Seminar - Hochleistungsrechner: Aktuelle Trends und Entwicklungen
Wintersemester 2016/2017

CPU/GPU-Kombination (APU)

Marcus Rogowsky
Technische Universität München

1.2.2017

Zusammenfassung

Diese Seminararbeit untersucht den möglichen Einsatz von APUs in HPC-Systemen. Dazu wird der grobe Aufbau des Gesamtsystems einer aktuellen APU beschrieben, sowie deren für den Einsatz im HPC-Bereich wichtigen Komponenten analysiert. Darunter fallen der Aufbau der GPU Kerne und des Speichersystems mit der Umsetzung von Shared Virtual Memory, Cache-Kohärenz und On-Die Speicher mit hoher Bandbreite. Außerdem wird der aktuelle Stand bei der Programmierung von APUs dargestellt und schließlich dargelegt, warum APUs in der jetzigen Form noch nicht bereit sind um in HPC-Systemen eingesetzt zu werden.

1 Einleitung

In vielen wissenschaftlichen Bereichen hat die Einführung von Supercomputern komplett neue Arbeitsweisen und damit einhergehende Ergebnisse geliefert. Die Nachfrage nach immer noch Leistungsstärkeren Supercomputern ist aber weiterhin ungebrochen, und nach dem Erreichen von Petascale-Systemen im Jahr 2008, werden nun Exascale-Systeme anvisiert. Die Leistungssteigerung von HPC-Systemen wird damit weitergeführt und gleichzeitig nach immer neuen Konzepten gesucht, um das Ziel möglichst effizient umzusetzen.

In den letzten Jahren haben auch GPU basierte Systeme Einzug in die Gruppe der Supercomputer genommen. Dank ihrer auf parallele Berechnungen optimierten Bauweise und hohen Speicherbandbreite sind sie CPU basierten Systemen in den Bereichen Rechenleistung pro Node und Effizienz überlegen. Der schnellste Supercomputer mit aktueller GPU Generation auf Platz 8 der Top 500 Liste, führt gleichzeitig die Green 500 Liste an, die Supercomputer nach Energieeffizienz sortiert aufführt.

Systeme mit diskreten GPUs haben aber auch Nachteile, die einem allgemeinen Einsatz im Weg stehen. So ist der auf den GPUs verbaute Speicher getrennt vom Speicher für die CPU Kerne und Daten müssen vor der Verarbeitung in den Speicher auf der GPU über die PCI-Express Verbindung kopiert werden. Diese Verbindung hat mit etwa 10 bis 50 Mikrosekunden eine hohe Latenz, kann aber nicht vollständig umgangen werden, da der auf der GPU verbaute Speicher für die meisten Anwendungen nicht groß genug ist um alle Daten vorzuhalten. Dies macht Anpassungen in der Software für den Einsatz auf GPU Systemen Notwendig um die Rechenleistung auch ausnutzen zu können.

Schulte et al. [9] untersuchten zur Lösung dieser Probleme ein Konzept für den Einsatz von AMD

APUs in HPC-Systemen, im Folgenden HPC APU genannt. Nach diesem Konzept sind die CPU und GPU Kerne sowie ein Speicher mit hoher Bandbreite auf einem Interposer verbaut. Die Entwicklung solcher Systeme begann schon vor über 10 Jahren. Nach der Übernahme von ATI durch AMD im Jahr 2006, wurde an einer Zusammenführung von CPU und GPU Kernen auf einem Chip gearbeitet. Zunächst unter dem Codenamen AMD Fusion, später aber aus rechtlichen Gründen umbenannt in Accelerated Processing Unit (APU). Im Jahr 2011 wurde dann die erste APU Serie Llano veröffentlicht.

Gleichzeitig beteiligte sich AMD an der Gründung der HSA Foundation, die das Ziel hat die Entwicklung von Software für heterogene System zu vereinheitlichen und zu vereinfachen. Nach Schulte et al. [9] ist die Umsetzung der Vorgaben der HSA Foundation an die Hardware ein wichtiger Schritt um APUs in HPC-Systemen einsetzen zu können. Im Jahr 2015 wurde mit Carizzo dann auch die erste APU veröffentlicht, die die Vorgaben der HSA Foundation erfüllt.

In den folgenden Kapiteln, wird untersucht, wie heutige APUs aufgebaut sind und in wie weit diese Umsetzung geeignet ist um in HPC-Systemen eingesetzt zu werden. Dazu wird zuerst der grobe Aufbau einer APU betrachtet, um anschließend einzelne wichtige Aspekte zu vertiefen. Das Kapitel "Graphics Core Next Architektur" beschreibt den Aufbau und die Funktionsweise der GPU Kerne. Das darauf folgende Kapitel zum Speichersystem betrachtet die Umsetzung verschiedener Aspekte. Zunächst die des Shared Virtual Memory Systems, anschließend der Cachekohärenz und abschließend das Konzept von auf dem Die verbauten Speicher mit hoher Bandbreite. Abschließend wird noch auf die Vorgaben der HSA Foundation eingegangen und zu welchen Veränderungen deren Umsetzung für die Programmierung von Software für APUs führen.

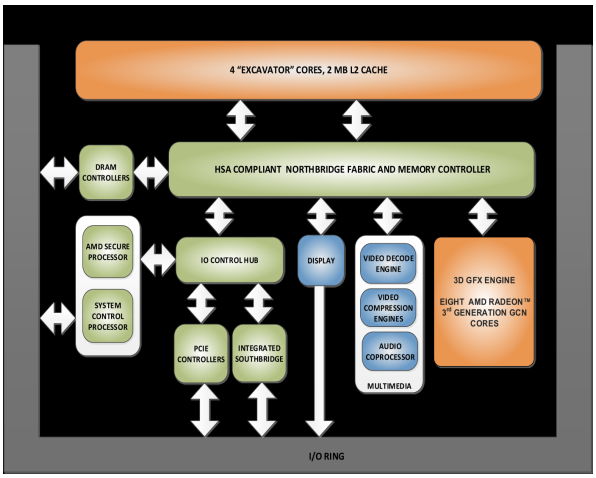

Abbildung 1: [6] Aufbau einer Carrizo APU

2 Hardware Aufbau

Das Ziel, das mit einer "Accelerated Processing Unit" (APU) verfolgt wird, ist, so viele Komponenten eines Systems wie möglich auf einem Die unterzubringen. Dabei geht es hauptsächlich um die Vereinigung der CPU mit der GPU aber auch mit weiteren Komponenten, wie Video-encoder und decoder. Ein solches System nennt man "System-on-a-Chip" (SOC), da die meisten Komponenten des Systems auf einem Chip vereinigt sind.

Die Vorteile dieser Bauweise sind eine schnellere Kommunikation zwischen den einzelnen Komponenten, da die Signale direkt auf dem Chip übertragen werden, die gemeinsame Nutzung von Resourcen, was zu Einsparungen bei der Datenübertragung beitragen kann und ein niedrigerer Energieverbrauch.

CPU Kerne arbeiten Aufgaben unabhängig von anderen CPU Kernen in einzelnen Threads ab. Sie sind deshalb besonders gut geeignet Programme mit vielen Ausführungszweigen auszuführen. Für das Ausführen von gleichen Rechenaufgaben auf mehreren Eingaben verfügen CPU Kerne über Vektoreinheiten wie AVX-512, die bis zu acht 64-

Bit Floatwerte gleichzeitig verarbeiten können. Die maximale Rechenleistung von Multikernprozessoren ist im Vergleich zu vektorbasierten Prozessoren trotzdem deutlich geringer. Selbst mit 24 Kernen der aktuellen Skylake Architektur von Intel erreicht man eine maximale Rechenleistung von etwa 1,5 TFLOPS mit 64-Bit Floatwerten bei einem Preis von etwa 7000$ und einem Stromverbrauch von 165 Watt.
Deshalb übernimmt die CPU auf einer HPC APU Aufgaben, die nicht effizient auf vektorbasierten Prozessoren ausgeführt werden können, wie die Kommunikation zwischen APUs und das Scheduling von Aufgaben.

GPU Kerne sind vektorbasierte Recheneinheiten, die die einzelnen Rechenschritte von Programmen mit mehreren Eingaben gleichzeitig verarbeiten können. AMDs GPU-Architektur hat eine Vektorbreite von 64 und Nvidias GPU-Architektur von 32 auf den GPU Kernen. Auf dem zur Zeit größten HPC GPU Chip von Nvidia sind 56 GPU Kerne aktiv, womit 1792 Rechenschritte gleichzeitig ausgeführt werden können. Damit erreicht eine GPU eine maximale Rechenleistung von etwa 5,3 TFLOPS mit 64-Bit Floatwerten bei einem Preis von etwa 9000$ und einem Stromverbrauch von 300 Watt.
Aufgrund der höheren Effizienz von GPU Kernen werden auf einer HPC APU die eigentlichen Berechnungen wenn es möglich ist darauf ausgeführt.

Der auf der APU integrierte Speichercontroller, auch Northbridge genannt, ist für die Abarbeitung der Speicherzugriffe zuständig. Bei dem HPC APU Konzept von Schulte et al. [9] kann der Speicher aus einem auf dem selben Chip verbauten Teil und einem externen angeschlossenen Teil bestehen. Der auf der HPC APU verbaute Speicher, zum Beispiel HBM2, würde zwar eine deutlich höhere Bandbreite von bis zu 1 TB/s ermöglichen, hätte aber eine kleinere maximale Größe gegenüber herkömmlichen Arbeitsspeicher.

Der IO Controller, auch Southbridge genannt, auf der APU ist für die externe Kommunikation des Chips zuständig. Bei einer HPC APU besteht diese hauptsächlich aus dem Datenaustausch mit der über PCI-Express angeschlossenen Netzwerkkarte. Aber auch das Lesen und Schreiben von Daten auf Festplatten wird von dieser Einheit ausgeführt.
Auf aktuellen APUs sind weitere Recheneinheiten verbaut, die spezielle Aufgaben besonders effizient ausführen können. Deren übliche Aufgaben sind das Encodieren oder Decodieren von Video und Audio Formaten. Diese Aufgaben werden auf einer HPC APU nicht benötigt, es ist allerdings denkbar, dass für andere Aufgaben spezielle Recheneinheiten verbaut werden könnten.

Aktuelle APU-Systeme richten sich vor allem an Endverbraucher, die ein Komplettsystem benötigen aber keine diskrete GPU verbauen wollen oder können, wie etwa in Notebooks oder Desktop-Systemen mit wenig Leistung. Dementsprechend ist die Anzahl an CPU und GPU Kernen niedrig gehalten, mit 2 bis 4 CPU Kernen, 3 bis 11 GPU Kernen und extern angeschlossenem DDR4 Speicher.
APU-Systeme die in Hochleistungsrechnern verbaut werden sollen, müssten dagegen deutlich leistungsfähiger sein. Der von Schulte et al. [9] veröffentlichte Entwurf einer HPC APU für Exascale Computing würde über 32 CPU Kerne, einer Rechenleistung von insgesamt 10 TFLOPS für die GPU Kerne sowie auf dem Chip verbauten gestapelten Speicher verfügen.

3 Graphics Core Next Architektur

Die Graphics Core Next (GCN) Architektur wurde 2011 von AMD veröffentlicht und wird seitdem für GPUs und APUs verwendet. Das Ziel bei der Entwicklung der neuen Architektur war, die Rechenleistung der GPU Kerne besser für allgemeine Berechnungen nutzbar zu machen. Dazu wurde die Programmierung vereinfacht und das Scheduling verbessert.
Seit der Einführung der GCN Architektur gab es drei nachfolgende Generationen mit denen weitere Verbesserungen eingeführt wurden. Im folgenden wird von der dritten GCN Generation ausgegangen,

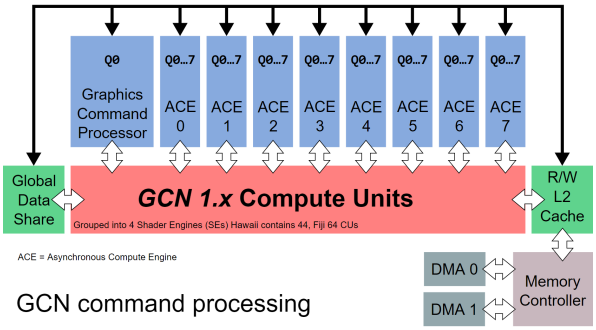

Abbildung 2: [1] Aufbau der Graphics Core Next Architektur

da diese die neuste ausführlich dokumentierte darstellt, und dabei der Schwerpunkt auf Aspekte gelegt, die für HPC Anwendungen von Relevanz sind.

3.1 Aufbau der GCN Architektur

Die GCN Architektur verfügt über acht Asynchronous Compute Engines (ACEs), die für das Scheduling von Rechenaufgaben zuständig sind. Dazu werden die Rechenaufgaben zunächst in Workgroups aufgeteilt die wiederum aus Wavefronts bestehen, die jeweils 64 Threads groß sind. Jede der ACEs kann dann pro Takt eine Wavefront an eine Compute Unit (CU) verteilen und dabei acht Queues gleichzeitig verwalten. Wavefronts werden auf den CUs grundsätzlich ungeordnet abgearbeitet und die ACEs stellen vor dem Verteilen der Aufgaben sicher, dass alle Abhängigkeiten zu anderen Rechenaufgaben eingehalten werden.
Zusätzlich zu den ACEs verfügt die GCN Architektur über zwei Direct Memory Access (DMA) Engines, mit denen Daten parallel zur Abarbeitung der Rechenaufgaben von Außen in den Speicher geschrieben oder aus dem Speicher gelesen werden können. Diese Engines sind nur auf diskreten GPUs verbaut, da bei APUs die Strategie verfolgt wird den Speicher von CPU und GPU Kernen gemeinsam zu benutzen und damit Datenaustausch zwischen den beiden Einheiten zu vermeiden.

Alle Speicherzugriffe finden über den Memory Controller statt. Auf APUs kann für die GPU und

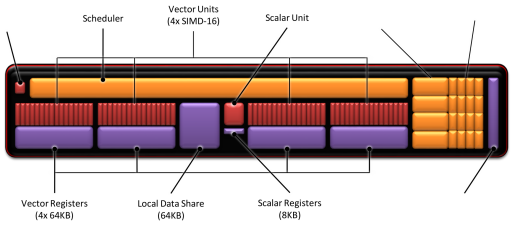

Abbildung 3: [2] Aufbau der GCN Compute Unit mit den für HPC Anwendungen relevanten Komponenten

CPU Kerne der selbe Memory Controller verwendet werden. Um Konflikte bei der parallelen Abarbeitung von Aufgaben auf den CUs zu vermeiden, verfügt jede Aufgabe über ihren eigenen virtuellen Speicherbereich.
Zwischen CUs und Memory Controller ist ein L2 Cache geschaltet, der für alle CUs kohärent ist. Zur Gewährleistung der Cache-Kohärenz mit CPU Kernen ist das L2 Cache-Interface so konzipiert, dass der Inhalt direkt mit dem Cachesystem von x86 CPUs ausgetauscht werden kann. Die Cache-Lines sind 64 Byte groß und das virtuelle Speichersystem unterstützt 4 KiB große Pages. Der Cache ist aufgeteilt in Partitionen die jeweils 64 KiB oder 128 KiB groß sind und die Gesamtgröße des Caches ist durch die Anzahl der verwendeten Partitionen festgelegt.

Neben dem L2 Cache gibt es noch einen weiteren globalen Speicher, der von allen CUs gelesen und geschrieben werden kann. Dieser als Global Data Share (GDS) bezeichnete Speicher ist 64 KiB groß und wird neben der Verwendung für architekturinternen Datenaustausch zwischen den CUs auch zur Bearbeitung von atomaren Operationen verwendet.

3.2 Aufbau der GCN Compute Unit

Die Compute Units der GCN Architektur bearbeiten die ihnen zugewiesenen Wavefronts. Dazu verfügt jede CU über eigene Ressourcen, die nicht mit anderen CUs geteilt werden.

Jede CU verfügt über vier SIMD Einheiten die jeweils 16 Operationen gleichzeitig bearbeiten können. Mit der fünften GCN Generation wird es zudem möglich sein Operationen auf kleineren Datentypen wie 16-Bit Float- und Integerwerten oder 8-Bit Integerwerten zusammenzufassen und damit die Anzahl der Operationen pro Takt zu verdoppeln oder zu vervierfachen.
Den SIMD Einheiten werden immer ganze Wavefronts zugeteilt, was bedeutet, dass für die Bearbeitung einer Operation für alle Elemente in der Wavefront vier Takte oder ein Vielfaches von vier Takten notwendig sind. Um Latenzen bei Speicherzugriffen auszugleichen, kann jede SIMD Einheit bis zu 10 Wavefronts gleichzeitig bearbeiten. Wird in einer Wavefront ein Speicherzugriff ausgeführt, kann die SIMD Einheit in der Zwischenzeit eine Operation von einer der restlichen Wavefronts ausführen.

Jede SIMD Einheit verfügt über 64 KiB an Registern. Bei der Verwendung von 32-Bit Registern stehen jeder SIMD Einheit damit 16 tausend Register zur Verfügung. Um die Wavefronts optimal auf den SIMD Einheiten abarbeiten zu können sollten immer mindestens sechs Wavefronts gleichzeitig bearbeitet werden. Das bedeutet, dass pro Thread nicht mehr als 42 Register verwendet werden sollten.

Zusätzlich zu den SIMD Einheiten hat jede CU eine Scalar Unit. Diese Einheit kann pro Takt eine Instruktion ausführen und berechnet eine 64-Bit breite Maske, mit der festgelegt wird, welche der 64 Threads in einer Wavefront für die nächste Operation aktiv sind. Dadurch werden Verzweigungen im Code realisiert, indem immer nur die Threads aktiviert sind, die den aktuellen Codepfad ausführen. Da so aber immer nur ein Codepfad pro Wavefront ausgeführt werden kann, führen Verzweigungen zu einer geringeren Rechenleistung und sollten deshalb soweit es möglich ist vermieden werden.

Jede CU verfügt außerdem über einen eigenen L1 Cache, der als Local Data Share (LDS) bezeichnet wird und 64 KiB groß ist. Dieser Speicher kann entweder komplett als L1 Cache verwendet oder ein Teil davon von der Anwendung selbst verwaltet werden. Der LDS ist für alle Threads in einer Workgroup kohärent, womit er das effiziente austauschen von Daten zwischen Threads innerhalb einer Workgroup möglich macht.

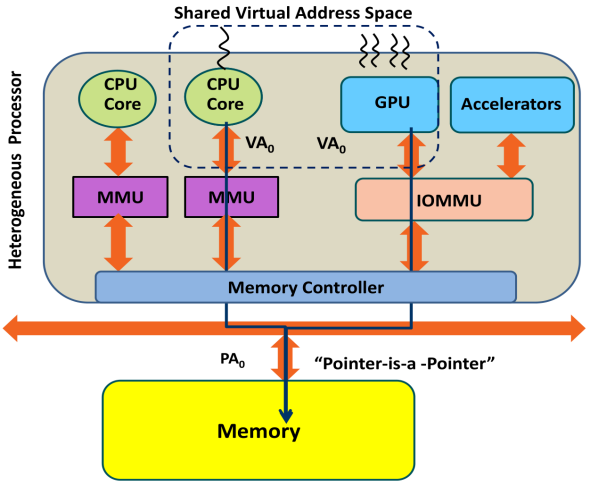

Abbildung 4: [10] Aufbau eines Shared Virtual Memory Systems für CPU und GPU Kerne

4 Speichersystem

Der Hauptnachteil an Systemen mit diskreten GPUs ist der geteilte Speicher und die damit einhergehenden Probleme bei der Programmierung und Ausführung von Programmen. Das Ziel des Speichersystems von APUs ist daher den selben Speicher für CPU und GPU Kerne zu verwenden aber gleichzeitig die hohe Rechenleistung der GPU Kerne beizubehalten.

4.1 Shared Virtual Memory

Auf einer APU verwenden CPU und GPU Kerne denselben Speicher. Allerdings reicht dies alleine noch nicht aus, damit die selben Daten im Speicher von beiden Kernen verwendet werden können. Speicher von der CPU ist in Pages mit virtuellen Adressen organisiert, denen physikalische Adressen zugewiesen werden können. Sollen CPU und GPU Kerne auf dieselben Daten zugreifen können, muss

das Pagingsystem auf die GPU Kerne erweitert werden.

Dazu wird auf APUs eine IO Memory Management Unit (IOMMU) verbaut, die die Adressübersetzung für die GPU Kerne übernimmt. Die IOMMU ist in der Northbridge der APU integriert und kann wie die MMUs der CPU Kerne die Pagetablestruktur für x86-64 Prozessoren verwenden. Zum Verwalten der IOMMU ist außerdem ein OS-Treiber notwendig, der Aufgaben auf den CPU Kernen ausführen kann.

Zusätzlich wird die GPU um einen Translation Lookaside Buffer (TLB) erweitert, der die physikalischen Adressen der zuletzt verwendeten Pages speichert, und der direkt von den CUs verwendet werden kann.

Greift eine CU auf eine Page zu, die nicht im TLB gespeichert ist, wird ein TLB-Miss ausgelöst. Die GPU sendet dann eine Address Translation Protokoll (ATS) Anfrage an die IOMMU, die über einen eigenen TLB verfügt. Ist die Adresse für die Page auch nicht im TLB der IOMMU gespeichert muss diese die Hardware-Pagetables auslesen. Wird die Adresse darin gefunden, sendet die IOMMU die Adresse mit einer ATS Antwort an die GPU zurück. Das ATS Protokoll erlaubt es bis zu acht aufeinanderfolgende virtuelle Pages mit einer Anfrage anzufordern, was in aktuellen APUs standardmäßig aktiviert ist.

Findet die IOMMU die gesuchte Adresse jedoch nicht im Hardware-Pagetable, sendet sie eine ATS Page-fault Antwort an die GPU zurück. Diese unterbricht daraufhin die Ausführung der Wavefront die diese Page benötigt und sendet eine Peripheral Page Request (PPR) Anfrage an die IOMMU, die diese Anfragen in einer Queue sammelt und dann für mehrere Anfragen gebündelt einen CPU Interrupt auslöst. Der IOMMU Treiber behandelt dann auf der CPU die PPR Anfragen, indem er die Pages wieder in den Speicher lädt. Anschließend benachrichtigt der Treiber dann die IOMMU darüber, dass die PPR Anfrage abgeschlossen wurde, die wiederum die GPU benachrichtigt. Die Wavefront wird dann wieder von einer ACE gescheduled was wieder zu einer ATS Anfrage für die Adresse der Page an die IOMMU führt, die jetzt die Adresse im Hardware-Pagetable finden kann.

Ändert sich das Speichermapping von Pages, müssen auch die Einträge in den TLBs aktualisiert werden. Dazu überwacht der IOMMU Treiber Änderungen in den Pagetables und schickt wenn eine Änderung stattgefunden hat eine Nachricht an die IOMMU, dass die alte Adresse ungültig ist. Diese entfernt dann die Adresse aus dem eigenen TLB und schickt eine Nachricht an die GPU, die dann die Adresse aus ihrem TLB entfernt. Beim nächsten Speicherzugriff der GPU wird dann die neue Adresse ermittelt.

Vesely et al. [10] haben die Performance des Pagingsystems aktueller APUs untersucht und kommen zu dem Ergebnis, dass schon mit diesen in bestimmten Testszenarien die Performance durch das Pagingsystem beschränkt wird. Vor allem bei Anwendungen mit ungeordneten Speicherzugriffen führte der Overhead des Pagingsystems zu einer Verringerung der Gesamtleistung auf ein Viertel der möglichen Rechenleistung. Außerdem ergaben ihre Messungen, dass es 25 mal länger dauert einen TLB-miss der GPU zu bearbeiten als auf der CPU, auf Grund der vielen Nachrichten, die ausgetauscht werden müssen. Die Behandlung von Page-faults auf der GPU kann sogar bis zu 82 mal langsamer sein als auf der CPU.
Auf HPC APUs die über deutlich mehr GPU Kerne und eine höhere Speicherbandbreite verfügen, wird dieses System dann vollends zum Flaschenhals. Bevor eine solche APU umgesetzt werden kann muss das Pagingsystem für APUs überarbeitet und damit deutlich verbessert werden.

4.2 Cache-Kohärenz

Die GPU Kerne und CPU Kerne auf einer APU haben weiterhin eigene L2 Caches. Verändert zum Beispiel die CPU Daten die in ihrem Cache bleiben und lädt die GPU danach dieselben Daten, sind die Änderungen nur sichtbar, wenn beide Caches kohärent gehalten werden. Um diese Kohärenz auf aktuellen APUs umzusetzen kommt ein Directory-

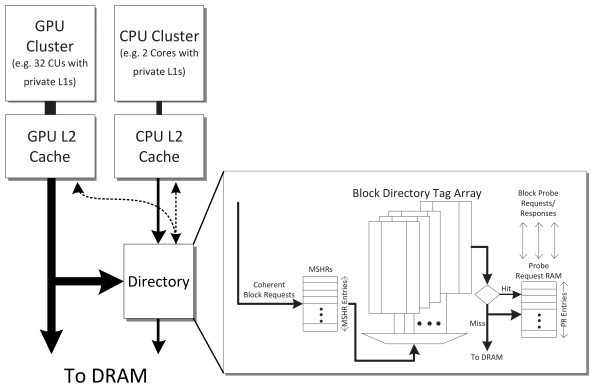

Abbildung 5: [8] Aufbau eines kohärenten Cache Systems für CPU und GPU Kerne

Based Cache Coherence System zum Einsatz.

Die GPU Kerne und die CPU Kerne haben weiterhin eigene L2 Caches die für die jeweiligen Kerne kohärent gehalten werden. Zusätzlich wird ein globales Speicherblockverzeichnis verbaut, dass für jeden Block mit zwei Bits speichert, ob er in einem der beiden L2 Caches gespeichert ist. Alle Speicherzugriffe der CPU werden über dieses Verzeichnis durchgeführt, bei der GPU dagegen müssen nur die Speicherzugriffe über das Verzeichnis durchgeführt werden, die Kohärent mit der CPU sein sollen.
Die GPU kann über zwei Busse Daten übertragen. Der so genannte "Garlic" Bus ist direkt mit dem Speicher verbunden und kann für GPU interne Speicherzugriffe genutzt werden. Der "Onion" Bus dagegen ist mit dem Speicherblockverzeichnis verbunden.

Tritt auf der GPU oder der CPU ein L2 Cache-Miss auf, wird eine Anfrage an das Verzeichnis geschickt, das für alle ankommenden Anfragen zunächst ein "Miss Status Handle Register" (MSHR) erstellt und in einer Tabelle speichert. Für die Register wird dann nacheinander überprüft, ob der Tag für den Speicherblock im Speicherblockverzeichnis gespeichert ist.

Ist der Tag in dem Verzeichnis gespeichert, bedeutet das, dass er bereits in dem anderen Cache vorhanden ist. Für Schreibzugriffe muss die andere Kopie ungültig gemacht werden, und für Lesezugriffe muss die aktuelle Kopie des Speicherblocks aus dem anderen Cache gelesen werden. Dazu werden Anfragen für den jeweiligen Cache erstellt und bis zur Bearbeitung im "Probe-Request RAM" (PPR) gespeichert. Wurde der Speicherblock von dem Cache geschickt oder die Bestätigung, dass der Speicherblock gelöscht wurde, wird die Anfrage aus dem PPR entfernt.
Für Lesezugriffe wird der Speicherblock an den anderen Cache geschickt und das Bit für den Cache im Verzeichnis für den Speicherblock gesetzt. Für Schreibzugriffe wird der Speicherblock aus dem Verzeichnis entfernt.

Ist der Tag nicht in dem Verzeichnis gespeichert, wird der Speicherblock aus dem Speicher geladen und das Bit im Verzeichnis für den Cache gesetzt, der jetzt eine Kopie des Speicherblocks enthält.

Power et al. [8] stellten bei ihrer Untersuchung dieses Systems auf aktuellen APUs allerdings fest, dass es nicht für die bei GPUs üblichen hohen Cache-Miss-Raten ausgelegt ist. Eine deutliche Vergrößerung der MSHR-Tabelle würde in ihren Testanwendungen zu einer Leistungssteigerung von bis zu 300% führen. Außerdem müsste das Verzeichnis bis zu vier Anfragen pro Takt unterstützen können anstatt nur einer wie bisher. Sie schlagen deshalb vor für die GPU und die CPU jeweils ein eigenes Speicherblockverzeichnis zu verbauen, das Speicheroperationen, die nicht kohärent gehalten werden müssen automatisch über den schnelleren Bus mit direkter Verbindung zum Speicher leitet und damit für diese das gemeinsame Speicherblockverzeichnis umgeht.

4.3 On-Die Stacked Memory

Der bisher betrachtete Speicher wird, wie es für CPU Speicher üblich ist, an die APU angeschlossen und verfügt über eine Bandbreite von bis zu ca. 20 GB/s. Während mit diesem Speicher eine ausreichende Größe für eine HPC APU erreicht werden kann, ist die Bandbreite für die Datenübertragung viel zu gering. Auch auf diskreten GPUs verbau-

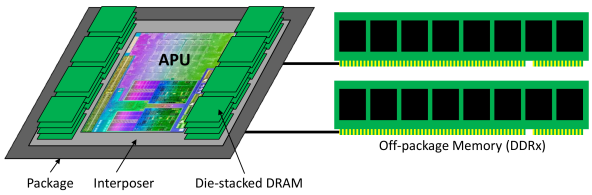

Abbildung 6: [7] Konzept einer APU mit gestapeltem Speicher auf einem Interposer und zusätzlich angebundenem externen Speicher

ter GDDR5 Speicher ist mit bis zu ca. 400 GB/s noch zu langsam, und mit der heutigen maximalen Größe von 32 GB zu klein. Gestapelter Speicher der auf dem selben Die wie die APU verbaut wird, kann dagegen in der aktuellen Version die geforderte Bandbreite von 2 TB/s liefern, ist aber auch auf eine Größe von 32 GB beschränkt.

Da nicht abzusehen ist, dass in naher Zukunft eine neue Speichertechnologie die geforderte Größe und Bandbreite gleichzeitig unterstützt, müssten für eine HPC APU zwei verschiedene Methoden kombiniert werden um deren Vorteile zu nutzen. Um die geforderte Bandbreite zu erreichen wird auf dem Die der APU gestapelter Speicher verbaut und über einen Interposer mit der APU verbunden, während gleichzeitig gewöhnlicher DDRx Speicher angeschlossen wird.

Für ein solches hybrides Speichersystem gibt es zwei grundsätzlich unterschiedliche Wege bei der Umsetzung. Eine Möglichkeit ist es Anwendungen expliziten Zugriff auf beide Speicher zu geben, was bedeutet, dass die Anwendung selbst entscheidet, welche Daten im schnellen Speicher gehalten werden. Der Vorteil ist, dass die Anwendung den Ablauf bei der Verarbeitung der Daten kennt und so eine intelligente Entscheidung fällen kann. Der Nachteil ist aber, dass die Anwendung den Speicher selbst verwalten muss, womit ihr Entwicklungsaufwand steigt, da sie explizit an ein solches Speichersystem angepasst werden muss.

Die andere Möglichkeit ist es den schnellen Speicher als Cache zu verwenden, der von der Hardware verwaltet wird. Damit können Programme, ohne dass Anpassungen notwendig sind, direkt auf der APU ausgeführt werden und von dem schnellen Speicher profitieren. Ein solches Cache-System müsste jedoch Aufgrund der Größe des Caches und der damit auftretenden Schwierigkeiten für Tag basierte Caches anders aufgebaut sein als bisherige Systeme.

Ein möglicher Ansatz für ein neues Speichersystem wurde von Meswani et al. [7] untersucht. Dieses System basiert auf Pages und beide Speicher bilden zusammen einen gemeinsamen Speicherbereich, bei dem die Pages zwischen den beiden Speichern wechseln können. Dazu werden die Zugriffe auf die Pages aufgezeichnet und das Betriebssystem transferiert alle 100 Millisekunden alle Pages mit Zugriffen (First-touch policy) oder nur die Pages mit den meisten Zugriffen (Hot-page policy) in den schnellen Speicher. Messungen mit Testanwendungen, die bei ausschließlicher Verwendung von gestapelten Speicher um durchschnittlich 20% schneller liefen, zeigten bei Verwendung dieses Systems einen Speedup von durchschnittlich 15%. Damit konnte der Vorteil den der schnellere Speicher bringt zu 75% ausgenutzt werden, ohne dass Anpassungen in den Anwendungen notwendig waren.

Die effektive Ausnutzung der hohen Bandbreite von gestapeltem Speicher für HPC Anwendungen stellt ein aktuelles Forschungsfeld dar, das für die Umsetzung von Exascale-Systemen eine entscheidende Rolle spielt.

5 Programmierung

Um die Vorteile der Bauweise von APUs in Anwendungen ausnutzen zu können, muss die Programmierung der Software dafür angepasst sein. AMD gründete dazu mit weiteren Firmen, unter anderem ARM und Samsung, im Juni 2012 die Heterogeneous System Architecture (HSA) Foundation, die das Ziel verfolgt einen herstellerübergreifenden und offenen Standard für die Programmierung von heterogenen Systemen zu entwickeln. Im März 2015 wurde die erste Ver-

sion der HSA Spezifikation veröffentlicht, gefolgt von der ersten HSA kompatiblen Hardware im Oktober 2015 mit der Carizzo APU. Mittlerweile unterstützen über 40 Firmen die Initiative, an 17 Universitäten wurden HSA Academic Centers of Excellence gebildet und im Sommer 2016 eine HSA Konferenz in Peking abgehalten.

Um die Softwareentwicklung für verschieden heterogene Systeme zu vereinheitlichen und zu vereinfachen, macht die HSA Spezifikation bestimmte Vorgaben darüber, was von der Hardware unterstützt werden soll, überlässt den Herstellern aber die Umsetzung. Sie besteht aus mehreren Teilen, die jeweils verschiedene Bereiche, die nötig sind um Software auf heterogenen System auszuführen, abdecken. Die HSA Platform System Architecture Specification legt die Vorgaben an die Hardware fest, die erfüllt werden müssen. Darunter sind unter anderem die Unterstützung von Shared Virtual Memory, Cache-Kohärenz, Signal- und Synchronisationsoperationen, atomare Operationen und User Mode Queueing. Das HSA Programmer's Reference Manuel spezifiziert die einheitliche Zwischensprache HSAIL, in die C++, OpenCL, OpenMP und Java Quellcode übersetzt werden, um anschließend in die hardwarespezifische Instruktionssprache übersetzt zu werden. Das HSA Runtime Programmer's Reference Manual definiert die HSA Runtime API, die eine Schnittstelle zum Ausführen von Code und zur Verwaltung von Ressourcen auf HSA Systemen bereitstellt.

Der Unterschied zwischen der bisherigen Programmierung für diskrete GPUs und der Vereinfachung mit HSA für APUs wird deutlich, wenn man die Schritte vergleicht, die Notwendig sind um Code auf der GPU auszuführen. In Abb. 7 sind die einzelnen Schritte Aufgeführt und farblich markiert. Zunächst muss die Anwendung die Daten, die verarbeitet werden sollen auf die GPU kopieren. Das Betriebssystem und der GPU-Treiber übernehmen dann den eigentlichen Kopiervorgang. Sind die Daten auf die GPU kopiert, kann die Anwendung eine Aufgabe an die Warteschlange im Treiber anhängen. Der Treiber arbeitet dann die Warteschlange ab und schließlich wird die Aufgabe auf der GPU ausgeführt. Die Anwendung wartet dann, bis sie vom Betriebssystem mitgeteilt bekommt, dass die Aufgabe ausgeführt wurde und muss sich die Ergebnisse aus dem Speicher auf der GPU holen. Das Betriebssystem und der GPU-Treiber kopieren dazu die Daten wieder in den CPU-Speicher und die Anwendung kann weiter mit den Ergebnissen arbeiten. Der Aufwand um Aufgaben auf der GPU auszuführen und die Ergebnisse anschließend auf der CPU weiter zu verarbeiten ist also relativ hoch, im Vergleich zum Beispiel zur Verwendung von OpenMP um Code parallel auf CPU Kernen auszuführen.

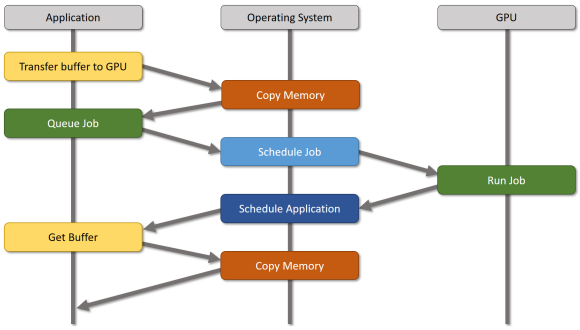

Abbildung 7: Herkömmliche Programmierung für die GPU. Schritte die mit HSA noch notwendig sind, sind grün markiert.

Mit der Umsetzung der HSA Spezifikation verbessert sich die Situation jedoch deutlich. Durch Verwendung von Shared Virtual Memory entfallen die beiden Kopiervorgänge im Betriebssystem und GPU-Treiber (Orange markiert). Mit einem cache-kohärenten Speichersystem entfallen außerdem die beiden Cache-Flushes, die Notwendig waren, damit die aktuellen Daten auf der GPU oder der CPU verwendet werden (Gelb markiert). Wird von der APU Signaling unterstützt, kann die GPU der Anwendung auf der CPU direkt mitteilen, dass die Aufgabe auf der GPU abgearbeitet wurde. Dadurch entfällt die Indirektion über das Betriebssystem und den GPU-Treiber (Dunkelblau mar-

kiert). Schließlich definiert die HSA Spezifikation noch User Mode Queueing, womit eine Anwendung Aufgaben auf der GPU direkt zum Ausführen in die Warteschlange auf der GPU einordnen kann. Damit entfällt auch hier die Indirektion über den GPU-Treiber (Hellblau markiert).
Übrig bleiben die beiden Schritte, die Aufgabe auf der GPU in die Warteschlange einzuordnen und diese auf der GPU auszuführen (Grün markiert). Damit lässt sich Code auf der GPU nun vergleichbar einfach ausführen, wie mit OpenMP auf der CPU.

Im April 2016 veröffentlichte AMD die ROCm Platform auf GitHub, die die Möglichkeit bietet mit offener Software Programme für APUs zu entwickeln. Darin enthalten sind eine Runtime für Linux, Compiler für C und C++ sowie ein Compiler für die von Nvidia entwickelte Programmiersprache CUDA. Die Entwicklung von Softwareentwicklungswerkzeugen für APUs steht damit noch am Anfang und es muss erst noch abgewartet werden, ob sich diese in den nächsten Jahren durchsetzen.

6 Schluss

Obwohl die Entwicklung von APUs schon vor über 10 Jahren begonnen hat, steht deren Einsatz und vor allem die Anpassung von Softwareentwicklungswerkzeugen noch ganz am Anfang. Um das Einsatzgebiet von sparsamen Desktop- und Notebooksystemen auf den HPC-Bereich zu erweitern, sind noch einige Verbesserungen an der Hardware notwendig. Das Speichersystem ist zudem bisher nur für niedrige Rechenleistungen ausgelegt, und würde in der jetzigen Form HPC-Anwendungen sehr stark ausbremsen.

Eine direkte Umsetzung einer für den HPC-Markt zugeschnittenen APU erscheint damit derzeit als eher unwahrscheinlich. Die Entwicklungskosten und das Risiko für die Entwicklung völlig neuer Hardwarekomponenten und einer nicht abzuschätzenden Akzeptanz im Markt sind außerdem zu hoch. Ein möglicher Weg bietet sich allerdings bei der Entwicklung von APUs für Spielkonsolen an, da diese ähnliche Anforderungen haben wie Desktopsysteme mit eingebauten Grafikkarten und die Anpassung der Software an das Hardwaresystem dort die übliche Praxis darstellt.

Bis APU Systeme also letztendlich in Supercomputern eingesetzt werden, werden noch einige Jahre vergehen, in denen leistungsfähige APUs zunächst in Spielkonsolen und Desktopsystemen Verbreitung finden werden. Auf lange Sicht bleibt das Konzept aber attraktiv, wenn bei der Umsetzung darauf geachtet wird, dass die Softwareentwicklung einfach ist und das Hardwaresystem auf hohe Rechenleistungen angepasst ist.

Literatur

[1] GCN command processing. `https://upload.wikimedia.org/wikipedia/commons/9/99/GCN_command_processing.svg`. Accessed: 2017-01-10.

[2] GCN Compute Unit. `http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2014/05/gcn_compute_unit-1140x511.png`. Accessed: 2017-01-10.

[3] Top500 Supercomuter Sites. `https://www.top500.org`. Accessed: 2017-01-29.

[4] J. Glossner. Multicore digital signal processor for heterogeneous systems era. IEEE SigPort, 2015.

[5] Advanced Micro Devices Inc. White Paper | AMD GRAPHICS CORES NEXT (GCN) ARCHITECTURE. 2012.

[6] Guhan Krishnan, Dan Bouvier, and Samuel Naffziger. Energy-efficient graphics and multimedia in 28-nm carrizo accelerated processing unit. IEEE Micro, 36(2):22–33, 2016.

[7] Mitesh R. Meswani, Sergey Blagodurov, David Roberts, John Slice, Mike Ignatowski, and Gabriel H. Loh. Heterogeneous memory architectures: A hw/sw approach for mixing die-stacked and off-package memories. In HPCA, pages 126–136. IEEE Computer Society, 2015.

[8] Jason Power, Arkaprava Basu, Junli Gu, Sooraj Puthoor, Bradford M. Beckmann, Mark D. Hill, Steven K. Reinhardt, and David A. Wood. Heterogeneous system coherence for integrated cpu-gpu systems. In Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO-46, pages 457–467, New York, NY, USA, 2013. ACM.

[9] Michael J. Schulte, Mike Ignatowski, Gabriel H. Loh, Bradford M. Beckmann, William C. Brantley, Sudhanva Gurumurthi, Nuwan Jayasena, Indrani Paul, Steven K. Reinhardt, and Gregory Rodgers. Achieving exascale capabilities through heterogeneous computing. IEEE Micro, 35(4):26–36, 2015.

[10] Ján Veselý, Arkaprava Basu, Mark Oskin, Gabriel H. Loh, and Abhishek Bhattacharjee. Observations and opportunities in architecting shared virtual memory for heterogeneous systems. In 2016 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS 2016, Uppsala, Sweden, April 17-19, 2016, pages 161–171, 2016.