Using GPUs for Packet-processing

Beschleunigte Paketverarbeitung mit Hilfe von GPUs

Thomas Schultz
Betreuer: Daniel Raumer
Seminar Future Internet SS2014
Lehrstuhl für Netzarchitekturen und Netzdienste
Fakultät für Informatik, Technische Universität München
Email: schultz@in.tum.de

KURZFASSUNG

Software-Router sind den Hardware-Routern in Bereichen wie Flexibilität und Erweiterbarkeit weit voraus, doch haben sie trotz moderner Hardware immer noch Probleme das Leistungspotential auszuschöpfen. Um effizientere Systeme zu entwickeln, versucht man die Leistung von Grafikprozessoren hierfür zu nutzen. Im folgender Abhandlung wird die Motivation für diesen Schritt als auch die Hindernisse auf dem Weg dorthin beschreiben. Anschließend werden zwei Implementierung auf Basis dieser Technik vorgestellt.

Schlüsselworte

software router, gpgpu

1. MOTIVATION:

Computernetzwerke erfüllen heutzutage eine wichtige Rolle in der globalen Kommunikation. Um die stetig wachsenden Datenmengen zu bewältigen werden immer neue Techniken benötigt. Auf der eine Seite betrifft das die reine Übertragungstechnik, basierend auf kabellosem Funk oder kabelgebunde Kupfer- bzw. Glasfaserkabel. Über Glasfaserverbindungen lassen sich dabei die höchsten Datenraten erzielen. Aktuell wird an Linkgeschwindigkeiten von 100 Gbit/s gearbeitet [1]. Auf der anderen Seite muss auch die Verarbeitung der Daten zwischen den Kommunikationspartnern schneller werden, um diese Geschwindigkeiten auch auszunutzen. Diese Verarbeitung, das Packet-processing, ist von fundamentaler Bedeutung für die Infrastruktur in großen Netzwerken wie zum Beispiel dem Internet. So benutzt jeder die großen Backbone-Netzwerke der Internetprovider, wenn er über seinen Internetanschluss Daten empfängt oder überträgt.

Gerade wenn es um kommerzielle Ziele geht, ist es von Vorteil, wenn sich das Verhalten dieses Netzes besser kontrollieren lässt. Dadurch könnten vertraglich festgelegte Leistungen gegen einen Aufpreis zugesichert werden. Gerade in jüngerer Vergangenheit, bei den Diskussionen um die Netzneutralität, offenbarte die Deutsche Telekom Pläne, wonach sie gezielt eigene Dienste wie IPTV (T-Entertain [9]) bevorzugen wollte. Derzeitige Systeme ermöglichen aber nur begrenzten Einfluss auf die Verarbeitung der Datenpakete. Die Betreiber sind hier an den gebotenen Funktionsumfang der Hersteller gebunden. Ohne diese lassen sich Änderungen an vorhandenen Systemen sehr schwer oder gar nicht durchsetzen. An dieser Stelle sind Software-Router proprietären Hardware-Routern, zum Beispiel von Cisco, in der Konfigurierbarkeit überlegen. Ihre Software ist in weiten Teilen unabhängig von der Hardware und kann so leicht erweitert oder gar ausgetauscht werden, um neue Funktionen zu implementieren.

Software-Router sind im Prinzip normale Computer, wie man sie im Büro oder zu Hause stehen hat. Ihr Vorteil besteht zu großen Teil in ihrer Flexibilität. Die benötigte Hardware ist standardisiert, überall zu erwerben und preislich um einiges günstiger als spezielle Netzwerk-Hardware. Zum Anderen ist die Paketverarbeitung in Software implementiert, die in den meisten Fällen quell-offen, also *open-source*, verfügbar ist. Damit lassen sich die Kosten einsparen und eigene Anpassungen bzw. Optimierungen für das Anwendungsgebiet vornehmen.

Das Problem dabei ist, dass Software-Router trotz der heutigen Rechensystem noch immer nicht die Leistung erbringen, um kommerziell in den Markt der etablierten Hardware-Router einzudringen. Immer neuere Forschungsprojekte belegen das große Interesse und Potential von Software-Routern [1] [2] [4]. Darunter sind ein paar Arbeiten, die sich mit parallelisierter Verarbeitung von Paketen mit Hilfe von Grafikkarten befassen. Dass moderne Grafikkartenarchitekturen nicht nur für Computerspiele geeignet sind, beweist die Top500 Liste [2] der weltweiten Supercomputer. In der Top10 finden sich bereits mehrere Systeme, die zur Steigerung der Rechenleistung auf Beschleunigungskarten auf Basis von Grafikprozessoren (GPUs) setzen. Zusätzlich entwickelt sich die Leistungsfähigkeit von GPUs im Gegensatz zu herkömmlichen Prozessoren (CPUs) schneller [7].
Im folgenden soll geklärt werden, warum sich GPUs für Paketverarbeitung eignen und wie man sie richtig einsetzt.

2. SYSTEM-ARCHITEKTUR

Um zu verstehen wie GPUs für die Verarbeitung von Netzwerkpaketen verwendet werden können, muss das gesamte System betrachtet werden. Ein einfacher Software-Router besteht, wie jeder PC, aus einem oder mehrerer Prozessoren, einem Arbeitsspeicher und geeigneten Netzwerkkarten. Dabei kann man die zugrundeliegende Hardware in zwei Klassen unterteilen: homogene und heterogene Systeme.

[1] IEEE P802.3bm Task Force `http://www.ieee802.org/3/bm/`

[2] Top500 Supercomputers: `http://www.top500.org/`

doi: 10.2313/NET-2014-08-1_05

2.1 Homogene Systeme

In einem homogene System befinden sich nur Komponenten einer einzelnen bestimmten Architektur. Die meist verbreitete und somit bekannteste ist die x86-Architektur von Intel. Diese basiert auf Intels Mikroprozessor 8086 aus dem Jahr 1978 und erbte dessen Namensschema. Trotz vieler Erweiterungen ist sie heute immer noch Basis moderner Mikroprozessoren von Intel sowie dem Konkurrenten AMD. Nebenbei gibt es auch andere bekannte Architekturen, zum Beispiel ARM, welche heutzutage in fast jedem Smartphone verwendet wird.
Ein Software-Router auf einem solchen System ist folglich an die verfügbaren Fähigkeiten dieser Architektur gebunden. Die x86-Architektur ist historisch bedingt auf komplexe, seriell zu verarbeitende Befehle optimiert. Multicore-Unterstützung wurde erst Anfang der 2000 Jahre relevant. Um den Sachverhalt zu vereinfachen, könnte man die x86-Architektur als gutes Allzweckwerkzeug bezeichnen, welches für die meisten Aufgaben gut, jedoch für wenig optimal geeignet ist.

2.2 Heterogene Systeme

Mit aufkommenden Bedarf an Leistungsverbesserungen wurde der Einsatz von spezielleren Architekturen vorangetrieben. In einem heterogenen System sind mehrere Komponenten unterschiedlicher Bauarten verbunden, um je nach Anforderung auf diese spezielle Hardware zurück zu greifen. Das beste Beispiel hierfür sind moderne Heimcomputer oder gar Notebooks. Neben einem Prozessor sind meistens dedizierte, eigenständige Grafikkarten verbaut. Diese vornehmlich für Spielzwecke entwickelten Ergänzungskarten haben sich mittlerweile auch einen Platz im wissenschaftlichen Umfeld geschaffen. Bei Verwendung von GPUs für Rechenoperationen, anstatt zur Grafikausgabe spricht man auch vom Begriff der *General Purpose Computation on Graphics Processing Unit* kurz GPGPU.
Durch ihre rapide Entwicklung und immensen theoretischen Rechenleistung, die um mehrere Faktoren größer ausfällt als die von leistungsstarken CPUs der selben Generation sind sie ein beliebtes Werkzeug. Seitdem NVIDIA und AMD (ehemals ATI) diesen Trend erkannt haben, bieten beide dementsprechend leicht modifizierte Karten für diesen Markt an. Nebenbei wurden auch angepasste Programmierschnittstelle (dazu später mehr in 2.4.1) veröffentlicht.

2.3 Software-Router

Betrachtet man einen Software-Router zunächst als Black-Box, benötigt dieser drei Elemente: die Netzwerkports, einen Verarbeitungspuffer sowie eine Verarbeitungslogik. Bildet man das auf einen Computer ab erhalten wir die wichtigen Bestandteile eine Software-Routers: Netzwerkinterfaces (NICs), Arbeitsspeicher (RAM) und Prozessoren. Wie der Name verrät, erledigt ein Software-Router seine Funktionalität durch den Kernel des Betriebssystems oder Programmen im Userspace. Dieser Schritt erfordert jedoch, dass jedes Netzwerkpaket im Arbeitsspeicher liegt, damit die CPU auf die Daten bzw. Header zugreifen kann. Folglich entstehen bei der Verarbeitung drei potentielle Schwachstellen:

1. I/O: Die Netzwerkkarte muss immer alle Pakete annehmen können
2. RAM: Lese- und Schreiboperationen müssen schnell genug geschehen

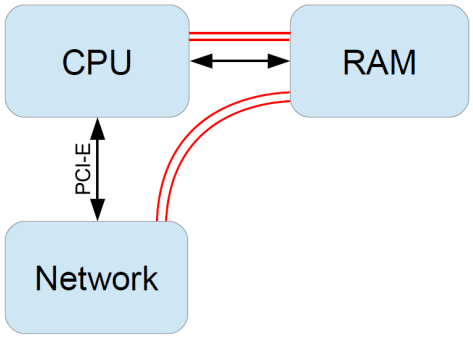

Abbildung 1: Vereinfachte Darstellung der Speicherzugriffe

3. CPU: Zügige Bearbeitung zur Vermeidung von Paket-Staus

Um einen groben Überblick über die Problematik zu erhalten, wird auf alle drei Punkte kurz eingegangen.

2.3.1 I/O Bandbreite

Heutige x86-Systeme unterstützen in der Regel den PCI-Express [3] Standard. Diese Schnittstelle wird zur Anbindung zusätzlicher Hardware in einem Computer verwendet. Grafikkarten, Soundkarten, spezielle SSDs und auch Netzwerkkarten verwenden diese Technik zur Kommunikation. Seit der Einführung im Jahr 2003 hat sich die mögliche Bandbreite dieser Schnittstelle stetig gesteigert. Zusätzlich ist die Ansteuerungslogik mittlerweile vom Chipsatz in die CPU selber gewandert. Um größere Flexibilität zu erlauben, ist der Standard modular aufgebaut. Es gibt verschieden große Steckplätze auf dem Mainboard, die jeweils mit 1, 2, 4, 8 oder 16 sogenannten Lanes angeschlossen werden. Dabei ist zu beachten, dass physikalisch und elektronisch Abwärtskompatibilität besteht, d.h eine x1 Karte passt in einen x16, umgekehrt jedoch nicht. Der aktuelle Standard, PCI-Express 3.0, ist in der Lage auf einem vollen 16-Lanes Steckplatz knapp 16 GB/s voll-duplex zu übertragen.
Eine über PCI-Express angeschlossene Netzwerkkarte muss also mindestens die Bandbreite für ihre Linkgeschwindigkeit samt Overhead zur Verfügung haben, oder sie wird bereits in der Performance gebremst. Für eine einzelne Netzwerkkarte mag dies noch kein Limit darstellen, kann aber bei größerer Anzahl und je nach Geschwindigkeit bzw Anzahl physikalischer Anschlüse (Ports) zu einem Problem werden. Jedes System kann nur eine bestimmte Zahl an PCI-Express-Lanes zur Verfügung stellen, so dass auch hier die maximale Bandbreite auf alle Erweiterungskarten aufgeteilt werden muss. Je nach Anzahl kann es passieren, dass die Steckplätze auf dem Mainboard nur mit der Hälfte oder weniger Lanes betrieben werden.

2.3.2 Speicher

In einer Von-Neumann-Architektur müssen alle zu verarbeitenden Daten im Arbeitsspeicher liegen. Seitdem Intel mit der Core-i Generation mit AMD gleichgezogen hat, und den RAM-Controller in die CPU integriert hat, verwenden beide die gleiche Topologie. Die einzelnen Arbeitsspeicher Module

[3] PCI-Express Spezification: `http://www.pcisig.com/specifications/pciexpress/`

doi: 10.2313/NET-2014-08-1_05

sind direkt mit dem Controller verbunden und müssen nicht mehr wie früher über einen gemeinsamen Systembus angesprochen werden.
Durch diverse Techniken, darunter DDR3 (DDR4 ist bereits für 2014 angekündigt) und die Verwendung von zum Beispiel Quad-Channel (paralleler Zugriff auf vier Speichermodule) ist die mögliche Bandbreite stetig gewachsen. Dennoch ist der Arbeitsspeicher eine kritische Systemkomponente eines Software-Router. Selbst im optimalen Fall, siehe Abbildung 1 führt ein Paket zu vier Zugriffen auf dem Speicher-Controller: Netzwerkkarte schreiben, CPU lesen, CPU schreiben, Netzwerkkarte lesen. Zwar lässt sich die Datenmenge dieser Zugriffe optimieren, falls zum Beispiel nicht das komplette Paket sondern nur der Header benötigt wird, jedoch gibt es keinen generellen Ausweg aus dieser Problematik.

2.3.3 Prozessor

In einen Software-Router müssen alle Entscheidungen und Berechnungen von einem Prozessor erledigt werden. Dieser muss entweder Teile oder auch das gesamte Paket lesen, um die konfigurierten Regeln anzuwenden. Für einen einfachen IP-Router besteht diese Aufgabe mindestens aus einem Nachschlag in der Routing-Tabelle und dem Herabsetzten des TTL-Wertes.
Der große Vorteil gegenüber Hardware-Routern ist jedoch die große Flexibilität der Software. Dadurch sind auch kompliziertere Anwendungsfälle einfacher zu realisieren. Virtual-Private-Networks (VPNs) zum Beispiel benötigen mehr Aufwand durch die Verkapselung von Datenpaketen und den nötigen kryptografischen Rechenoperationen. Je nach Komplexität der Konfiguration kann der Rechenaufwand für die CPU um Faktoren steigen, sodass diese die Performance des Software-Routers beeinträchtigt.
Eine Prozessor mit nur einem Kern muss alle empfangenen Pakete bearbeiten. Die Zeit die er dafür maximal benötigen darf, ist durch die Linkgeschwindigkeit und die Paketgröße gegeben. Verteilt man die Arbeit auf mehrere Kerne erhöht sich diese Zeit, jedoch kommen sich die einzelnen Kerne bei Zugriff auf Speicher bzw. dem gemeinsamen Cache in die Quere.

2.4 GPGPU

Will man nun einen Software-Router mit Hilfe von GPUs beschleunigen, sind einige wichtige Aspekte zu beachten. In erster Linie sind das die fundamentalen Unterschiede zwischen einem CPU- und einem GPU-Kern. Im Folgenden werden beide Begriffe auch für den physikalischen Chip, dem sogenannten *Die* verwendet.

Bei einer modernen CPU befinden sich bereits mehrere Kerne auf einem Chip (z.B. Intel XEON Server CPUs mit bis zu 15[4]). Neben diesen Rechenkernen wird ein Großteil der Transistoren für den Cache benötigt. Dieser Cache wird zur Pufferung von Daten aus dem Arbeitsspeicher als auch für die Synchronisation unter den CPU-Kernen verwendet. Moderne CPUs verwenden eine Vielzahl von Funktionen (Pipelining, Out-of-Order Execution, etc) um die Ausführung von Befehlen zu beschleunigen. Für diese Optimierungen werden zusätzliche Transistoren neben den eigentlichen Recheneinheiten, den *arithmetic logic units* (ALUs), benötigt, wodurch eine CPU noch komplexer wird.

Eine GPU ist dagegen anders aufgebaut. Eine handelsübliche Grafikkarte verfügt auf ihrer Platine über einen eigenen dedizierten Speicher, der im Vergleich zu dem Arbeitsspeicher des Hosts eine größere Bandbreite besitzt. Der Großteil der GPU selber besteht aus vielen simplen Rechenkernen die für paralleles Arbeiten ausgelegt sind. Diese SIMD-Einheiten (Single Instruction, Multiple Data) führen eine Operation auf einem Vektor an Daten aus. Aus diesem Grund verfügen GPU auch über eine deutlich höhere theoretische Rechenkraft als CPUs. Des Weiteren bedient man sich bei Grafikkarten eines weiteren Tricks. Möchte die CPU auf Daten zugreifen, die zur Zeit nicht mehr im Cache liegen, verliert man wertvolle Zyklen bis die Daten aus dem Arbeitsspeicher geladen wurden. Alternativ kommt ein anderer, wartender Thread an die Reihe. So ein Kontextwechsel ist allerdings mit Overhead verbunden, da alle Register zurück in den Cache geschrieben und neue Daten geladen werden müssen.
Bei GPUs hingegen führt ein Wechsel zu kaum Verlusten, sodass bei Speicherzugriff einfach ein anderer Thread ausgeführt wird. Diese Technik nennt sich Latency-Hiding. Damit eine GPU effektiv ausgelastet ist, müssen also immer mehr Threads vorhanden sein als verarbeitet werden. Bei Kernzahlen um die 2500 (NVIDIA GK110 bis 2880 [6]), können das unter Umständen zehntausende sein.

2.4.1 Verwendung von GPUs

Obwohl der Einsatz von Grafikkarten zur Beschleunigung mittlerweile weit verbreitet ist, ist ihre Verwendung umständlich. Programme laufen grundsätzlich erst auf dem Prozessor des Hosts. Um die Rechenleistung der GPU zu nutzen, müssen die Daten auf den getrennten Speicher der Beschleunigungskarte kopiert, deren Ausführung initiiert werden und nach Abschluss wieder in der Arbeitsspeicher des Hosts zurückgeschrieben werden. Dabei wird sowohl der Speicher-Controller sowie die PCI-Express Schnittstelle belastet. Bei diesen Kopiervorgängen muss darauf geachtet werden, dass entsprechende Daten im Cache der CPU als ungültig erklärt werden, damit die Datenkonsistenz gewährleistet bleibt.

Für Programmierer von GPU-beschleunigtem Code stehen diverse Frameworks zur Verfügung. Diese Frameworks stellen dem Anwender eine abstrahierte Schnittstelle zwischen der Hardware und gängigen Programmiersprachen zur Verfügung. Als bekannteste Vertreter sind hier NVIDIA's CUDA und OpenCL zu nennen. Beide unterscheiden sich hinsichtlich ihrer Implementierung und der unterstützen Hardware, sind aber vom Prinzip sehr ähnlich. Während CUDA nur auf Produkten von NVIDIA selber läuft, ist OpenCL unabhängig von der verwendeten Plattform. Im Folgendem wird das Prinzip dieser Schnittstellen anhand von OpenCL erklärt.

Mit OpenCL[13] lassen sich mehrere Ressourcen in einem System vereinen. Dazu wird die Hardware nach OpenCL fähigen Geräten durchsucht. Diese sind in der Regel in unabhängige Recheneinheiten aufgeteilt. Das Hauptprogramm hat nun die Aufgabe, die Verwendung dieser Recheneinheiten zu steuern. Dazu werden die Daten in sogenannte Work-Items aufgeteilt und an die Recheneinheiten verteilt. Für

[4]Intel Xeon E7 Family: `http://ark.intel.com/products/family/78584`

doi: 10.2313/NET-2014-08-1_05

die Ausführung werden sogenannte Kernels aus dem Programmcode erzeugt, die dann die Work-Items verarbeiten. Dieses Hauptprogramm kann in beliebiger Sprache geschrieben sein, die OpenCL unterstützt. Der OpenCL Code, angelehnt an C99, wird erst während der Laufzeit für die verwendete Hardware compiliert und dann auf dieser ausgeführt.

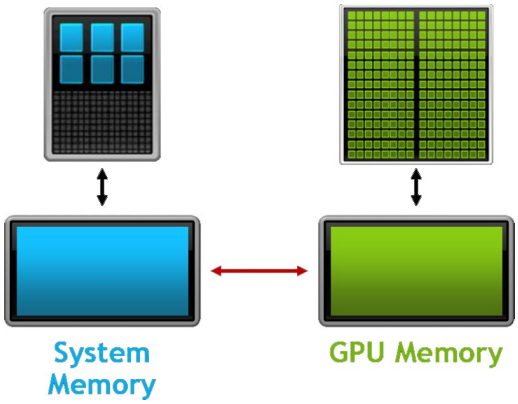

Abbildung 2: Vorherrschendes Speicherkonzept (Quelle [10])

2.4.2 *Unified Memory*

Um die Nachteile dieser getrennten Speicherbereiche zu beseitigen, strebt man Systeme auf Basis gemeinsamer Speicher (Unified-Memory) an. Bei einem solchen System können CPU und GPU auf einen gemeinsamen Speicher zugreifen. Dadurch werden Kopiervorgänge zwischen beiden Einheiten unnötig. Auch aus Sicht der Programmierers gestaltet sich die Verwendung einfacher, da statt den Daten selber einfach Pointer übergeben werden können. In Abbildung 2 ist das zur Zeit verwendete Konzept dargestellt. Alle Daten müssen hier zwischen den beiden Speicherblöcken übertragen werden, hier angedeutet durch den roten Pfeil. Abbildung 3 hingegen besitzt nur einen Speicherblock auf den von beiden zugegriffen werden kann.

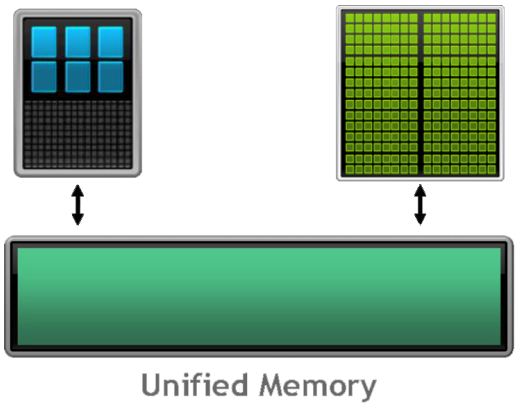

Abbildung 3: Unified Memory Konzept (Quelle [10])

In der Praxis sind solche Systeme selten anzutreffen, da die Architektur schwierig zu implementieren ist. Um Kohärenz zu bewahren müssen sich CPU und GPU bei Zugriff auf den gemeinsamen Speicher absprechen. Der gemeinsame Speicher-Controller muss mit seiner Bandbreite beide Kerne gleichzeitig bedienen. Verwendet man, wie üblich, gewöhnliche Speichertechnologie wie DDR3 entsteht hier ein Engpass. Dadurch sind die eigenständigen Beschleunigungskarten in Bereich der Speicherbandbreite deutlich im Vorteil. Hier müssen die Daten zwar erst per PCI-Express auf diesen gelangen, doch von dort erfolgt der Zugriff deutlich schneller. Das wiederum heißt Softwareentwickler müssen sich hier weniger Sorgen um Speicherzugriffe während der Bearbeitung machen.

Auf dem Markt gibt es dennoch Vertreter dieser Art. Seit dem AMD den Grafikkartenhersteller ATI aufgekauft hat, produzieren sie Mikroprozessoren auf denen neben x86 basierten CPU-Kernen auch GPU-Kerne verbaut sind. Dieses Prozessorkonzept verkauft AMD unter den Namen *Heterogeneous System Architecture* kurz HSA [5]. Die von AMD *Accelerated Processing Units* (APUs) genannten Modelle verwenden den sonst nur für die CPU ausgelegten Arbeitsspeicher für beide Arten von Kernen.

Die Leistung dieser Kombination kommt allerdings noch nicht an die Leistung von eigenständigen Beschleunigungskarten heran. Dieser Umstand ist auch nicht sonderlich verwunderlich, wenn man die thermische Verlustleistung betrachtet. Während moderne Grafikkarten auf komplizierte Kühlkonzepte setzen, um den Chip, Speicher und deren Stromversorgung innerhalb der thermischen Grenzen z halten, muss sich eine integrierte Grafikeinheit diese Ressourcen mit dem Hauptprozessor teilen. Trotzdem ist die Idee der Vereinigung beider Architekturen auf Chip-Level statt auf System-Level ein vielversprechender Ansatz. Dadurch können PCI-Express Steckplätze statt für Beschleunigungskarten für Netzwerkkarten verwendet werden, während die Rechenleistung durch Multi-Sockel Systeme bereitgestellt wird. Da es sich allerdings um eine relativ jungen Entwicklung handelt fehlen bisherige Ergebnisse in Bezug auf Software-Router.

3. PACKET-PROCESSING

Die Verarbeitung von Netzwerkpaketen ist ein relativ junges Einsatzgebiet für Beschleunigungskarten auf Basis von GPUs. Ein Leistungssteigerung kann nur erreicht werden, wenn sich die Berechnungen genügend parallelisieren lassen. Im Fall von Software-Routern, lässt sich die Verarbeitung von einzelnen Paketen sehr gut parallelisieren. In den folgenden Abschnitten wird nun auf die Besonderheiten dieses Einsatzgebiets eingegangen.

3.1 Zero-Copy Mechanismen

Eine Möglichkeit die interne Verarbeitung zu beschleunigen sind sogenannte Zero-Copy Mechanismen. Im Prinzip versucht man mit dieser Strategie, unnötiges Kopieren der Daten von einer Stelle im Speicher zu einer anderen zu vermeiden. Wie bereits im Abschnitt 2.3.2 beschrieben, führen solche Speicherzugriffe zu Bandbreitenverbrauch sowie größere Verzögerung in der Verarbeitung.
Benutzt man nun separate Grafikkarten mit eigenständigem Speicher muss man sich zwangsläufig Gedanken machen, welche Daten man in den Kopiervorgang einbezieht. Es gibt bereits einige Tools [8], die diese Technik einsetzen und gezeigt haben, dass hier ein großes Potential liegt.

3.2 Paralleles Bearbeiten

Der Grund warum die meiste Software von Mehrkern-Systemen nur begrenzt profitiert, sind Daten- oder Kontrollabhängigkeiten. Sobald die Ausführung von Code-Segmenten

doi: 10.2313/NET-2014-08-1_05

in irgendeiner Art von dem Ergebnis vorheriger Berechnungen abhängen, muss diese Information erst intern kommuniziert werden. Je mehr Kerne verbaut sind, desto größer wird generell dieser Kommunikations-Overhead und bedingt so nur eine mäßige Skalierung.
Bei Netzwerkpaketen ist in der Regel ein Datenpaket unabhängig von seinem Vorgänger. Damit lässt sich der Rechenaufwand auf mehrere Rechenkerne aufteilen, ohne dass es zu Synchronisationsproblemen führt. Durch diesen Umstand lässt sich die Verarbeitung durch Grafikkarten beschleunigen.

Blockweise Bearbeitung

Wie bereits erwähnt, sind GPUs auf die parallele Ausführung optimiert. Im Gegensatz zu den meisten wissenschaftlichen Aufgaben die auf GPGPU laufen, enthalten Netzwerkpakete nur relativ wenig Daten, die nur kurze zur Bearbeitung in der GPU verweilen. Initiiert man also für jedes eingehende Paket einen GPU-Thread wird viel Zeit und Bandbreite für die Koordination verbraucht [3]. Überträgt man hingegen in einem Aufruf mehrere Datensätze, so kann man den Mehraufwand pro einzelnes Paket stark senken.
Als Konsequenz daraus, erhält man erst einen Vorteil, wenn man Paket-Blöcke erstellt. Auf diese Weise wird die Auslastung der Systemschnittstellen reduziert und gleichzeitig kann garantiert werden, dass die GPU effizient ausgenutzt wird. Damit ein Software-Router also von einer Grafikkarte profitieren kann, müssen mehrere Netzwerkpakete in einem Puffer gesammelt und erst anschließend als ein Block an die GPU übergeben werden.

Messungen aus [11] mit verschiedenen Blockgrößen ergeben einen Leistungszuwachs ab ein paar hundert Paketen pro Block. Dabei ist zu beachten, dass dieser experimentell erprobte Wert von dem Funktionsumfang abhängt. Ebenso ist ein klarer Trend festzustellen: je größer die Blöcke werden, desto größer fällt der Leistungszuwachs aus. Auch die Daten aus [3] bestätigen die Größenordnung dieses Wertes.

Verzögerung

Das Zusammenfassen von Netzwerkpaketen vor der Verarbeitung erzeugt jedoch Zeit-bedingt eine gewisse Verzögerung. Um einen guten Kompromiss zwischen zeitlicher Verzögerung und Verarbeitungsgeschwindigkeit zu erreichen, ist die Blockgröße dementsprechend auszulegen. Dabei begünstigen schnellere Übertragungsgeschwindigkeiten das Problem, da pro Pakete geringere Zeitintervalle benötigt werden.
Zusätzlich zu der Größe der einzelnen Blöcke sollte auch ein Time-out konfiguriert werden. Für den Fall, das keine Pakete mehr ankommen, würde der letzte Block solange wartend verweilen, bis die nötige Anzahl erreicht ist. Durch hinzufügen eine Time-outs kann sicher gestellt werden, dass die Verzögerung auch in diesem Fall nicht höher ausfällt.

Die Autoren von [11] haben in ihren Test ebenso die Verzögerung gemessen. Hierfür stellten sie Messung der Paketumlaufzeit (Round Trip Time or RTT) mit ausschließlich Verarbeitung auf der CPU sowie mit Blöcken variabler Größen auf der GPU an. Bei eine Blockgröße von 1024 ergaben sich Werte von knapp dem zehnfachen von denen auf der CPU. Reduzierung dieser Zeit ist nur auf Kosten des Durchsatz möglich.

Reihenfolge

Ein weiteres Problem paralleler Verarbeitung ist die Reihenfolge. In einer strikt seriellen Verarbeitung wird wie in einer FIFO-Warteschlange alle eingehenden Pakete der Reihe nach bearbeitet. Das wiederum heißt, die Reihenfolge der Pakete, beispielhaft einer TCP Verbindung, bleibt erhalten. Werden die Pakete hingegen auf verschiedene Threads auf der Grafikkarte aufgeteilt, so ist die Reihenfolge ihrer Bearbeitung unbestimmt. Soll neben einem IP-Router auch noch weitere Funktionalitäten wie einer Firewall oder ähnliches integriert werden, muss dafür gesorgt werden, dass die Verarbeitung das gleiche Ergebnis wie die serielle Verarbeitung liefert.
Ein Möglichkeit die Reihenfolge nicht zu verändern, ist die Zuteilung eingehender Pakete anhand ihrer Ursprungs- und Ziel-Adresse. Gelangen alle Pakete mit den gleichen Eigenschaften in einen Block, kann garantiert werden, dass die Reihenfolge erhalten bleibt. In der Regel wird dies durch eine geeignete Hash-Funktion gelöst, die alle möglichen Pakete auf die zu Verfügung stehenden Ressourcen aufteilt.

4. BESPIELE

Leistungssteuerung und Messung von Software-Routern ist eine komplexe Angelegenheit. Zum Einen werden ständig neue Technologie entwickelt die wiederum bisherige Ergebnisse relativieren. Zum Anderen ist die verwendete Software, darunter Betriebssystem, Treiber und Applikation selber, in ständigem Wandel. Aus diesem Grund kann es vorkommen, dass verschiedene Veröffentlichungen unterschiedliche Erkenntnissen liefern. Die Optionen sind meist so vielfältig, dass trotz genannter Details oft kein direkter Vergleich untereinander möglich ist.
Will man zusätzlich noch die Effektivität von GPUs bewerten, ist viel Erfahrung mit Software-Routern sowie der Hardware nötig. Aus diesem Grund gibt es relativ wenig Vielfalt bei diesen Arbeiten, wovon hier kurz zwei Ansätze vorgestellt werden.

4.1 Snap und Click

Snap [11] ist eine GPU-basierte Erweiterung für Click (*click modular router* [5]). Click ist ein Framework für Software-Router Konfigurationen für Linux. Click basiert auf einer graphenorientierte Beschreibung. Der Router besteht aus einer Anordnung von Knoten, den Elementen, welche die Funktionen repräsentieren und Verbindungen, welche mögliche Pfade für Netzwerkpakete darstellen. Abbildung 4 veranschaulicht einen simplen Graphen, bei dem alle am Interface *eth0* ankommenden Pakete gezählt und anschließend verworfen werden. Zur Erstellung einer Konfiguration wird eine Textdatei genutzt, die den Graphen beschreibt.

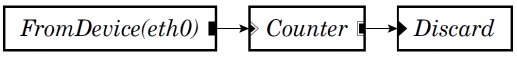

Abbildung 4: Router zählt Pakete und verwirft alle (Quelle: [4])

Click alleine besitzt keine Unterstützung für die Verwendung von GPUs, weshalb Snap entwickelt wurde. Snap fügt eine neu Art von Elementen hinzu, die direkt auf von der GPU arbeiten. Zu diesem Zweck wurde der Quellcode von Click

[5]Click Modular Router: `http://www.read.cs.ucla.edu/click/click`

doi: 10.2313/NET-2014-08-1_05

nur an wenigen Stellen modifiziert und hauptsächlich um die neuen Elemente ergänzt.
Um die Flexibilität von Click zu bewahren stehen weiterhin alle ursprünglichen Elemente zur Verfügung. Sollen Pakete jetzt auf der GPU verarbeitet werden gibt es zwei Schnittstellen: Host-to-GPU und GPU-to-Host. Dazwischen werden Elemente platziert die direkt auf der GPU ablaufen. Snap verwendet die erste in Abschnitt 3.2 beschriebene Technik, um Pakete blockweise in den Grafikkartenspeicher zu kopieren. Ebenso wird eine FIFO-Warteschlange darüber geführt, welche Blöcke in der GPU in Bearbeitung sind.

Sind die Berechnungen aller Pakete in einem Block fertig, wird dieser als ganzes wieder in den Arbeitsspeicher zurück kopiert. Anschließend wird überprüft ob es einen Block in der Warteschlange gibt, der vorher eingefügt wurde, und somit auf dessen Verarbeitung gewartet werden muss. Ist die Reihenfolge korrekt, können wieder einzelne Pakete aus dem Block für weitere Verarbeitung auf der CPU freigegeben werden. In Abbildung 5 sind die einzelnen Schritte dieser beiden Schnittstellen dargestellt.

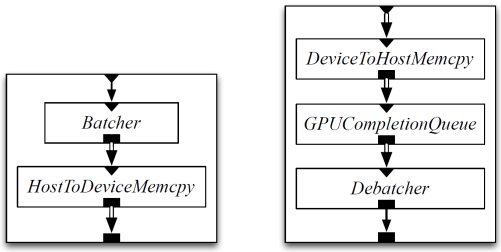

Abbildung 5: Snap GPU-Schnittstellen (Quelle: [11])

Zur Vermeidung unnötiger Kopiervorgänge lassen sich in Snap sogenannte Regions of Interest (ROIs) innerhalb der Netzwerkpakete spezifizieren. Mit diesem Trick muss nur das zur Verarbeitung relevante Teil der Header kopiert werden. Sollten diese Bereiche durch mögliche Optionen im Header an dynamischen Positionen liegen, muss diese bereits vor dem Kopiervorgang bestimmt werden. Im Gegensatz zu Zero-Copy Mechanismen können so trotz Kopiervorgängen die Datenmengen klein gehalten und der Durchsatz gesteigert werden.
Durch die pipelineartige Verarbeitung entlang des Graphen entsteht ein weiteres Problem: was geschieht mit Paketen in einem Block, die unterschiedlich bearbeitet werden müssen? Die Autoren von Snap [11]haben dazu Messungen von zwei Konzepten gemacht. Die eine Möglichkeit bestand in Aufsplittung des Pfades auf der GPU. Durch die verwendete SIMD-Architektur von GPUs reduziert sich durch das Aufsplitten die Leistung, da nicht mehr alle Daten die gleichen Verarbeitungsschritte durchlaufen. Die zweite Möglichkeit ist das Klassifizieren von Paketen und Anhängen von Metadaten. Danach durchlaufen alle Pakete jedes Element, wobei nur die zu den Metadaten passenden bearbeitet werden. Die Klassifizierung kann dabei sowohl in dem von der GPU bearbeiteten Bereich als auch in dem der CPU stattfinden. Abbildung 6 verdeutlicht die Klassifizierung und zeigt wie eine GPU-Pipeline genutzt werden kann um divergente Pfade zu ermöglichen. Die Messungen ergaben, dass die Klassifizierung höhere Leistung erzielt, obwohl jedes Paket alle Element durchläuft.

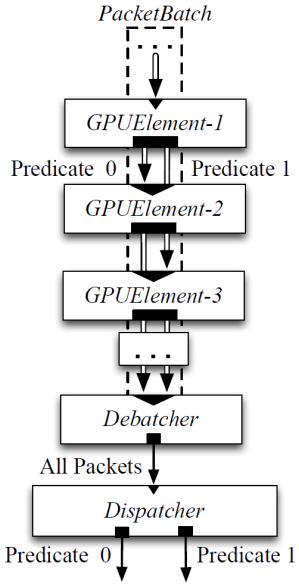

Abbildung 6: Konvergente Pfade der GPU-Pipeline (Quelle: [11])

4.2 PacketShader

PacketShader [3] ist im Gegensatz zu Snap, welches eine Erweiterung zu einem weit verbreiteten Framework darstellt, ein eigenständiges. Es ermöglicht ebenfalls wie Snap eine Programmierung des Routers im user-Level. Hierfür entwickelten die Autoren drei wichtige Komponenten: eine im Kernel integrierte Schnittstelle zur Kommunikation mit den NICs, eine auf Multithreading ausgelegte Steuerungslogik und den Verarbeitungsprozess.

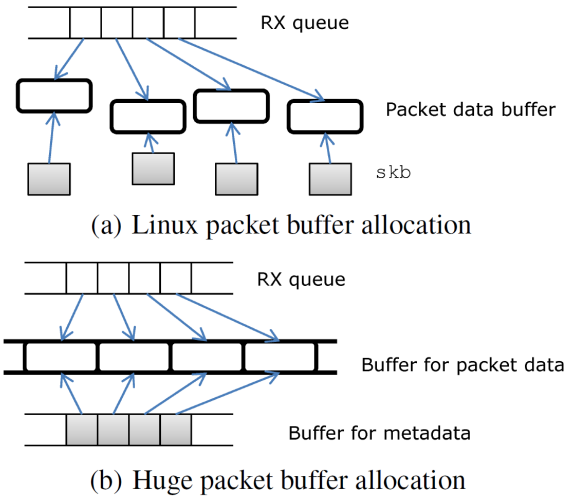

Abbildung 7: Pufferzuweisung von PacketShader (Quelle: [3])

Die Netzwerk-Schnittstelle wurde gegenüber dem Linux Kernel grundlegend verändert. Während Linux für jedes empfangene Paket ein eigenen Pufferbereich samt Metadaten anlegt, werden nun zwei großer Ringpuffer mit festen Zellengrößen für Daten und Metadaten bereitgestellt. Jedem Element im Empfangspuffer wird nun eine Position innerhalb des Ringes zugewiesen. Durch diese Veränderung lässt sich

die Fragmentierung der Speicherbereiche effizient umgehen, da Pakete auch im Speicher bündig aneinander liegen. Durch die vertikale Streuung oben, sowie die strikte Anordnung der Puffer unten, versucht die Abbildung 7 diese Anpassung grafisch zu veranschaulichen. Da die Pakete nicht mehr durch den Netzwerk-Stack von Linux müssen, konnte zusätzlich die Größe der Metadaten von 208 auf 8 Bytes reduziert werden. Somit werden nur noch die für einen Software-Router relevanten Daten gespeichert.

Zur Kommunikation zwischen Host bzw. CPU und GPU verwendet PacketShader spezielle Threads. Messungen belegten eine Verschlechterung der Leistung, wenn zu viele Threads mit der GPU kommunizieren wollen. Gelöst wurde dieses Problem durch Unterteilung der Threads in 'Master' und 'Worker'. Ein Master-Thread ist ausschließlich zu Kommunikation zwischen einem CPU-Kern und der GPU dar. Ein Worker-Thread regelt hingegen die Pakete und den damit verbundenen I/O. Dabei dient der Master-Thread als Vermittler beim Zugriff auf die GPU. Um Cache-Misses gering zu halten, arbeitet jeder Thread nur auf einem ihm zugewiesenen CPU-Kern.

Die Verarbeitung ist wiederum die höchst gelegene Steuerinstanz von PacketShader und ist wiederum in drei Abschnitte gegliedert:

- **pre-shader:** In diesem Schritt wird ein Block aus einem Warteschlangenpuffer geladen, klassifiziert die enthaltenen Pakete und erstellt eine neue Datenstruktur um sie auf die GPU zu kopieren. (Abbildung 8 links)
- **shader:** Der Master-Thread kopiert die die Daten auf die GPU, startet dort ihre Ausführung und überträgt die verarbeiteten Paketblock wieder auf die entsprechende Warteschleife des Worker-Threads. (Abbildung 8 mitte)
- **post-shader:** Im letzten Schritt können noch weitere Operationen durchgeführt werden, bevor der Block wieder in die einzelnen Pakete aufgeteilt wird und an die entsprechenden NICs gesendet werden. (Abbildung 8 rechts)

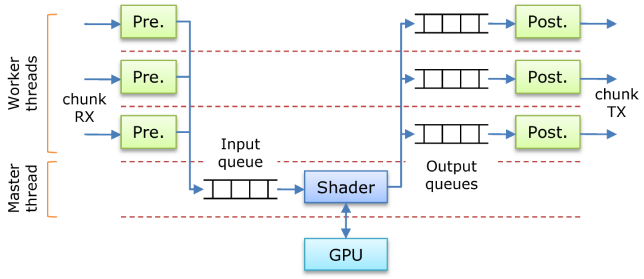

Abbildung 8: Ablauf der Verarbeitung durch die Master- bzw Worker-Threads (Quelle: [3])

Abbildung 9 verdeutlicht den Zusammenhang aller genannten Bestandteile. Für weitere Optimierungen nutzen die Autoren ein Feature namens *concurrent copy and execution* der verwendeten NVIDIA Karten, das gleichzeitiges Kopieren und Verarbeiten erlaubt. Dadurch können die Recheneinheiten auf der GPU ohne zu Pausieren neue Daten verarbeiten.

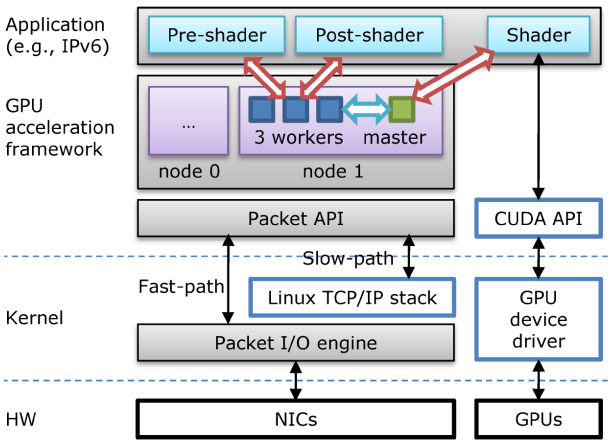

Abbildung 9: Interne Struktur von PaketShader (Quelle: [3])

4.3 Vergleich

Beide Ansätze unterscheiden sich in der Art ihrer Implementation deutlich von einander. Während die Autoren von Snap auf die Erweiterung existierender Technik setzen, wurde bei PacketShader ein eigens Konzept entwickelt. Dabei lässt sich gut beobachten wie beide Umsetzungen die gleichen grundlegenden Techniken verwenden. Der große Vorteil von Snap ist die einfache Integration in bestehende Systeme auf Basis von Click.
Vergleicht man die errichten Leistungen erhält man ein sehr ähnliches Bild. In beiden Fällen erreichen schlichte Router Konfiguration eine Leistung von etwa 40 GBit/s. Im Gegensatz zu früheren Projekten stellt das eine Verbesserung dar und beweist die Skalierbarkeit von Software-Routern.

5. FAZIT

Zusammenfassend kann man damit belegen, dass sich Grafikprozessoren auch in dem Bereich der Netzwerkverarbeitung effektiv Einsätzen lassen. Als Entwickler entsprechender Framworks gibt es einige Aspekte die es zu beachten gilt. Es müssen Lösungen gefunden werden, die eine genaue Kenntnis der verwendeten Hardware aber ebenso gleichzeitig Wissen über Softwareoptimierungen verlangen. So spielt ein gutes Speichermanagement für einen Software-Router eine erhebliche Rolle. Dennoch können die einzelnen Techniken auf verschiedenen Systemen unterschiedlich starke Auswirkungen haben. Ebenso sind die Laufzeiteinstellungen, z.B. die Blockgröße für die GPU, die Anzahl der Threads etc, stark von der verwendeten Hardware abhängig.
In den häufigsten Einsatzgebieten aktueller Software-Router, wie zum Beispiel der DSL-Router im eigenen Haus, reicht oft geringer Datendurchsatz. Aus diesem Grund beschränkt sich der Einsatz von GPU-beschleunigten Implementierungen hauptsächlich auf akademische Forschungsarbeiten. Dort beschränkt sich der Einsatz jedoch nicht nur auf Router Funktionalitäten sondern auch auf Firewalls oder Intrusion Detection Systeme [14].

In Zukunft könnte die Verwendung von Grafikprozessoren noch um einiges leichter werden. Durch den großen Erfolg GPU-basierter Softwarelösungen streben die Hardwarehersteller immer bessere Nutzbarkeit für Entwickler an. Ein

doi: 10.2313/NET-2014-08-1_05

großer Schritt für die nähere Zukunft wird die Einführung leistungsstärkerer All-In-One Chips, die bisherige dedizierte Zusatzkarten überflüssig machen könnten. AMD hat bereits mit ihrem HSA-Programm erste Grundsteine gelegt und auch Intel ist mittlerweile auf einem ähnlichen Weg [12] und integriert eine stetig schnellere Grafikeinheit in ihre Prozessoren.

6. LITERATUR

[1] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. RouteBricks: Exploiting Parallelism to Scale Software Routers. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)*, 2009.

[2] S. Han, K. Jang, K. Park, and S. Moon. Building a single-box 100 gbps software router. In *In IEEE Workshop on Local and Metropolitan Area Networks*, 2010.

[3] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: A gpu-accelerated software router. *SIGCOMM Comput. Commun. Rev.*, 40(4):195–206, Aug. 2010.

[4] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The click modular router. *ACM Trans. Comput. Syst.*, 18(3):263–297, Aug. 2000.

[5] G. Kyriazis. Heterogeneous System Architecture: A Technical Review. Website, (21.03.2014), Mar. 2012. `http://developer.amd.com/wordpress/media/2012/10/hsa10.pdf`.

[6] NVIDIA Corporation. NVIDIA's Next Generation CUDA Compute Architecture: Kepler GK110. Website, (23.03.2014), 2012. `http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf`.

[7] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. Lefohn, and T. J. Purcell. A survey of general-purpose computation on graphics hardware. In *Eurographics 2005, State of the Art Reports*, pages 21–51, Aug. 2005.

[8] L. Rizzo. netmap: A novel framework for fast packet i/o. In *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 101–112, Boston, MA, 2012. USENIX.

[9] R. Sietmann. Der fünfte Weg: Die Zukunft von IPTV, Internet-TV und die Netzneutralität. Website, (22.03.2014), 2011. `http://www.heise.de/ct/artikel/Der-fuenfte-Weg-1167899.html`.

[10] R. Smith. NVIDIA Announces CUDA 6: Unified Memory for CUDA. Website, (25.03.2014), 2013. `http://www.anandtech.com/show/7515/nvidia-announces-cuda-6-unified-memory-for-cuda`.

[11] W. Sun and R. Ricci. Fast and flexible: Parallel packet processing with gpus and click. In *ANCS*, pages 25–35. IEEE, 2013.

[12] Tom Piazza. Processor Graphics. Website, (22.05.2014), 2012. `http://www.highperformancegraphics.org/previous/www_2012/media/Hot3D/HPG2012_Hot3D_Intel.pdf`.

[13] J. Tompson and K. Schlachter. An introduction to the opencl programming model. 2012.

[14] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis. Gnort: High performance network intrusion detection using graphics processors. In *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*, RAID '08, pages 116–134, Berlin, Heidelberg, 2008. Springer-Verlag.