Master-Seminar: Hochleistungsrechner - Aktuelle Trends und Entwicklungen
Aktuelle GPU-Generationen (NVidia Volta, AMD Vega)

Stephan Breimair
Technische Universität München

23.01.2017

Abstract

GPGPU - General Purpose Computation on Graphics Processing Unit, ist eine Entwicklung von Graphical Processing Units (GPUs) und stellt den aktuellen Trend bei NVidia und AMD GPUs dar.

Deshalb wird in dieser Arbeit gezeigt, dass sich GPUs im Laufe der Zeit sehr stark differenziert haben. Während auf technischer Seite die Anzahl der Transistoren stark zugenommen hat, werden auf der Software-Seite mit neueren GPU-Generationen immer neuere und umfangreichere Programmierschnittstellen unterstützt. Damit wandelten sich einfache Grafikbeschleuniger zu multifunktionalen GPGPUs. Die neuen Architekturen NVidia Volta und AMD Vega folgen diesem Trend und nutzen beide aktuelle Technologien, wie schnellen Speicher, und bieten dadurch beide erhöhte Anwendungsleistung. Bei der Programmierung für heutige GPUs wird in solche für herkömmliche Grafikanwendungen und allgemeine Anwendungen differenziert. Die Performance-Analyse untergliedert sich in High- und Low-Level-Analyse. Entscheidend für das GPU-Profiling ist, ob Berechnungskosten Speicherkosten verdecken beziehungsweise umgekehrt oder mangelnde Parallelisierung vorliegt.

Insgesamt konnte mit dieser Arbeit dargelegt werden, dass die aktuellen GPU-Architekturen eine weitere Fortführung der Spezialisierung von GPUs und einen anhaltenden Trend hin zu GPGPUs darstellen. Allgemeine und wissenschaftliche Anwendungen sind heute also ein wesentliches Merkmal von GPUs geworden.

1 Einleitung

Grafikbeschleuniger existieren bereits seit Mitte der 1980er Jahre, wobei der Begriff „GPU“, im Sinne der hier beschriebenen Graphical Processing Unit (GPU) [1], 1999 von NVidia mit deren Geforce-256-Serie eingeführt wurde.

Im strengen Sinne sind damit Prozessoren gemeint, die die Berechnung von Grafiken übernehmen und diese in der Regel an ein optisches Ausgabegerät übergeben. Der Aufgabenbereich hat sich seit der Einführung von GPUs aber deutlich erweitert, denn spätestens seit 2008 mit dem Erscheinen von NVidias „GeForce 8“-Serie ist die Programmierung solcher GPUs bei NVidia über CUDA (Compute Unified Device Architecture) möglich.

Da die Bedeutung von GPUs in den verschiedensten Anwendungsgebieten, wie zum Beispiel im Automobilsektor, zunehmend an Bedeutung gewinnen, untersucht diese Arbeit aktuelle GPU-Generationen, gibt aber auch einen Rückblick, der diese aktuelle Generation mit vorhergehenden vergleicht. Im Fokus stehen dabei die aktuellen Architekturen, deren technische Details beziehungsweise die Programmierung sowie Performance-Analyse von und für GPUs.

Im Abschnitt 2 wird dementsprechend diskutiert, wie sich GPUs seit ihrer Einführung in den Markt der elektronischen Rechenwerke, entwickelt und spezialisiert haben. Abschnitt 3 gibt einen tieferen Überblick über die Architekturen aktueller GPUs von NVidia und AMD, technische Details und die Leistungsfähigkeit. Anschließend werden in Abschnitt 4 Möglichkeiten der

Programmierung für GPUs vorgestellt. Da es jedoch auch im GPU-Bereich nötig beziehungsweise sinnvoll ist, eine Performance-Analyse durchzuführen, soll dies im Abschnitt 5 genauer auf verschiedenen Leveln diskutiert werden.
Insgesamt liegt der Fokus dieser Arbeit aber auf aktuellen Architekturen für technische und wissenschaftliche Berechnungen. Herkömmliche Grafikprogrammierung, wie zum Beispiel mit C++ und DirectX oder OpenGL, sollen deshalb nur am Rande Erwähnung finden.

2 Entwicklung der GPUs

Dieser Abschnitt gibt einen Überblick über die Entwicklung der GPUs seit ihrer Einführung.

2.1 Rückblick [13]

Die ersten Jahre der GPUs sind in Tabelle 1 abgebildet. Bei AMD folgten danach bis heute (Stand 12.2017) noch die HD Radeon-R200-Serie, die Radeon-R300-Serie, die Radeon-400 und Radeon-500-Serie. Bei diesen ist die erstmalige Verwendung von HBM-Speicher mit dem Fiji-Grafikprozessor in der Radeon R9 Fury X (gehört zur Radeon-R300-Serie) hervorzuheben. Außerdem nutzten die Polaris-Grafikchips (Radeon-400-Serie) bereits die GCN 4.0 Architektur, wodruch die Effizienz gesteigert werden konnte. Allerdings war beziehunsweise ist diese Generation und auch die nachfolgende Rebrand-Generation (Radeon-500-Serie) nicht in der Lage die Leistung des Konkurrenten NVidia zu erreichen. Die neueste Grafikkarten-Serie (Stand 12.2017) seitens AMD ist die Radeon-Vega-Serie, die die GCN-Architektur bereits in der 5. Ausbaustufe verwendet und DX 12.1, OpenGL 4.5, Open CL 2.0+ sowie Vulkan 1.0 unterstützt sowie 12,5 Milliarden Transistoren besitzt. Für weitere Details zum Vega-Grafikchip siehe Unterunterabschnitt 3.1.2.
Neben AMD und NVidia existierten beziehungsweise existieren noch andere Grafikchip-Hersteller mit teilweise eigenen technischen Neuerungen, die aber hier der Übersichtlichkeit halber keine Erwähnung finden sollen.

2.2 Differenzierung der GPUs

Der größte Ausbau der Maxwell-Architektur (GM200) von NVidia verfügte gegenüber dem Vorgänger (Kepler) aber nicht mehr über erweiterte Double-Precision-Fähigkeiten (FP64), wozu bis zu Kepler separate ALUs verbaut waren. Dies geschah vermutlich aus Platzgründen. [2] Während Double-Precision-Operationen für 3D-Anwendungen wie Spiele nicht benötigt wurden, machte das Fehlen von Double-Precision-Fähigkeiten den Einsatz in den professionelleren Tesla-Karten unmöglich. Deshalb wurde dort mit GK210 eine verbesserte Variante der älteren Kepler-Architektur eingesetzt. Dies kann als Beginn einer echten Differenzierung der GPUs betrachtet werden. Auch die zweite Generation der Maxwell-Architektur (GM204) wies eine zu geringe Double-Precision-Leistung auf, um in den Tesla-Karten zum Einsatz zu kommen, wo also weiterhin die Kepler-Architektur verwendet wurde. Die GeForce-10-Serie für den Desktop- und Workstation-Bereich nutzt die Pascal-Architektur im 16 nm Fertigungsprozess. Die Ausbaustufen reichen von GP108 bis GP100, wobei sich der größte GP100-Chip stark von den restlichen unterscheidet. Durch den kleineren Fertigungsprozess entfielen die Limitierungen wegen zu großer Die-Fläche, wie die zu geringe Double-Precision-Leistung, allerdings entschied sich NVidia wieder, für die professionellen Tesla-Karten einen eigenen, nämlich den sich stark unterscheidenden GP100-Chip, einzusetzen. Dort wurden ein anderer Clusteraufbau, High Performace Computing Fähigkeiten und ein ECC-HBM2-Speicherinterface verwendet. Die noch aktuellere Volta-Architektur ist derzeit (Stand Dezember 2017) nur für die Tesla-Karten verfügbar und erhöht durch ein Schrumpfen des Fertigungsprozesses von 16 nm auf 12 nm und eine Vergrößerung der Die-Fläche die Transistorenzahl auf 21 Milliarden (gegenüber 15 Milliarden bei Pascal). Weitere Details zur aktuellen Volta-Architektur finden sich in Unterunterabschnitt 3.1.1.
Die Separierung in Desktop- bzw. Workstation-Karten und Tesla-Karten umfasst jedoch nicht die ganze vorhandene Differenzierung von GPUs. Mit NVidia Quadro existieren GPU-Karten, die den Desktop- bzw. Workstation-Pendants sehr ähneln und sogar die gleichen Treiber verwenden, allerdings durch eine andere Beschaltung eine andere Chip-ID erhalten, was wiederum den Treiber veranlasst spezielle Funktionen für professionelle Anwendungen im CAD-, Simulations- und Animationsbereich freizuschalten. Von ATI beziehungsweise AMD existieren entsprechende Karten unter den Bezeichnungen FireGL- und FirePro bezie-

Jahr	Produkt / Serie	Transistoren	CUDA-Kerne	Besondere Technologie(n) / Besonderheiten
1990er			-	Entwicklung dedizierter GPUs beginnt
1997	NV Riva 128	3 Millionen	-	Erster 3D-Grafikbeschleuniger
1995-98	ATI Rage	8 Millionen	-	Twin Cache Architektur
2000	NV GeForce 256	17 Millionen	-	Erste, echte GPU; DX7, OpenGL
2000	ATI Radeon-7000	30 Millionen	-	Effizienzsteigerungen (HyperZ)
2000	NV GeForce 2	25 Millionen	-	Erste GPU für Heimgebrauch
2001	NV GeForce 3	57 Millionen	-	Erste programmierbare GPU; DX8, OpenGL; Pixel- u. Vertex-Shader
2002	ATI Radeon-9000	>100 Millionen	-	Einstieg ATIs in High-End-Markt
2003	NV GeForce FX	135 Millionen	-	32-bit floating point (FP) programmierbar
2004	NV GeForce 6	222 Millionen	-	GPGPU-Cg-Programme; DX9; OpenGL; SLI; SM 3.0
2006	NV GeForce 8	681 Millionen	128	Erste Grafik- und Berechnungs GPU; CUDA; DX10; Unified-Shader-Architektur
2007	NV Tesla			GPUs für wissenschaftliche Anwendungen
2008	NV GeForce-200		240	CUDA; OpenCL; DirectCompute
2008	ATI-Radeon-HD-4000		-	GDDR5; Übernahme ATIs durch AMD
2009	NV Fermi / GeForce-400	3 Milliarden	512	Zunehmende GPGPU-Bedeutung; C++ Unterstützung; DX11; ECC-Speicher; 64-bit Addressierung
2010	ATI-Radeon-HD-5000	∼2 Milliarden	-	Terrascale-Architektur; Optimierungen für GPU-Computing
2012	NV Kepler / GeForce-600	∼3,5 Milliarden	1536	GPU-Boost; Graphics Processing Cluster (GPCs)
2012	AMD-Radeon-HD-7000	4,3 Milliarden	-	Neue GCN-Architektur; Vulkan-API

Tabelle 1: Entwicklung der GPUs

hungsweise seit der 4. Generation der GCN Architektur unter dem Namen Radeon Pro.

2.3 Spezialisierung der GPUs

Hinsichtlich Spezialisierung von GPUs, sind die bereits aus den vorhergehenden Abschnitten bekannten Differenzierungen, wie Tesla- und Quadro-Karten, zu nennen.

Allerdings geht die Differenzierung von GPUs inzwischen soweit, dass von Spezialisierung auf einzelne Anwendungsgebiete gesprochen werden muss. Für autonomes Fahren bietet NVidia beispielsweise Drive PX an, ein Fahrzeugcomputer mit künstlicher Intelligenz, der bezüglich Umfang von Einzelprozessoren für einzelne Fahrzeugfunktionen bis zu Kombinationenen mehrerer Grafikprozessoren für autonome Robotertaxis reicht. Laut NVidia [3] kombiniert die Plattform Deep Learning, Sensorfusion und Rundumsicht, was sogar Level-5, also vollständige, Fahrzeugautonomie bieten soll.

Darüber hinaus bietet NVidia mit NVidia DGX-1 eine Plattform für die KI-Forschung und den KI-Einsatz in Unternehmen. Letztendlich kommt aber hierbei ebenfalls eine aktuelle Tesla V100 und spezialisierte Software zum Einsatz. [4]

Zuletzt existiert unter dem Namen NVidia Tegra auch ein komplettes System-on-a-Chip (SoC) für mobile Geräte, aber auch Autos, das neben der Grafikeinheit auch ARM-Kerne, Chipsatz, teilweise ein Modem und weiteres enthält. Dieses SoC trägt bei NVidia den Namen Tegra, das in der aktuellen Generation (Tegra X1) allerdings nur eine Mawell-Grafikeinheit enthält. Hervorzuheben ist jedoch, dass trotz des mobilen Anwen-

dungsfokus CUDA unterstützt wird.

3 Architekturen aktueller GPUs

Nachdem im vorhergehenden Abschnitt die Entwicklung der GPUs dargelegt wurde, sollen im folgenden die technischen Details der aktuellen Architekturen von NVidia und AMD, sowie deren Leistungsfähigkeit, Kosten und Anwendungsgebiete dargestellt werden.

3.1 Technische Details

Bei den aktuellen Architekturen soll bei NVidia der Fokus auf der Volta-Architektur in Form der Tesla V100 GPU, die auf eben dieser Architektur basiert, und bei AMD auf Vega-Chips liegen, der 5. Ausbaustufe der GCN-Architektur.

3.1.1 NVidia Tesla V100 GPU [14]

Die Volta-Architektur wird gegenwärtig nur in Form der Tesla V100 GPU (GV100) umgesetzt, die unter anderem die folgenden Neuerungen und Verbesserungen mitbringt:

- Neue Streaming Multiprozessor (SM) Architektur für Deep Learning
- NVLink 2.0
- HBM2 Speicher mit höherer Effizienz
- Volta Multi-Prozess Dienst (MPS)
- Verbesserte Unified Memory and Address Translation Dienste
- Modi für maximale Performance und Effizienz
- Cooperative Groups and Launch APIs
- Für Volta optimierte Software
- Weitere, wie zum Beispiel unabhängiges Thread-Scheduling

Daneben enthält die Tesla-GPU 21,1 Milliarden Transistoren auf einer Die-Fläche von 815 mm^2 und wird im TSMC 12 nm FFN (FinFET NVidia) Fertigungsprozess hergestellt.

Durch die neue Streaming Multiprozessor (SM) Architektur ist diese 50% energieeffizienter als diejenige in der Pascal-Architektur, bei gleichzeitig deutlich erhöhter FP32- und FP64-Leistung. Mithilfe neuer Tensor-Kerne steigt die Spitzen-TFLOPS-Leistung auf das Zwölffache für das Training und das Sechsfache bei der Inferenz.

NVLink 2.0 ermöglicht eine höhere Bandbreite, mehr Links und eine bessere Skalierbarkeit für Multi-GPU- und CPU-Systeme. Deshalb unterstützt Volta GV100 bis zu sechs NVLinks mit einer Gesamtbandbreite von 300 GB/s.

Das Subsystem für die HBM2-Speicherverwaltung erreicht laut NVidia eine Spitzenleistung von 900 GB/s Speicherbandbreite. Durch die Kombination des neuen Volta-Speichercontrollers und einer neuen Generation Samsung HBM2-Speichers, erreicht Volta insgesamt eine 1,5-fache Speicherbandbreite gegenüber Pascal.

Der Volta Multi-Prozess Dienst (MPS) hardwarebeschleunigt kritische Komponenten des CUDA-MPS-Servers, wodurch eine bessere Leistung, Entkopplung und besseres QoS erreicht wird, wenn mehrere Anwendungen für wissenschaftliche Berechnungen die GPU teilen müssen. Dadurch steigt die erlaubte Anzahl an MPS-Clients von 16 bei Pascal auf nun 48.

Verbesserte Unified Memory- und Address Translation-Dienste erlauben eine genauere Migration der Speicherseiten zu demjenigen Prozessor, der diese am häufigsten nutzt. Dadurch wird erneut die Effizienz beim Zugriff auf Speicherbereiche, die von mehreren Prozessoren verwendet werden, erhöht. Auf der sogenannten IBM-Power-Plattform gestattet es eine entsprechende GPU-seitige Unterstützung derselben auf die Seitentabellen der CPU direkt zuzugreifen.

Die Thermal Design Power (TDP) von Volta ist mit 300 Watt spezifiziert, die jedoch nur im Maximum-Performance-Mode genutzt wird, wenn Anwendungen höchste Leistung und höchsten Datendurchsatz benötigen. Dagegen eignet sich der Maximum-Efficiency-Mode für Datencenter, die die Leistungsaufnahme regulieren und die höchste Leistung pro Watt erzielen wollen.

Mit NVidia CUDA 9 wurden Cooperative Groups (kooperative Gruppen) für miteinander kommunizierende Threads eingeführt. Mithilfe der Cooperative Groups können Entwickler die Granularität bestimmen mit welcher Threads kommunizieren, worduch effizientere, parallele Dekompositionen von Anwendungen erstellt werden können. Alle GPUs seit Kepler unterstützen diese Gruppen, Pascal und Volta aber auch die Launch-APIs, die die Synchronisation zwischen CUDA-Thread-Blöcken ermöglichen.

Der Aufbau einer Volta-GPU besteht aus sechs GPU Processing Clusters (GPCs), die jeweils über sieben Texture Processing Clusters (TPCs) und 14 Strea-

ming Multiprozessoren (SMs) verfügen. Jeder Streaming Multiprozessor hat dabei:

- 64 FP32 Kerne
- 64 INT32 Kerne
- 32 FP64 Kerne
- 8 Tensor Kerne
- Vier Textureinheiten

Insgesamt verfügt jede volle GV100 GPU also über 5376 FP32- und INT32-, 2688 FP64-, 672 Tensor-Kerne und 336 Textureinheiten. Zusätzlich besitzt eine vollständige GV100-GPU acht 512-bit Speichercontroller mit insgesamt 6144 KB L2 Cache.
Die Anzahl der Streaming Multiprozessoren hat sich gegenüber der GP100, die auf der Pascal-Architektur aufbaut, nicht nur von 56 auf 80 erhöht, sondern diese enthalten nun auch erstmalig 8 Tensor-Kerne, die explizit für Deep Learning vorgesehen sind.
In Abbildung 1 ist ein einzelner Streaming Multiprozessor einer Volta GV100 dargestellt.
Wie leicht zu erkennen ist, nehmen die Tensor-Kerne eine nennenswerte Fläche innerhalb des Streaming Multiprozessors ein, was insofern bemerkenswert ist, als dass es sich um die erste GPU-Generation überhaupt handelt, die diese Tensor-Kerne beinhaltet. Von diesen Tensor-Kernen ist jeder in der Lage 64 floating point fused multiply-add (FMA) Operationen pro Takt durchzuführen. Diese Berechnungsmethode unterscheidet sich von der herkömmlichen Berechnungsmethode für Addition und Multiplikation dadurch, dass die Berechnung mit voller Auflösung durchgeführt wird und eine Rundung erst am Ende erfolgt. [16] Wie bereits erwähnt ist Tesla V100 mithilfe der Tensorkerne in der Lage 125 Tensor-TFLOPS für Training- und Inferenzanwendungen zu liefern.
Jeder Tensor-Kern rechnet auf einer 4×4-Matrix und führt die Operation $D = A \times B + C$ auf dieser aus, wobei A, B, C und D 4×4 Matrizen sind. In der Praxis werden mit den Tensor-Kernen Operationen auf größeren, auch mehrdimensionalen, Matrizen durchgeführt, die hierzu in diese 4×4 Matrizen zerlegt werden. Volta Streaming Multiprozessoren besitzen zudem ein verbessertes L1 Datencache- und Shared Memory-System, das die Effizienz steigern und die Programmierbarkeit vereinfachen soll. Durch die Kombination in einem Block verringert sich die Latenz bei

Abbildung 1: Volta GV100 Streaming Multiprocessor (SM) [14]

gleichzeitig besserer Bandbreite. Shared Memory bietet hohe Bandbreite, geringe Latenz und konstante Performance, muss jedoch bei der CUDA-Programmierung explizit verwaltet werden. Bei der Kombination von Cache und Shared-Memory wird fast die gleiche Leistung erreicht (ca. 7% weniger), jedoch ohne die Notwendigkeit dies in Anwendungen manuell zu steuern.
Neben den bereits teilweise zuvor erwähnten neuen Funktionalitäten, ist die Volta-Architektur im Gegensatz zu Pascal zuletzt in der Lage FP32- und INT32-Operationen gleichzeitig auszuführen, da separate FP32- und INT32-Kerne vorhanden sind, wodurch sich zum Beispiel bei FMA Geschwindigkeitsvorteile in bestimmten Anwendungsszenarien ergeben können.

3.1.2 AMD [10]

Auch wenn von AMD selbst als Vega-Architektur bezeichnet, so handelt es sich bei Vega streng genommen um die 5. Ausbaustufe der GCN-Architektur. Im Fol-

genden werden diese Begrifflichkeiten jedoch der Einfachheit halber synonym verwendet.
Für diese neue Architektur beziehungsweise Ausbaustufe sind verschiedene Implementierungen geplant, wobei die aktuelle als Vega 10 bezeichnet wird. Abbildung 3 zeigt die im 14 nm LPP FinFet Fertigungsprozess hergestellte Architektur Vega 10 mit ihren 12,5 Milliarden Transistoren. Durch das kleinere Fertigungsverfahren beträgt der Boost-Takt nun 1,67 Ghz gegenüber 1 Ghz bei Produkten mit Architekturen, die im 28 nm Verfahren hergestellt wurden. Bei Vega werden als Zielgruppen Gaming, Workstation, HPC und Machine Learning genannt.
Mit 64 Rechneneinheiten der nächsten Generation (Next-generation compute units (NCUs)) besitzt Vega 10 4096 Stream-Prozessoren, deren Anzahl bisherigen Radeon GPUs ähnelt, die aber aufgrund des Takts und der Architekturverbesserungen Instruktionsanweisungen schneller ausführen. Dadurch sind diese in der Lage 13,7 Teraflops von Berechnungen einfacher Genauigkeit und 27,4 Teraflops mit halber Genauigkeit durchzuführen. Die neue Architektur ist in Abbildung 3 dargestellt.
Des weiteren bietet Vega sogenanntes Infinity Fabric mit geringen Latenzen, das Logikeinheiten innerhalb des Chips verbindet und gleichzeitig Quality of Service und Sicherheitsapplikationen zur Verfügung stellt.
Ebenfalls neu in der Vega-Architektur ist eine neue Speicherhierarchie mit einem Cache Controller für hohe Bandbreite (HBCC). Üblicherweise werden Daten von den Registern der Rechenelemente aus dem L1-Cache bezogen, der seinerseits den L2-Cache nutzt. Dieser L2-Cache wiederum stellt einen Zugriff mit hoher Bandbreite und geringer Latenz auf den Videospeicher der GPU bereit. Dabei müssen das gesamte Datenset und alle Ressourcen im Videospeicher gehalten werden, da es ansonsten zu Einschränkungen bezüglich der Bandbreite und Latenz kommt. Bei Vega verhält sich der Videospeicher nun seinerseits wie eine Art letzter Cache, wozu dieser Speicherseiten, die noch nicht im GPU-Speicher vorliegen, selbst über den PCIe Bus bezieht und im Cache mit hoher Bandbreite speichert. Bisher wurde stattdessen die komplette Berechnung der GPU verzögert, bis der Kopiervorgang abgeschlossen war. Diese Neuheit der Speicherhierarchie bei der Vega-Architektur ist möglich, da die Speicherseiten meist

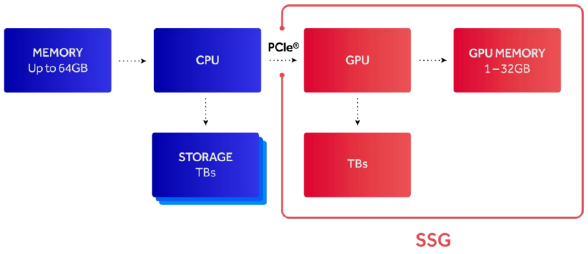

Abbildung 2: Systemspeicherarchitektur mit SSG [9]

deutlich kleiner als die kompletten Ressourcen sind und folglich schnell vom neuen Controller für hohe Bandbreite (HBCC), einem neuen Bestandteil der Speichercontrollerlogik, kopiert werden können.

[9] Davon profitiert insbesondere die neue Radeon Pro SSG, eine GPU mit Solid State Disk (SSD), die Latenz und CPU-Overhead deutlich reduzieren soll. Diese spezielle GPU verhält sich, als besäße sie ein Terabyte lokalen Videospeichers. Bei der bisher üblichen Systemarchitektur für Massenspeicher in einem CPU-GPU-System erfolgt die Datenübertragung zwischen zwei Endpunkten gewöhnlich in den folgenden drei Schritten: Zuerst werden die Daten von einem PCIe-Endpunkt in den System-Cache gelesen und anschließend von dort in den Prozesscache kopiert, von wo sie zuletzt zum anderen PCIe-Endpunkt kopiert werden.

Abbildung 2 zeigt wie die Systemspeicherarchitektur mit der Radeon Pro SSG aufgebaut ist. Dort werden sogenannte Ende-zu-Ende Leseoperationen ermöglicht, indem zunächst, über PCIe exponierter, Videospeicher reserviert wird, anschließend die SSD angewiesen wird eine Leseoperation durchzuführen und zuletzt die Daten auf der GPU verarbeitet werden. Dies hat auch den Vorteil, dass die CPU, sobald der Pfad zwischen GPU und NVMe-Speicher hergestellt ist, zum Datentransfer nicht mehr benötigt wird und folglich für andere Aufgaben verwendet werden kann, was die Systemperformance weiter erhöht, da Datensätze nur so schnell von der GPU verarbeitet werden können, wie diese dieser zur Verfügung stehen. Die Größe solcher Datensätze nimmt bei modernen Anwendungen kontinuierlich zu, was jedoch durch immer schnellere GPU-Architekturen kompensiert wird. Allerdings wird dann häufig die Datenübertragung zur GPU zum Flaschenhals, der mit der CPU-GPU-Systemarchitektur der Ra-

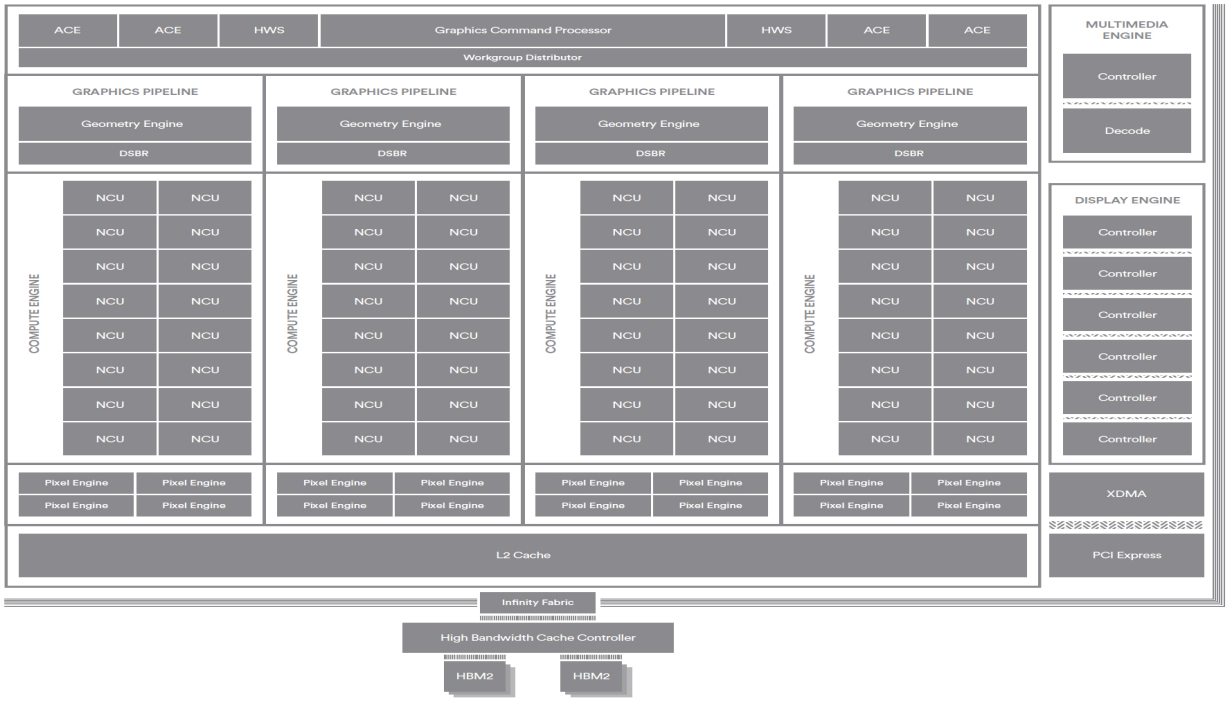

Abbildung 3: AMD Vega 10 Architektur [10]

deon Pro SSG entfällt.

Zur Vega 10-Archiektur zurückkommend gilt es weiterhin hervorzuheben, dass alle Grafikblöcke nun Clients des gemeinsamen L2-Cache sind, womit Datenredundanz verringert wird. Aus diesem Grund wurde der L2-Cache mit 4 MB gegenüber der Vorgängergeneration verdoppelt.

Außerdem wird eine Geometrie-Engine der nächsten Generation (NGG) in Vega integriert, die einen viel höheren Polygon-Durchsatz pro Transistor erreicht. Hierzu werden „primitive“ Shader unterstützt, womit Teile der Geometrie-Pipeline mit einem neuen Shader-Typ kombiniert und ersetzt werden. In üblichen Szenarien wird rund die Hälfte sogenannter Primitiven der Geometrie verworfen, was bisher durch die Geometrie-Pipelines erst nach der Vertex-Verarbeitung geschehen konnte, nun aber früher erfolgen kann. Dadurch erreichen die vier Geometrie-Engines von Vega 10 nun 17 statt nur 4 Primitive pro Takt.

Ebenfalls mit Vega wird sogenanntes „Rapid Packed Math“ für spezialisierte Anwendungen wie Machine Learning und Videoverarbeitung eingeführt, wobei 16-bit Datentypen, wie float und Integer, zum Tragen kommen. Dies ist ebenso Bestandteil der Vega NCU wie die Unterstützung neuer Instruktionen, zum Beispiel die Summe der absoluten Differenzen (SAD), die häufig in Video- und Bildverarbeitung benötigt wird.

Nicht zuletzt wurde auch die Pixel-Engine der Vega-Architektur überarbeitet, unter anderem in Form des Draw-Stream Binning Rasterizers (DSBR), der unnötige Berechnungen und Datentransfer vermeiden soll. Beim Rendern des Bildes wird dieses hierbei zunächst in ein Gitter von Kacheln aufgeteilt, wobei Stapel von Primitiven gebildet werden. Der DSBR traversiert anschließend diese Stapel nacheinander und überprüft, welche vollständige oder Teile von Primitive enthalten. Dadurch und durch weitere Maßnahmen soll die Leistungsaufnahme verringert und höhere Bildwiederholungsraten erreicht werden.

Abschließend werden mit der Vega Architektur DirectX in Feature Level 12.1, verbesserte Stromsparfunktionen und Display- und Multimediaverbesserungen wie HDR

integriert beziehungsweise unterstützt.

3.2 Leistungsfähigkeit

Dieses Kapitel beschreibt die Leistungsfähigkeit aktueller GPU-Architekturen und deren Umsetzungen. Ein direkter Vergleich ist jedoch schwierig beziehungsweise selten möglich, da die Architekturen zum einen sehr neu sind (Stand 12.2017) und zum anderen deren Umsetzungen unterschiedliche Andwenundgsfokusse haben.

3.2.1 NVidia Tesla V100 GPU [14]

Der Vergleich zwischen Pascal und Volta zeigt, dass die Matrixmultiplikation von 4×4 Matrizen und die dazu nötigen 64 Operationen von Volta zwölfmal so schnell berechnet werden. In cuBlAS, einer Bibliothek für Berechnungen der linearen Algebra, die auf CUDA aufbaut, weist Volta gegenüber Pascal bei einfacher Genauigkeit eine bis zu 1,8-fach und bei gemischter Genauigkeit (FP16 Input, FP32 Output) bis zu 9,3-fach höhere Geschwindigkeit auf.

3.2.2 AMD [10]

Die Radeon Pro SSG besitzt in einem CPU-GPU-System gegenüber einem herkömmlichen CPU-GPU-System eine rund fünf Mal höhere Performance bei der Videoverarbeitung in 8K. Weiterhin ist Vega, mithilfe der neuen Geometrie-Engine der nächsten Generation (NGG) und Fast Path, in der Lage über 25 Gigaprimitive, gegenüber ca. 5 bei Fiji, zu verwerfen.

Obwohl die GPUs von NVidia und AMD aufgrund der Spezialisierung nicht vollständig vergleichbar sind, existieren Bechmarks, wie der PassMark Benchmark [5], die die verschiedenen Karten dennoch vergleichen. In besagtem PassMark Benchmark (Stand 14.12.2017) führt die Titan V100 das Testfeld mit 16,842 Punkten mit einigem Vorsprung an. Eine GeForce GTX 1080 Ti, die zweithöchste Ausbaustufe der Pascal-Architektur, erreicht dagegen nur 13,634 Punkte, ist damit aber immer noch deutlich schneller als die schnellste Radeon Vega Karte im Test mit 11,876 Punkten. Damit wäre die aktuelle Umsetzung der Tesla-Architektur circa 50% schneller als diejenige der Vega-Architektur.

4 Programmierung für GPUs

Dieser Abschnitt beschreibt übliche Vorgehensweisen bei der GPU-Programmierung und dazu nötige Software. Die programmierbaren Einheiten einer GPU verfolgen dabei ein single-program multiple-data (SPMD) Programmiermodell, was bedeutet, dass eine GPU aus Effizienzgründen parallel mehrfach das gleiche Programm auf verschiedenen Elementen ausführt. Deshalb sind Programme für GPUs auch immer derart strukturiert, dass es mehrere gleichzeitig zu verarbeitende Elemente und parallele Verarbeitung dieser Elemente von einem einzigen Programm gibt.
Die Programmierung für Grafikanwendungen unterscheidet sich dabei von der Programmierung für allgemeine Anwendungen. Erstere erfolgt dabei folgendermaßen:

1. Der Programmierer spezifiziert die Geometrie, die eine bestimmte Region des Bildschirms abdeckt. Der Rasterer generiert daraus an jeder Pixelstelle ein Fragment, das von der Geometrie abgedeckt ist.
2. Jedes Fragment wird von einem Fragment-Programm bearbeitet.
3. Das Fragment-Programm berechnet dabei den Wert des Fragments mithilfe einer Kombination aus mathematischen Operationen und der globale Speicher liest von einem globalen Texturspeicher.
4. Das resultierende Bild kann dann als Textur in weiteren Durchläufen innerhalb der Grafik-Pipeline weiter verwendet werden.

und für allgemeine Anwendungen wie folgend:

1. Der Programmierer spezifiziert direkt den Berechnungsbereich als strukturiertes Raster von Threads.
2. Ein SPMD-Progamm für allgemeine Anwendungen berechnet den Wert jedes Threads.
3. Der Wert wird dabei erneut durch eine Kombination mathematischer Operationen, sowie lesendem und schreibendem Zugriff auf den globalen Speicher berechnet. Der Buffer kann dabei sowohl für lesende und schreibende Operationen verwendet werden, um flexiblere Algorithmen zu ermöglichen (z.B. In-Place-Algorithmen zur Reduzierung der Speichernutzung).
4. Der resultierende Buffer im globalen Speicher kann wiederum als Eingabe für weitere Berechnungen genutzt werden.

Früher musste bei der Programmierung der Umweg über geometrische Primitive, den Rasterer und

Fragment-Programme, ähnlich wie bei der Grafikprogrammierung, gegangen werden.
Dies legt bereits nahe, dass als Softwareumgebung für die GPGPU-Programmierung früher direkt die Grafik-API genutzt wurde. Auch wenn dies vielfach erfolgreich genutzt wurde, unterscheiden sich, wie bereits oben erwähnt, die Ziele herkömmlicher Programmiermodelle und einer Grafik-API, denn es mussten Fixfunktionen, grafik-spezifische Einheiten, wie zum Beispiel Texturfilter und Blender, verwendet werden.
Mit DirectX 9 wurde die high-level shading language (HLSL) eingeführt, die es erlaubt die Shader in einer C-ähnlichen Sprache zu programmieren. NVidias Cg verfügte über ähnliche Fähigkeiten, war allerdings in der Lage für verschiedene Geräte zu kompilieren und stellte gleichzeitig die erste höhere Sprache für OpenGL bereit. Heute ist die OpenGL Shading Language (GLSL) die Standard-Shading-Sprache für OpenGL. Nichtsdestotrotz sind Cg, HLSL und GLSL immer noch Shading-Sprachen, wobei die Programmierung grafikähnlich erfolgen muss.
Inzwischen gibt es deshalb sowohl von NVidia, als auch von AMD, spezielle GPGPU Programmiersysteme. NVidia stellt hierzu das bekannte CUDA-Interface zur Verfügung, das wie Cg über eine C-ähnliche Syntax besitzt, über zwei Level von Parallelität (Daten-Parallelität und Multithreading) verfügt und nicht mehr grafikähnlich programmiert werden muss.
AMD bietet Hardware Abstraction Layer (HAL) und-Compute Abstraction Layer (CAL). Außerdem wird OpenCL unterstützt, das seinerseits von CUDA inspiriert wurde und auf einer großen Bandbreite von Geräte genutzt werden kann. Bei OpenCL erfolgt die Programmierung in OpenCL C, das zusätzliche Datentypen und Funktionen für die parallele Programmierung bereitstellt. Es handelt sich also wie bei der Programmierung von CUDA um eine C-ähnliche Syntax. [15], [13]

5 Performance-Analyse bei GPUs

Progammierwerkzeuge zur Laufzeitanalyse werden im allgemeinen Profiler genannt, die den Programmierer dabei unterstützen ineffiziente Programmteile zu identifizieren. Der Fokus liegt hierbei auf Funktionen, die am häufigsten durchlaufen werden, da diese den größten Einfluss auf die Gesamtlaufzeit eines Programms haben. Auch wenn moderne Profiler Nebenläufigkeit unterstützen, so stellt diese enorme Anforderungen an etwaige Analysetools. Schon bei einer Threadanzahl wie sie bei einer CPU vorkommt kann es zu Schwierigkeiten bei der Erkennung von Laufzeitproblemen, wie etwa durch Deadlocks, kommen.

5.1 High-Level Analyse

Als High-Level wird im Folgenden die Analyse der GPU-Performance mit Softwaretools bezeichnet, welche aber oft ihrerseits Low-Level-Analyse nutzen und diese in Form einer API oder Ähnlichem dem Programmierer zur Verfügung stellen. Die einzelnen Tools und ihre Funktionsweise ist genauer in Unterabschnitt 5.3 beschrieben.
Als Beispiel, wie High-Level-Analyse durchgeführt werden kann, soll die Arbeit von Sim, Dasgupta, Kim und Vuduc [17] dienen, die feststellen, dass es sich äußerst schwierig gestaltet, bei Plattformen mit vielen Kernen die Ursachen für Engpässe festzustellen. Deshalb stellen die genannten Autoren ein Framework namens GPUPerf Framework vor, das solche Engpässe aufdecken soll. Es kombiniert ein genaues analytisches Modell für moderne GPGPUs und gut interpretierbare Metriken, die direkt die möglichen Performance-Verbesserungen für verschiedene Klassen von Optimierungstechniken zeigen. Das zugrunde liegende analytische Modell, das als MWP-CWP-Modell (siehe Abbildung 4) bezeichnet wird, nutzt folgende Eingaben: Anzahl der Instruktionen, Speicherzugriffe, Zugriffsmuster sowie architekturelle Eigenschaften wie DRAM-Latzen und -Bandbreite.
Auf der einen Seite beschreibt der Memory Warp Parallelism (MWP) die Anzahl an Warps, Ausführungseinheiten einer GPGPU, pro Streaming-Multiprozessor (SM), die gleichzeitig auf den Speicher zugreifen können. Auf Ebene des Speichers spiegelt MWP also Parallelität wider. MWP ergibt sich dabei als Funktion aus Speicherbandbreite, Parameter von Speicheroperationen, wie Latzenz und der aktiven Anzahl von Warps in einem SM. Das Framework modelliert die Kosten von Speicheroperationen durch die Anzahl von Anfragen über MWP. Auf der anderen Seite entspricht Computation Warp Parallelism (CWP) der Anzahl von Warps, die ihre Berechnungsperiode innerhalb einer Speicherwarteperiode plus eins abschließen können.

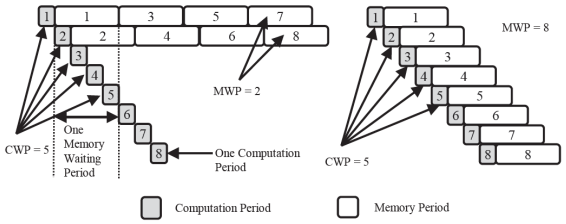

Abbildung 4: MWP-CWP-Modell, links: MWP $<$ CWP, rechts: MWP $>$ CWP [17]

Eine Berechnungsperiode ergibt sich dabei aus den durchschnittlichen Berechnungszyklen pro Speicherinstruktion. Das MWP-CWP-Modell dient dazu, zu identifizieren, welchen Kosten durch Multi-Threading verborgen werden können. Dazu wird in drei Fälle unterschieden:

1. MWP $<$ CWP: Die Kosten einer Berechnung werden durch Speicheroperationen verdeckt. Die gesamten Ausführungskosten ergeben sich durch die Speicheroperationen. (vgl. Abbildung 4, links)
2. MWP $\geq$ CWP: Die Kosten der Speicheroperationen werden durch die Berechnung verdeckt. Die gesamten Ausführungskosten ergeben sich aus der Summe der Berechnungskosten und einer Speicherperiode. (vgl. Abbildung 4, rechts)
3. Nicht genug Warps: Aufgrund mangelnder Parallelisierung werden sowohl Berechnungs- als auch Speicherkosten verdeckt.

Das Framework enthält nun einen sogenannten Performance-Advisor, der Informationen über Performance-Engpässe und möglichen Performance-Gewinn durch Beseitigung dieser Engpässe bereitstellt. Dazu nutzt dieser Advisor eben obiges MWP-CWP-Modell und stellt vier Metriken bereit, die sich entsprechend Abbildung 5 visualisieren lassen. Die vier enthaltenen Metriken sind:

- B_{itilp}: zeigt den potentiellen Performance-Gewin durch Erhöhung der Inter-Thread Parallelität auf Instruktionslevel.
- B_{memlp}: zeigt den potentiellen Performance-Gewin durch Erhöhung der Parallelität auf Speicherlevel.
- B_{fp}: spiegelt den potentiellen Performance-Gewinn wider, wenn die Kosten für ineffiziente

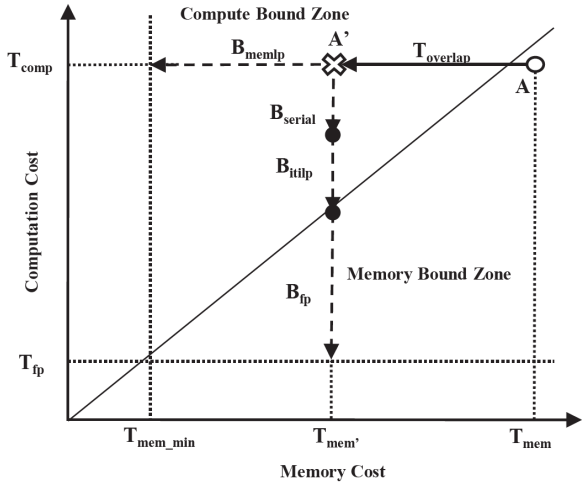

Abbildung 5: Potentieller Performance-Gewinn [17]

Berechnungen ideal entfernt werden. In der Praxis können hier keine 100% erreicht werden, da der Kernel immer auch Operationen wie Datentransfer haben muss.
- B_{serial}: zeigt die Kosteneinsparung, wenn Overhead durch Serialisierungseffekte wie Synchronisation und Ressourcenkonflikte entfernt werden.

Neben allgemeiner High-Level-Analyse gibt es auch Performance-Modellierung und -Optimierung, die sich auf spezielle Operationen bezieht. Guo, Wang und Chen [11] stellen zum Beispiel ein Analysetool vor, das Sparse-Matrix-Vektor-Multipliktion auf GPUs optimieren kann. Dazu stellt es, unter Verwendung einer integrierten analytischen und profilbasierten Performance-Modellierung, ebenfalls verschiedene Performance-Modelle bereit, um die Berechnungsdauer präzise vorherzusagen. Außerdem ist in dem genannten Tool ein automatischer Selektionsalgorithmus enthalten, der optimale Sparse-Matrix-Vektor-Multiplikation-Werte hinsichtlich Speicherstrategie, Speicherformaten und Ausführungszeit für eine Zielmatrix liefert.
Weitere Frameworks sind in Unterabschnitt 5.3 beschrieben.

5.2 Low-Level Analyse

Auf GPU-Basis ist die Low-Level Analyse mithilfe sogenannter Performance-Counter und anderen Methoden noch relativ neu. Verschiedene Hersteller wie NVidia

(The PAPI CUDA Component) und IBM (Parallel Environment Developer Edition) stellen Technologien bereit, die es ermöglichen auf Hardware-Counter innerhalb der GPU zuzugreifen. Prinzipiell ist der Übergang und die Definition von High- und Low-Level Analyse aber natürlich fließend.
[6] Der Einsatz der GPU-Counter soll anhand der Performance API (PAPI) gezeigt werden, die daneben aber auch noch sehr viele weitere Performanceanalyse-Funktionen bereitstellt. PAPI bietet hierzu ein Counter-Interface, das den Zugriff, das Starten, Stoppen und Auslesen von Countern für eine spezifizierte Liste von Events ermöglicht. Die Steuerung beziehungsweise Programmierung von PAPI erfolgt mittels C oder Fortran. Außerdem kann PAPI sogenannte benutzerdefinierte Gruppen von Hardware-Events namens Event-Sets verwalten, was für Programmierer einem feingranulares Mess- und Kontrollwerkzeug gleichkommen soll.
Auch die Umsetzung beziehungsweise Implementierung des zuvor genannten GPUPerf Frameworks kann als Low-Level Analyse bezeichnet werden.
Zuletzt gibt es auch Empfehlungen und Vorgehensweisen, beispielsweise von NVidia selbst [12], für Performance-orientierte Implementierungen. Allgemeine Empfehlungen sind in diesem Kontext zum Beispiel, mit 128-256 Threads pro Threadblock zu sarten, nur Vielfache der Warpgröße (32 Threads) zu nutzen und als Rastergröße 1000 oder mehr Threadblöcke einzusetzen.

5.3 Hilfsmittel (Tools)

Zur Analyse der Laufzeit gibt es auch bei GPUs eine Reihe von Profiling-Tools. Neben den bereits, zur Veranschaulichung von High- und Low-Level Analyse, vorgestellten Frameworks, ist eines dieser Tools NVidia PerfKit. [18]
Durch die enthaltene PerfAPI werden in PerfKit Performance-Counter zugänglich, die aufzeigen, wie ein Programm die GPU nutzt, welche Performance-Probleme vorliegen und ob nach einer Veränderung diese Probleme gelöst wurden. PerfKit kann dedizierte Experimente auf individuellen GPU-Einheiten ausführen, worduch Performance-Charakteristiken gewonnen werden, nämlich der Speed of Light (SOL)- und Bottleneck-Wert. Ersterer gibt im Allgemeinen an, wie stark der entsprechende GPU-Teil genutzt wird; er spiegelt die aktive Zeit der entsprechenden GPU-Einheit wider. Letzterer zeigt welchen zeitlichen Anteil eine GPU-Einheit einen Engpass darstellt. Es kann eine große Anzahl an verschiedenen GPU-Counter untersucht werden.
Der AMD Radeon GPU Profiler stellt Low-Level Timing-Daten für Barriers, Warteschlangensignale, Wellenfront-Belegung, Event-Timings und weitere bereit. [7]
Ebenfalls für AMD Chips steht die AMD CodeXL Entwicklungssuite bereit, die neben einem CPU-Profiler, einem GPU debugger, einem statischen Shader/Kernel Analysator auch einen GPU Profiler beinhaltet. Dieser sammelt Daten von den Performance-Countern, vom Anwendungsablauf, Kernelbelegung und bietet eine Hotspot-Analyse. Der Profiler sammelt Daten der OpenCL Runtime und von der GPU selbst und dient ebenfalls zur Reduzierung von Engpäassen und der Optimierung der Kernel-Ausführung. [8]
Die Autoren Zhang und Owens [19] schlagen ein quantitatives Performance-Analyse-Modell basierend auf einem Microbenchmark-Ansatz vor, das den nativen Instruktionssatz einer GPU nutzt.
Daneben existiert natürlich noch eine Vielzahl weiterer Frameworks und Tools, die zum GPU Profiling eingesetzt werden können, aber hier der Übersichtlichkeit halber keine Erwähnung finden sollen. Es ist stets eine Vergleichbarkeit mit CPU-Profiling gegeben, mit dem Unterschied, dass Parallelität bei GPUs naturgemäß eine viel entscheidendere Rolle spielt.

6 Zusammenfassung

Mit dieser Arbeit konnte zu Beginn dargelegt werden, wie sich Graphical Processing Units (GPUs). Seit dem Beginn der Entwicklung dedizierter Grafikchips hat sich der Funktionsumfang dieser kontinuierlich erweitert und die Transistorenzahl stark erhöht. Außerdem konnte ein Wandel von reinen Grafik-Chips hin zu multifunktionalen GPGPUs identifiziert werden, der mit dem Einzug von Programmierschnittstellen wie CUDA seinen Anfang fand. Immer umfangreichere und effizientere Programmierschnittstellen wie OpenGL, DirectX und die Vulkan API werden in ihren neuen Versionen in der Regel mit neueren Architekturen der Grafik-Chips integriert. Die Leistungsfähigkeit der GPUs hat im Lau-

fe der Zeit stark zugenommen, wobei diese heute primär vom Anwendungsgebiet abhängt.

Die neuen Architekturen von NVidia und AMD, nämlich Volta und Vega, haben gemeinsam, dass sie schnellen HBM2 Speicher verwenden. Bei den Neuheiten der Volta-Architektur sind besonders die neue Streaming-Multiprozessor-Architektur mit Tensor-Kernen für Matrixmultiplikationen und bei der AMD Vega-Architektur die 64 Next-generation compute units (NCUs) und eine neue Speicherhierarchie mit eine Cache Controller für hohe Bandbreite (HBCC) hervorzuheben.

Die Leistungsfähigkeit profitiert von diesen Architektur-Veränderungen zum Teil erheblich. Neben allgemeinen Geschwindigkeitsvorteilen in Benchmarks, erreicht die AMD Radeon Pro SSG, eine auf Vega basierende GPU mit eigener SSD, eine rund fünfmal höhere Performance als eine herkömmliche Systemarchitektur, wodurch ein flüssiges Videoprocessing bei 8K ermöglicht wird. NVidias Volta erreicht mithilfe der Tensor-Kerne bei der Matrixmultiplikation gegenüber der Vorgängergeneration Pascal eine zwölffache Geschwindigkeit.

Bei der Programmierung, wie diese Arbeit darlegt, muss heute sehr stark in Programmierung für herkömmliche Grafikanwendungen und allgemeine Anwendungen differenziert werden.

Die Performance-Analyse erfolgt auch bei GPUs mithilfe von Profilern, die im Wesentlichen eine High-Level Analyse für den Programmierer bereitstellen, indem sie selbst zum Teil eine Low-Level Analyse durchführen. Entscheidend für das GPU-Profiling ist, ob Berechnungskosten Speicherkosten verdecken beziehungsweise umgekehrt oder mangelnde Parallelisierung vorliegt.

Weiterer Forschungsbedarf besteht zum Beispiel in der Performance-Analyse bei Machine-Learning, insbesondere bei und in Kombination mit den neuen Tensor-Kernen der Volta-Architektur.

Insgesamt zeigt sich mit den aktuellen GPU-Architekturen eine weitere Fortführung der Spezialisierung von GPUs und ein anhaltender Trend hin zu GPGPUs, wobei inzwischen sogar der Hauptfokus auf wissenschaftlichen und allgemeinen Anwendungen zu liegen scheint.

Literatur

[1] https://www.duden.de/rechtschreibung/GPU_Grafikprozessor_EDV (aufgerufen am 06.12.2017).

[2] http://www.3dcenter.org/artikel/launch-analyse-nvidia-geforce-gtx-titan-x (aufgerufen am 06.12.2017).

[3] https://www.nvidia.de/self-driving-cars/drive-px/ (aufgerufen am 06.12.2017).

[4] https://www.nvidia.de/data-center/dgx-1/ (aufgerufen am 07.12.2017).

[5] https://www.videocardbenchmark.net/high_end_gpus.html (aufgerufen am 14.12.2017).

[6] http://icl.cs.utk.edu/projects/papi/wiki/Counter_Interfaces (aufgerufen am 14.12.2017).

[7] https://gpuopen.com/gaming-product/radeon-gpu-profiler-rgp/ (aufgerufen am 14.12.2017).

[8] https://gpuopen.com/compute-product/codexl/ (aufgerufen am 14.12.2017).

[9] A. —. R. T. Group. *RADEON PRO - Solid State Graphics (SSG) Technical Brief*. Techn. Ber. https://www.amd.com/Documents/Radeon-Pro-SSG-Technical-Brief.pdf (aufgerufen am 26.11.2017).

[10] A. —. R. T. Group. *Radeon's next-generation Vega architecture*. Techn. Ber. https://radeon.com/_downloads/vega-whitepaper-11.6.17.pdf (aufgerufen am 26.11.2017). Juni 2017.

[11] P. Guo, L. Wang und P. Chen. "A performance modeling and optimization analysis tool for sparse matrix-vector multiplication on GPUs". In: *IEEE Transactions on Parallel and Distributed Systems* 25.5 (2014), S. 1112–1123.

[12] P. Micikevicius. "GPU performance analysis and optimization". In: *GPU technology conference*. Bd. 84. 2012.

[13] J. Nickolls und W. J. Dally. "The GPU computing era". In: *IEEE micro* 30.2 (2010).

[14] NVidia. *NVIDIA TESLA V100 GPU ARCHITECTURE. THE WORLD'S MOST ADVANCED DATA CENTER GPU"*. Techn. Ber. http://www.nvidia.com/object/volta-architecture-whitepaper.html (aufgerufen am 26.11.2017). Aug. 2017.

[15] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone und J. C. Phillips. "GPU computing". In: *Proceedings of the IEEE* 96.5 (2008), S. 879–899.

[16] E. Quinnell, E. E. Swartzlander und C. Lemonds. "Floating-point fused multiply-add architectures". In: *Signals, Systems and Computers, 2007. ACSSC 2007. Conference Record of the Forty-First Asilomar Conference on*. IEEE. 2007, S. 331–337.

[17] J. Sim, A. Dasgupta, H. Kim und R. Vuduc. "A performance analysis framework for identifying potential benefits in GPGPU applications". In: *ACM SIGPLAN Notices*. Bd. 47. 8. ACM. 2012, S. 11–22.

[18] N. P. Toolkit. "NVIDIA PerfKit". In: (2013).

[19] Y. Zhang und J. D. Owens. "A quantitative performance analysis model for GPU architectures". In: *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on*. IEEE. 2011, S. 382–393.