B, WA, WB); //ta ukaz
grid threads tells GPU, how to work with this data.
//Tole bo 8x5 blokov in  v vsakem bo 256 niti. vsaka nit se bo posamezno
izvajala

    // stop and destroy timer_GPU
    CUT_SAFE_CALL(cutStopTimer(timer_GPU)); // ustavi merjenje casa
    printf("\nCas izvajanja na GPE: %f (ms) \n",
cutGetTimerValue(timer_GPU)); // izpisi cas izvajanja
    CUT_SAFE_CALL(cutDeleteTimer(timer_GPU)); // skenslaj timer_GPU
////////////////////////////////////////////////////////////////////////
/////

    // check if kernel execution generated and error
    CUT_CHECK_ERROR("Kernel execution failed");
```

```
    // KOPIRANJE REZULTATOV IZ GPE V GLAVNI POMNILNIK (tukaj se najprej
pojavi napaka pri povečevanju matrik)
    CUDA_SAFE_CALL(cudaMemcpy(h_C, d_C, mem_size_C,
                              cudaMemcpyDeviceToHost) );
////////////////////////////////////////////////////////////////////////
      // DODALI timer_CPU
    unsigned int timer_CPU = 0; // spremenljivka timer
    CUT_SAFE_CALL(cutCreateTimer(&timer_CPU)); // ustvari timer
    CUT_SAFE_CALL(cutStartTimer(timer_CPU)); // zacni meriti cas

    // compute reference solution
    float* reference = (float*) malloc(mem_size_C);
    //for (int i = 0; i < 100; ++i)
     computeGold(reference, h_A, h_B, HA, WA, WB);//dimenzije matrike
(zadnje 3) // glavna CPU funkcija

      // dodal ivan timer_CPU
    CUT_SAFE_CALL(cutStopTimer(timer_CPU)); // ustavi merjenje casa
    printf("Cas izvajanja na CPE: %f (ms) \n",
cutGetTimerValue(timer_CPU)); // izpisi cas izvajanja
    CUT_SAFE_CALL(cutDeleteTimer(timer_CPU)); // skenslej timer

    // PREVERIMO REZULTATE
    CUTBoolean res = cutCompareL2fe(reference, h_C, size_C, 1e-
6f);//originalno je bilo 1e-6f
    printf("Testiranje %s \n", (1 == res) ? "USPESNO" : "NI USPESNO \n");
    if (res!=1) printDiff(reference, h_C, WC, HC);

    // SPROSTIMO ZASEDEN POMNILNIK
    free(h_A);
    free(h_B);
    free(h_C);
    free(reference);
    CUDA_SAFE_CALL(cudaFree(d_A));
    CUDA_SAFE_CALL(cudaFree(d_B));
    CUDA_SAFE_CALL(cudaFree(d_C));
}

// NAPOLNIMO MATRIKE Z NAKLJUČNIMI VREDNOSTMI TIPA float
void randomInit(float* data, int size)
{
    for (int i = 0; i < size; ++i)
        data[i] = rand() / (float)RAND_MAX;
}

void printDiff(float *data1, float *data2, int width, int height)
{
  int i,j,k;
  int error_count=0;
  for (j=0; j<height; j++) {
    for (i=0; i<width; i++) {
      k = j*width+i;
      if (data1[k] != data2[k]) {
         printf("diff(%d,%d) CPU=%4.4f, GPU=%4.4f n", i,j, data1[k],
data2[k]);
         error_count++;
      }
    }
  }
  printf(" nTotal Errors = %d n", error_count);
}
```

11.2. Program: MatrixMul.h

```
#ifndef _MATRIXMUL_H_
#define _MATRIXMUL_H_

// Thread block size
#define BLOCK_SIZE 10 //originalno 16, ta parameter direktno vpliva na
velikost matrike

// Matrix dimensions
// (chosen as multiples of the thread block size for simplicity)
#define WA (9 * BLOCK_SIZE) // Matrix A width = 48
#define HA (9 * BLOCK_SIZE) // Matrix A height = 80
#define WB (9 * BLOCK_SIZE) // Matrix B width = 128
#define HB WA  // Matrix B height = 48
#define WC WB  // Matrix C width  = 128
#define HC HA  // Matrix C height = 80

#endif // _MATRIXMUL_H_
```

11.3. Program: MatrixMul_gold.cpp

```
////////////////////////////////////////////////////////////////////////////
/////
// export C interface
extern "C"
void computeGold( float*, const float*, const float*, unsigned int,
unsigned int, unsigned int);

////////////////////////////////////////////////////////////////////////////
/////
//! Compute reference data set
//! C = A * B
//! @param C          reference data, computed but preallocated
//! @param A          matrix A as provided to device
//! @param B          matrix B as provided to device
//! @param hA         height of matrix A //48
//! @param wA         width of matrix A//80
//! @param wB         width of matrix B//128
////////////////////////////////////////////////////////////////////////////
/////
void
computeGold(float* C, const float* A, const float* B, unsigned int hA,
unsigned int wA, unsigned int wB)
{
    for (unsigned int i = 0; i < hA; ++i)
        for (unsigned int j = 0; j < wB; ++j) {
            double sum = 0;
            for (unsigned int k = 0; k < wA; ++k) {
                double a = A[i * wA + k];
                double b = B[k * wB + j];
                sum += a * b;
            }
            C[i * wB + j] = (float)sum;
        }
```

11.4. Program: MatrixMul_kernel.cu

```
#ifndef _MATRIXMUL_KERNEL_H_
#define _MATRIXMUL_KERNEL_H_

#include <stdio.h>
#include "matrixMul.h"

#define CHECK_BANK_CONFLICTS 0
#if CHECK_BANK_CONFLICTS
#define AS(i, j) CUT_BANK_CHECKER(((float*)&As[0][0]), (BLOCK_SIZE * i +
j))
#define BS(i, j) CUT_BANK_CHECKER(((float*)&Bs[0][0]), (BLOCK_SIZE * i +
j))
#else
#define AS(i, j) As[i][j]
#define BS(i, j) Bs[i][j]
#endif

////////////////////////////////////////////////////////////////////////////
/////
//! Matrix multiplication on the device: C = A * B
//! wA is A's width and wB is B's width
////////////////////////////////////////////////////////////////////////////
/////
__global__ void
matrixMul( float* C, float* A, float* B, int wA, int wB)
{
    // Block index//koordinata bloka
    int bx = blockIdx.x;
    int by = blockIdx.y;

    // Thread index //koordinata niti
    int tx = threadIdx.x;
    int ty = threadIdx.y;

    // Index of the first sub-matrix of A processed by the block//blok je
podmatrika
    int aBegin = wA * BLOCK_SIZE * by;

    // Index of the last sub-matrix of A processed by the block
    int aEnd   = aBegin + wA - 1;

    // Step size used to iterate through the sub-matrices of A
    int aStep  = BLOCK_SIZE;

    // Index of the first sub-matrix of B processed by the block
    int bBegin = BLOCK_SIZE * bx;

    // Step size used to iterate through the sub-matrices of B
    int bStep  = BLOCK_SIZE * wB;

    // Csub is used to store the element of the block sub-matrix
    // that is computed by the thread
    float Csub = 0;//tukaj bo shranjen rezultat

    // Loop over all the sub-matrices of A and B
    // required to compute the block sub-matrix
    for (int a = aBegin, b = bBegin;
             a <= aEnd;
```

```
             a += aStep, b += bStep) {

        // Declaration of the shared memory array As used to
        // store the sub-matrix of A
        __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];

        // Declaration of the shared memory array Bs used to
        // store the sub-matrix of B
        __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];

        // Load the matrices from device memory
        // to shared memory; each thread loads
        // one element of each matrix
        AS(ty, tx) = A[a + wA * ty + tx];//nalozi iz pomnilnika
        BS(ty, tx) = B[b + wB * ty + tx];

        // Synchronize to make sure the matrices are loaded//casovno
sinhronizira
        __syncthreads();

        // Multiply the two matrices together;
        // each thread computes one element
        // of the block sub-matrix
        for (int k = 0; k < BLOCK_SIZE; ++k)//pomnozi
            Csub += AS(ty, k) * BS(k, tx);

        // Synchronize to make sure that the preceding
        // computation is done before loading two new
        // sub-matrices of A and B in the next iteration
        __syncthreads();
    }

    // Write the block sub-matrix to device memory;
    // each thread writes one element
    int c = wB * BLOCK_SIZE * by + BLOCK_SIZE * bx;
    C[c + wB * ty + tx] = Csub;
}

#endif // #ifndef _MATRIXMUL_KERNEL_H_
```

12. Literatura

[1] David A. Patterson, Jonh L. Hennessy, »Computer Organization And Design«, 2008, The Processor, str. 300-344; Graphics and Computing GPUs, str. A-1 – A-20.

[2] Dušan Kodek, »Arhitektura in organizacija računalniških sistemov«, 2008, Centralna procesna enota, str. 157-174

[3] (2009) Wikipedia, the free encyclopedia
http://en.wikipedia.org/wiki/Main_Page

[4] (2005) IEEE Information for authors. Dostopno na:
http://www.ieee.org/portal/cms docs/pubs/transactions/auinfo03.pdf

[5] CUDA Zone – The resource for CUDA developers, dostopno na
http://www.nvidia.com/object/cuda_home.html

[6] Dr. Dobb's, CUDA, Supercomputing for the Masses, dostopno na:
http://www.ddj.com/hpc-high-performance-computing/207200659

[7] Intel Core Architecture, dostopno na:
http://www.intel.com/technology/architecture-silicon/core/

[8] Learn more about NVIDIA SLI, dostopno na:
http://www.slizone.com/page/slizone_learn.html

[9] NVIDIA CUDA Programming Guide, dostopno na:
http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf