
Laborprotokoll

GPGPU

Systemtechnik Labor
5BHITT 2015/16, Gruppe C

Patrick Malik & Thomas Taschner

Note:
Betreuer: Michael Borko

Version 1.0
Begonnen am 15. Januar 2016
Beendet am 24. Januar 2016

Inhaltsverzeichnis

1	Einführung	1
1.1	Ziele	1
1.2	Aufgabenstellung	1
1.3	Links	1
2	Ergebnisse	2
2.1	Nutzung von GPUs in normalen Desktop-Anwendungen	2
2.1.1	Anwendungen	2
2.2	Vorteile der GPU bei rechenintensiven Anwendungen	2
2.3	Entwicklungsumgebungen und Programmiersprachen	3
2.4	Transkompilieren	3
2.5	Benchmarks	3
2.5.1	Specs	3
2.5.2	Setup	5
2.5.3	Ergebnisse	7
2.5.4	Interpretation	9
2.5.5	Handbrake Fazit	9

1 Einführung

1.1 Ziele

Die Aufgabe beinhaltet eine Recherche über grundsätzliche Einsatzmöglichkeiten für GPGPU. Dabei soll die Sinnhaftigkeit der Technologie unterstrichen werden. Die Fragestellungen sollen entsprechend mit Argumenten untermauert werden.

Im zweiten Teil der Arbeit soll der praktische Einsatz von OpenCL trainiert werden. Diese können anhand von bestehenden Codeexamples durchgeführt werden. Dabei wird auf eine sprechende Gegenüberstellung (Benchmark) Wert gelegt.

Die Aufgabenstellung soll in einer Zweiergruppe bearbeitet werden.

1.2 Aufgabenstellung

Informieren Sie sich über die Möglichkeiten der Nutzung von GPUs in normalen Desktop-Anwendungen. Zeigen Sie dazu im Gegensatz den Vorteil der GPUs in rechenintensiven Implementierungen auf [1Pkt]. Gibt es Entwicklungsumgebungen und in welchen Programmiersprachen kann man diese nutzen [1Pkt]?

Können bestehende Programme (C/C++ und Java) auf GPUs genutzt werden und was sind dabei die Grundvoraussetzungen dafür [1Pkt]? Gibt es Transpiler und wie kommen diese zum Einsatz [1Pkt]?

Präsentieren Sie an einem praktischen Beispiel den Nutzen dieser Technologie. Wählen Sie zwei rechenintensive Algorithmen (z.B. Faktorisierung) und zeigen Sie in einem aussagekräftigen Benchmark welche Vorteile der Einsatz der vorhandenen GPU Hardware gegenüber dem Ausführen auf einer CPU bringt (OpenCL). Punkteschlüssel:

Auswahl und Argumentation der zwei rechenintensiven Algorithmen (Speicher, Zugriff, Rechenoperationen) [0..4Pkt]

Sinnvolle Gegenüberstellung von CPU und GPU im Benchmark [0..2Pkt]

Anzahl der Durchläufe [0..2Pkt]

Informationen bei Benchmark [0..2Pkt]

Beschreibung und Bereitstellung des Beispiels (Ausführbarkeit) [0..2Pkt]

1.3 Links

<https://github.com/ReneHollander/opencl-example>

2 Ergebnisse

2.1 Nutzung von GPUs in normalen Desktop-Anwendungen

Im privaten Bereich gibt es praktisch keine Anwendung für GPGPU. Ausnahmen sind Programme für die Videobearbeitung. Typische Alltagssoftware lässt sich kaum parallelisieren. Im Bereich der Wissenschaft und Technik sieht es dann schon wieder anders aus. Hier machen mehrere GPGPU-taugliche Grafikkarten einen normalen PC zu einem Supercomputer. Voraussetzung dafür ist ein hochleistungsfähiger Grafikprozessor, der bis zu 800 Kerne hat und parallel arbeitet. Damit ein Grafikprozessor überhaupt sinnvoll genutzt werden kann, muss die Anwendung eine sehr hohe Zahl von parallel ausführbaren Berechnungen enthalten. Wenn man die Rechenwerke eines Grafikprozessors kontinuierlich auslasten will, dann sollte man mindestens eine Millionen gleichartiger Berechnungen liefern können, sonst hat man gegenüber einer normalen CPU keine Vorteile. Die GPU-Rechenwerke müssen mit einem kontinuierlichen Datenstrom versorgt werden. Aus diesem Grund bezeichnet man sie auch als Stream-Prozessoren. [1] Zu Ergänzen ist die erste Aussage auch noch mit Mathematikprogrammen wie Matlab oder ähnlichem. [2]

2.1.1 Anwendungen

- Physikalisch Simulationen (Strömung, Gravitation, Temperatur, Crash-Tests)
- Komplexe Klimamodelle (Wettervorhersage)
- Datenanalysen und Finanzmathematik
- Verarbeitung von akustischen und elektrischen Signalen
- CT- und Ultraschall-Bildrekonstruktion
- Modellieren von Molekülstrukturen
- Kryptografie
- Neuronale Netze
- Videotranskodierung

2.2 Vorteile der GPU bei rechenintensiven Anwendungen

GPUs haben den Vorteil, dass sie auf spezielle Arten von Prozessen optimiert sind. CPUs sind hingegen darauf ausgelegt beliebige Aufgaben zu verarbeiten. Um rechenintensive Anwendungen auch tatsächlich auf der GPU laufen lassen zu können, müssen/sollten diese gut parallelisierbar sein. Im Vergleich zur CPU hat man wesentlich mehr Kerne auf die parallelisiert werden kann.

2.3 Entwicklungsumgebungen und Programmiersprachen

Von Nvidia wird unter anderem eine Visual Studio Abwandlung namens NSight zur Verfügung gestellt, welche es einem ermöglicht in CUDA C/C++, OpenCL, DirectCompute, Direct3D, and OpenGL zu programmieren.

OpenCL Studio eröffnet einem die Möglichkeit mit OpenCL und OpenGL zu arbeiten.

NVidia CUDA Toolkit bietet einem die Möglichkeit CUDA Applikationen in C und C++ zu programmieren. Applikationen für OpenCL werden zum Großteil in C oder C++ programmiert. OpenCL C/C++ bieten die Option direkt für OpenCL zu programmieren. Bei der APP(Accelerated Parallel Processing) SDK von AMD, handelt es sich um eine Technologie, welche es einem ermöglicht die GPU neben dem Verarbeiten von Grafiken auch für weitere Dinge zu verwenden. Mit der APP-SDK kann unter anderem in Visual Studio programmiert werden. [3]

Bei C++ AMP (Accelerated Massive Parallelism) handelt es sich um ein Programmiermodell, welches die Programmiersprache C++ um eine Library und die Fähigkeit Code auf die GPU auszulagern erweitert. Es setzt auf DirectX 11 auf. [4]

2.4 Transkompilieren

Um bestehende Applikationen auf GPUs sinnvoll zu nutzen, müssen diese prinzipiell parallelisierbar sein (unabhängige Teile). Zudem sollte das Programm einfach skalieren, es sollte für bspw. das 10-fache an Arbeit nicht das 15-fache an Ressourcen benötigen. Threads die auf GPUs laufen sollten im Allgemeinen eher kurz gehalten sein und möglichst wenige Verzweigungen besitzen, da eine stetige Synchronisierung von Nöten ist. [5]

CUDA LLVM bietet einem die Möglichkeit für eine neue oder bereits existierende Sprache ein Frontend zu schreiben, welches diese in die Internal Representation von LLVM umwandelt. Viele Frontends sind bereits vorhanden, siehe: [6] [7]

Rootbeer ist ein GPUCompiler, welcher Javacode auf der GPU ausführen lässt, ohne dass der Code extra angepasst werden muss.[8]

2.5 Benchmarks

2.5.1 Specs

Die Spezifikationen wurden mit Hilfe von CPU-Z, GPU-Z und notebookcheck.net ermittelt. [9] [10] [11]

Laptopmodell: Lenovo Ideapad y510p

- Mainboard:
 - Name: Intel HM86 Chipset
 - Modell: LENOVO VIQY0Y1
- CPU:
 - Modell: Intel Core i7 4700MQ @ 2.40GHz
 - Kerne: 4 (virtuell 8)

Chipset: Intel Haswell

Southbridge: HM86

- RAM:

16GB DDR3

Dual Channel

Maxfrequenz: PC3-12800 (800MHz)

- GPU (2x):

Modell: Nvidia GeForce GT 755M

Architektur: GK107 (Kepler)

Busbreite: 128 Bit

VRAM: 2GB GDDR5 (Hynix)

Core (Taktfrequenz): 980MHz (Boost bis 1020MHz)

Memory (Taktfrequenz): 1350MHz

- OS

Name: Windows 10 Pro 64bit (Updates bis 22.10.2016)

Laptopmodell: MSI GT70 2OC

- Mainboard:

Name: Intel HM87 Chipset

- CPU:

Modell: Intel Core i7 4700MQ @ 2.40GHz

Kerne: 4 (virtuell 8)

Chipset: Intel Haswell

Southbridge: HM86

- RAM:

8GB DDR3L

Single Channel

Maxfrequenz: PC3-12800 (800MHz)

CL-Taktung: 11-11-11-28

- GPU 1:

Modell: Nvidia GeForce GTX 770M

Architektur: GK106 (Kepler)

Busbreite: 192 Bit

VRAM: 3GB GDDR5 (Samsung)

Core (Taktfrequenz): 706MHz (Boost bis 797MHz)

Memory (Taktfrequenz): 1002MHz

- GPU 2:

Modell: Intel HD Graphics 4600

Architektur: GT2

Busbreite: 64 Bit

VRAM: shared memory

Core (Taktfrequenz): 400MHz (Boost bis 1150MHz)

Memory (Taktfrequenz): 800MHz

- OS

Name: Windows 8 (Updates bis 22.10.2016)

2.5.2 Setup

GPUPI

Bei GPUPI handelt es sich um ein Programm österreichischer Herkunft, welches mithilfe der BPP-Formel Pi berechnet. Die Besonderheit dieses Programms, sind die Einstellungsoptionen, welche einem die Möglichkeit geben, zum Einen die Anzahl der Nachkommastellen anzugeben (1 Million bis 32 Milliarden) und zum Anderen die Möglichkeit bietet die Berechnung sowohl auf der CPU als auch auf der GPU durchzuführen. Hier wird nicht nur OpenCL, sondern auch Nvidia CUDA unterstützt (soweit unsere Erfahrungen). Da es sich hier um ein Benchmarkprogramm handelt, werden auch relevante Zeiten ausgegeben. Wir haben in unserem Test sowohl kleine Berechnungen (100Millionen), mittlere (1 Milliarde) und große (2 Milliarden) durchgeführt, um den Unterschied zu verdeutlichen. Die folgenden Grafiken bilden auf der x-Achse die durchgeführten Batches (Arbeitspakete, immer 20) ab und auf der y-Achse die Dauer in Sekunden.[12]

CompuBenchCL

Bei CompuBenchCL handelt es sich um ein Benchmarkprogramm, welches verschiedenste Arten von Algorithmen durchführt und deren Leistung beurteilt bzw. misst. Der Nachteil des Programms ist die Undurchsichtigkeit. Auch hier wird einem die Möglichkeit geboten auszuwählen ob die Tests auf der GPU oder CPU laufen sollen, was sich für unsere Zwecke gut eignet. Die folgende Grafik zeigt die verschiedenen Tests im Vergleich, in den meisten Fällen ist die GPU klar im Vorteil, jedoch beim Bitcoin Mining überraschender Weise nicht. Das Facedetection Beispiel folgt dem Viola-Jones Algorithmus, die Ocean Surface Simulation ist eine Fast Fourier Transformation, bei dem "T-Rex"Beispiel handelt es sich um einen Path tracer, welcher durch verschiedene Events und Beleuchtungen rechenintensiv wird. [13]

Handbrake

Handbrake ist eine von Eric Petit entwickelte freie Software zur Transkodierung von Videodateien. Das Programm wird in Objective-C, C und C# entwickelt und bietet ebenfalls die Möglichkeit mittels 'Intel QuickSync Video' Videos auf der GPU rendern zu lassen. [14]

Zu Testzwecken wurde eine Videodatei mit den folgenden Spezifikationen auf dem Laptop MSI GT70 2OC mit der CPU und der Intel HD Graphics 4600 Grafikkarte konvertiert:

Input

- Video
 - Auflösung: 1920x1080 Pixel
 - Bildrate: 29.970 FPS
 - Codec: AVC
 - Profil: High@L4.1
 - Dauer: 7min 59s
 - Bildseitenverhältnis: 16:9
 - Bitrate: 30.0 Mbps
 - Bitratenmodus: konstant
 - Farbraum: NTSC YUV 4:2:0
 - Farbtiefe: 8 Bit
- Audio
 - Format: PCM
 - Bitrate: 4608 kbps
 - Bitratenmodus: konstant
 - Samplingrate: 96.0 KHz
 - Bittiefe: 24 Bits
 - Kanäle: 2 (Stereo)
 - Dauer: 7min 59s

Output

- Video
 - Auflösung: 1920x1080 Pixel
 - Bildrate: 29.970 FPS
 - Codec: AVC
 - Profil: High@L4.1
 - Dauer: 7min 59s
 - Bildseitenverhältnis: 16:9
 - Bitrate: 4.0 Mbps
 - Bitratenmodus: konstant
 - Farbraum: YUV 4:2:0
 - Farbtiefe: 8 Bit

- Audio

Format: AAC HE-AAC/LC

Bitrate: 128 kbps

Bitratenmodus: variabel

Samplingrate: 48.0 KHz

Bittiefe: 16 Bits

Kanäle: 2 (Stereo)

Dauer: 7min 59s

Das Ziel ist es ein Video einer Blu-Ray in ein portableres Format zu konvertieren, welches auf Abspielgeräten, wie Smart-TVs, Spielekonsolen, aber auch leistungsfähigeren Handys abspielbar ist.

Zusätzlich zu den Angaben beim Output wurde in Handbrake ebenfalls der Parameter x264 Preset: Slow (CPU) und QSV Preset: Best Quality (GPU) eingetragen. Anzumerken ist, dass das Best Quality Preset qualitativ in etwa dem Slow Preset entspricht.

2.5.3 Ergebnisse

Die GPUPi und CompuBench Testläufe wurden am Lenovo Ideapad y510p durchgeführt. Die Handbrake Testläufe wurden am MSI GT70 2OC mit der Intel HD 4600 durchgeführt.

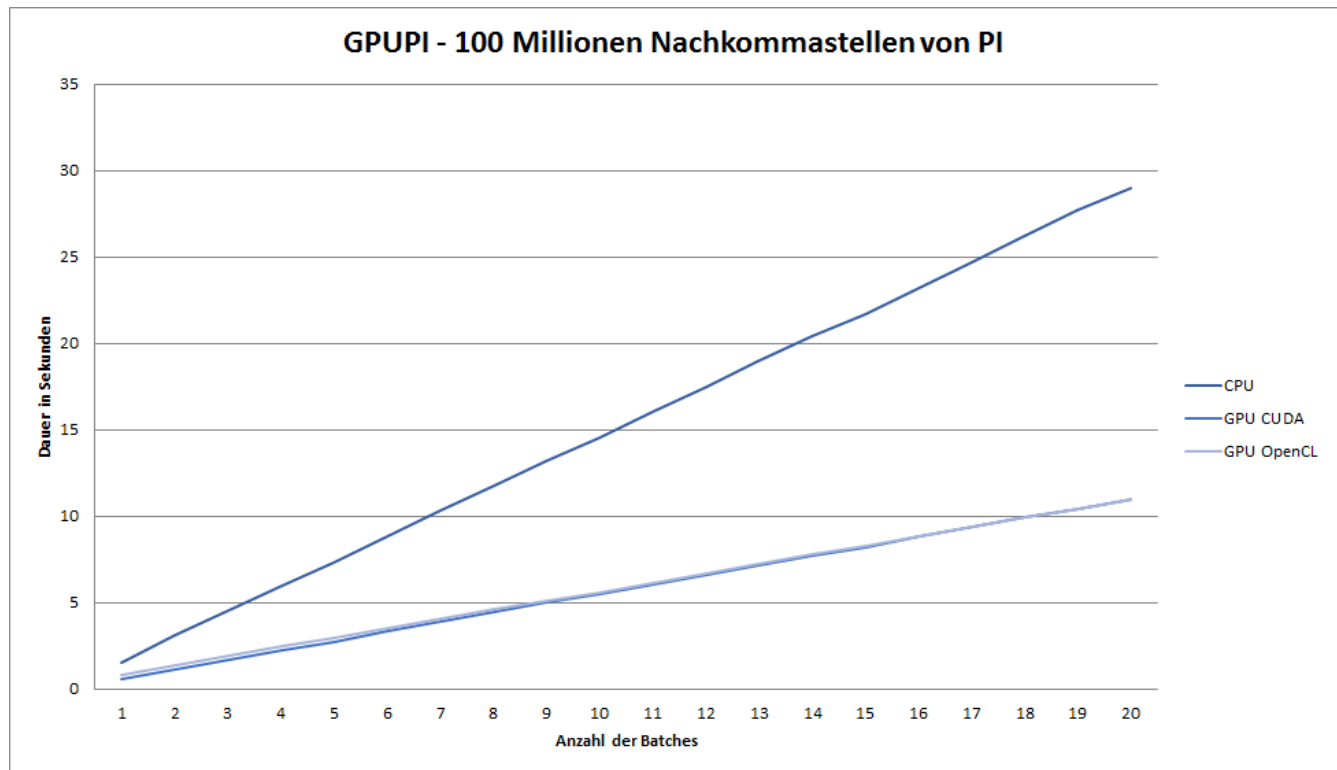


Abbildung 1: GPUPI 100 Millionen Nachkommastellen

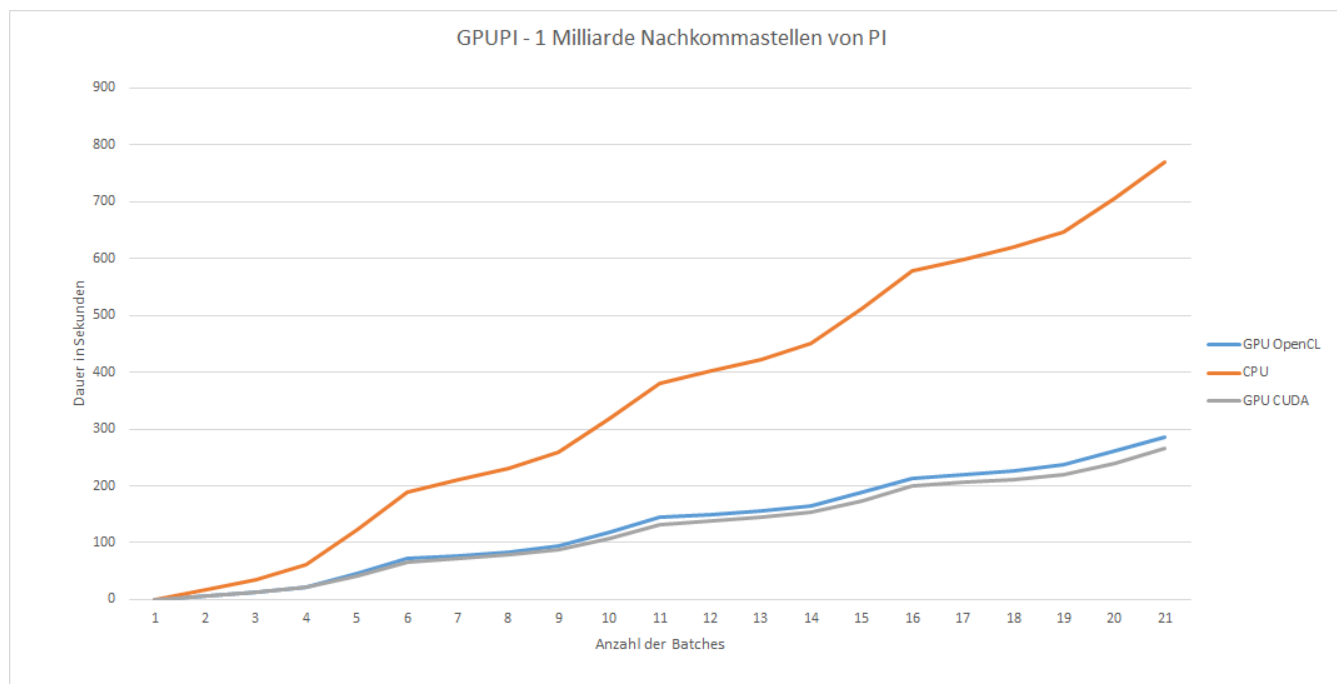


Abbildung 2: GPUPI 1 Milliarde Nachkommastellen

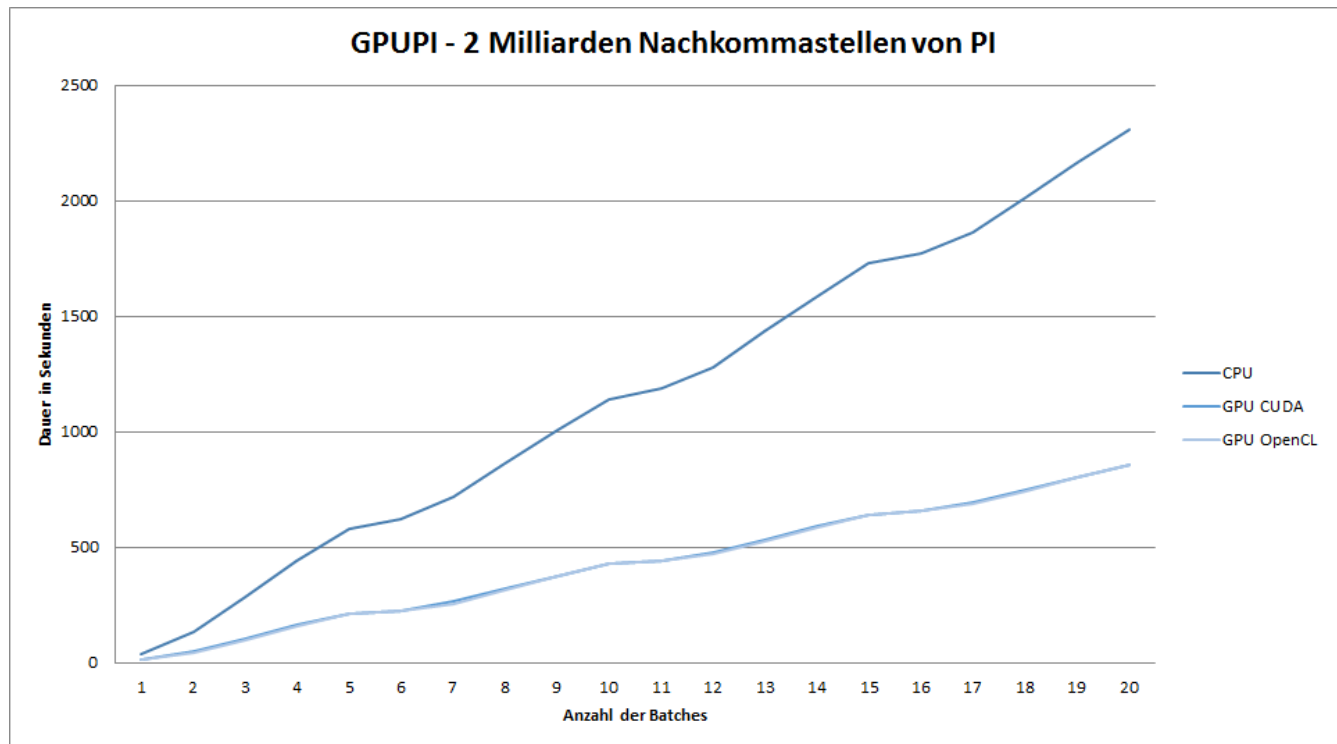


Abbildung 3: GPUPI 2 Milliarden Nachkommastellen

2.5.4 Interpretation

Es ist offensichtlich, dass die GPU bei diesem Anwendungsfall ihre Vorteile mit sich bringt. Sie liefert wesentlich schneller Ergebnisse, besonders bei der Berechnung von 2 Milliarden Nachkommastellen Pi's ist der Unterschied der Rechenzeit zwischen GPU und CPU beträchtlich. Während man auf der CPU für die Berechnung 2308s (38min 28.217s) benötigt, braucht die GPU 856.5 (14min 16.559s) und hierbei handelt es sich letztlich nur um ein Beispiel mit eher geringem Umfang.

Der Unterschied zwischen CUDA und OpenCL wird nicht zu deutlich, wobei bei kürzeren Berechnungen der Unterschied interessanterweise größer ausfällt als bei längeren.

2.5.5 Handbrake Fazit

Anhand der Grafiken (Abbildung 5 und Abbildung 6) lässt sich erkennen, dass die GPU im Schnitt 4.7x schneller hochauflösende Videos konvertieren kann, als die CPU. Die CPU hat im Schnitt 16.22 fps erreicht, während die GPU auf 76.02 fps gekommen ist. Die CPU Auslastung lag beim CPU Rendering stets bei 100%, während beim GPU Rendering eine Auslastung von ca. 20% zustande gekommen ist. Die Ursache dafür ist die Tatsache, dass in dem Fall die Bilder des Videos mit der CPU dekodiert wurden, beim den Durchläufen also ausschließlich nur die GPU am Enkodieren war. Interessanterweise dauert der erste Durchlauf in beiden Fällen um einige Sekunden kürzer.

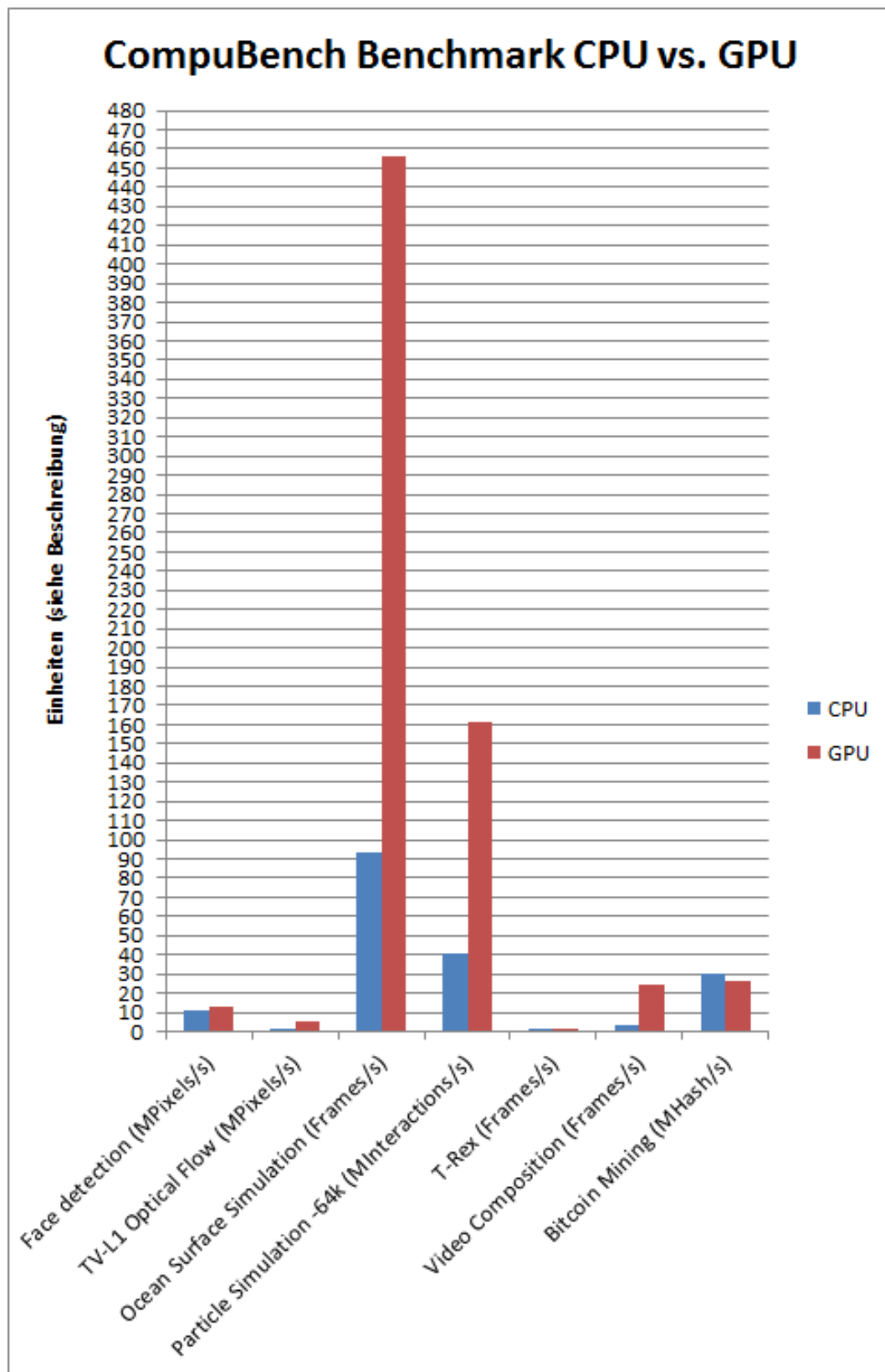


Abbildung 4: Ergebnisse des CompuBench Benchmarks

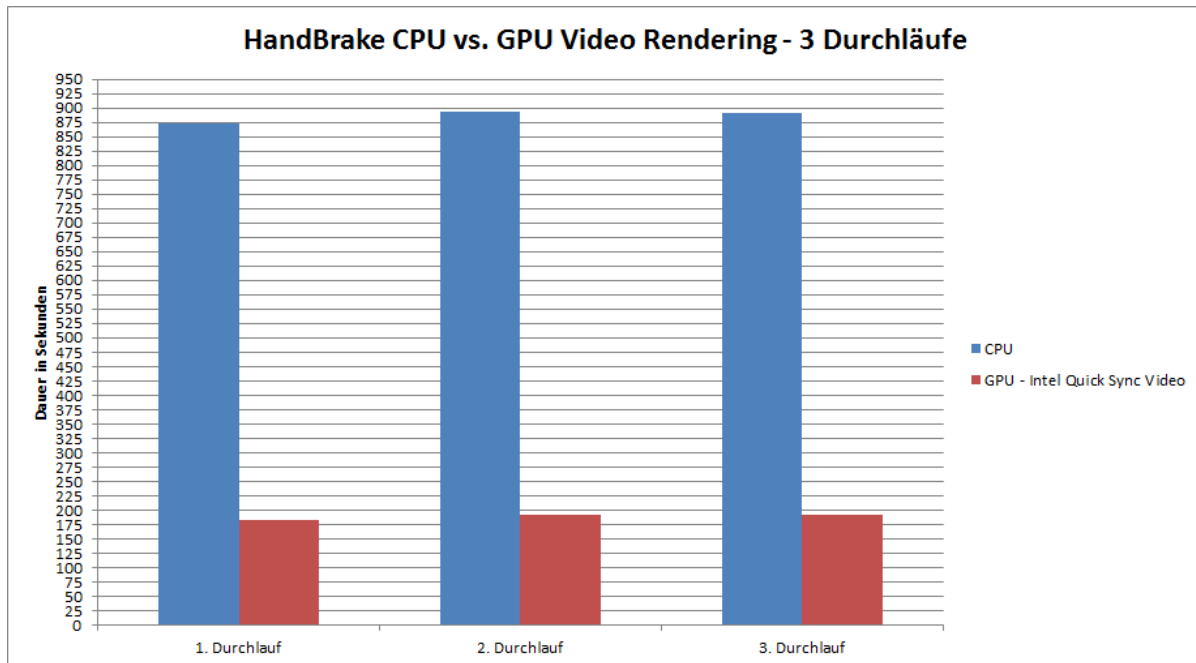


Abbildung 5: Handbrake: CPU vs. GPU - 3 Durchläufe

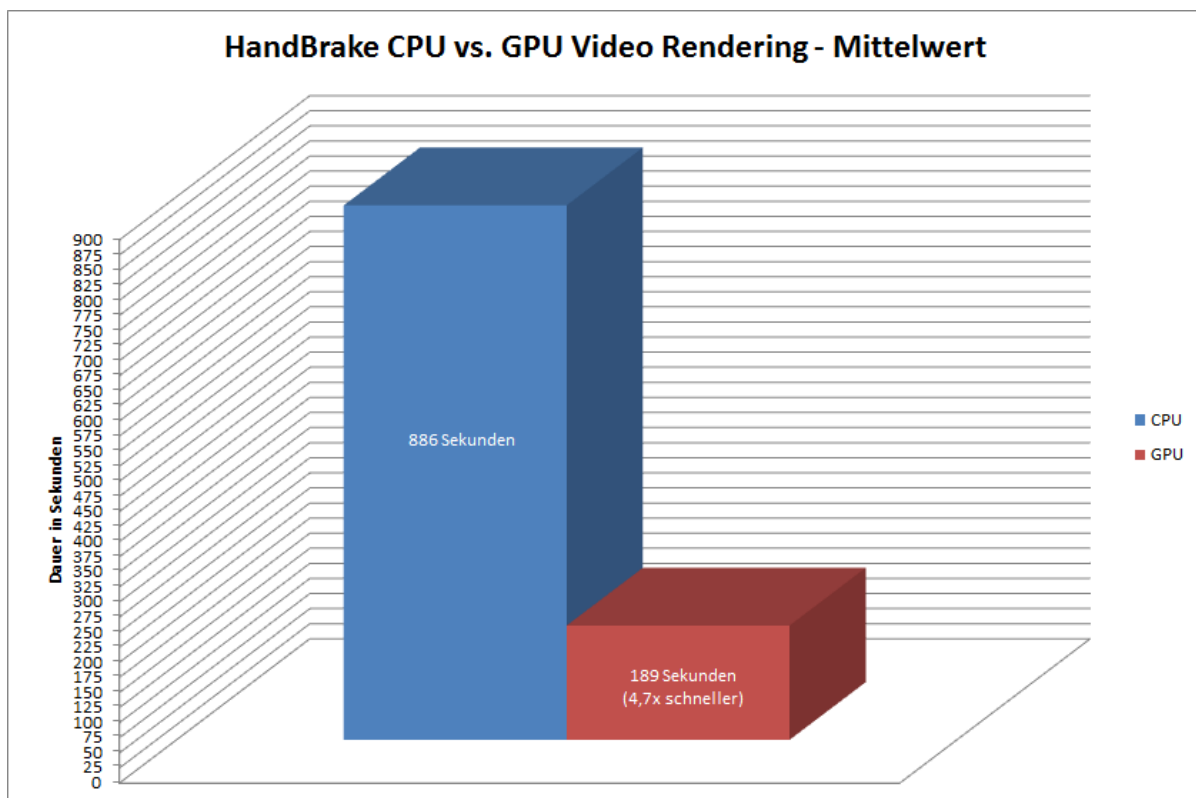


Abbildung 6: Handbrake: CPU vs. GPU - Mittelwert der Durchläufe

Literatur

- [1] gpgpu. <http://www.elektronik-kompodium.de/sites/com/1405281.htm>. Zuletzt besucht: 22.01.2016.
- [2] matlab. <http://de.mathworks.com/discovery/matlab-gpu.html>. Zuletzt besucht: 22.01.2016.
- [3] App sdk – a complete development platform. <http://developer.amd.com/tools-and-sdks/rocm-zone/amd-accelerated-parallel-processing-app-sdk/>. Zuletzt besucht: 22.01.2016.
- [4] C++ amp (c++ accelerated massive parallelism). <https://msdn.microsoft.com/de-de/library/hh265137.aspx>. Zuletzt besucht: 22.01.2016.
- [5] Grundlagen der gpu-programmierung. https://www.matse.itc.rwth-aachen.de/dienste/public/show_document.php?id=7251. Zuletzt besucht: 22.01.2016.
- [6] Llvm. <https://developer.nvidia.com/cuda-llvm-compiler>. Zuletzt besucht: 22.01.2016.
- [7] Llvm frontends. <http://llvm.org/Projects/WithLLVM/>. Zuletzt besucht: 22.01.2016.
- [8] Rootbeer gpu compiler. <https://github.com/pcpratts/rootbeer1>. Zuletzt besucht: 22.01.2016.
- [9] Cpu-z | softwares | cpuid. <http://www.cpuid.com/softwares/cpu-z.html>. Zuletzt besucht: 23.01.2016.
- [10] Gpu-z video card gpu information utility. <https://www.techpowerup.com/gpuz/>. Zuletzt besucht: 23.01.2016.
- [11] Notebook / laptop reviews and news - notebookcheck.net. <http://www.notebookcheck.net/>. Zuletzt besucht: 23.01.2016.
- [12] gpupi. <https://www.overclockers.at/news/gpupi-international-support-thread>. Zuletzt besucht: 22.01.2016.
- [13] compubench. <https://compubench.com/result.jsp?benchmark=compu15d>. Zuletzt besucht: 22.01.2016.
- [14] Handbrake: Open source video transcoder. <https://handbrake.fr/>. Zuletzt besucht: 22.01.2016.

Abbildungsverzeichnis

1	GPUPi 100 Millionen Nachkommastellen	8
2	GPUPi 1 Milliarde Nachkommastellen	8
3	GPUPi 2 Milliarden Nachkommastellen	9
4	Ergebnisse des CompuBench Benchmarks	10
5	Handbrake: CPU vs. GPU - 3 Durchläufe	11
6	Handbrake: CPU vs. GPU - Mittelwert der Durchläufe	11