Seminar: Hochleistungsrechner: Aktuelle Trends und Entwicklungen
Wintersemester 2015/2016

Verbindung von CPUs und GPUs: AMD APUs

Johannes Klicpera
Technische Universität München

3.2.2016

Zusammenfassung

Im HPC-Bereich ist der Einsatz von GPUs und anderen externen Beschleunigern mittlerweile weit verbreitet. Die immer kleiner werdende Strukturgröße moderner Chips ermöglicht jedoch die direkte Integration der GPU auf dem Chip der CPU. Sowohl AMD also auch Intel haben diese im Consumerbereich bereits teilweise durchgeführt. In dieser Arbeit werden Möglichkeiten und Vorteile dieser Kombination für den High Performance Computing (HPC)-Bereich vorgestellt und erläutert. Diese sind unter anderem die einfache Arbeitsverteilung zwischen CPU und GPU, uniformer Speicher und einfachere Programmierung. Zudem wird mittels mehrerer veröffentlichter Analysen der Vorteil von APUs im Vergleich zu diskreten GPUs quantifiziert.

1 Einleitung

Wie durch Moore's Law vorhergesagt, wächst die Anzahl an Transistoren auf einem Chip seit Jahrzehnten exponentiell. Allerdings sinkt der Leistungsverbrauch moderner Transistoren wegen Leckströmen kaum weiter. Die Wärmeerzeugung pro Chipfläche bleibt deshalb nicht mehr konstant, wie einst von Robert H. Dennard vorhergesagt, sondern steigt exponentiell an. Dieses Zusammenbrechen von Dennard'scher Skalierung führt dazu, dass nicht mehr alle Transistoren auf einem Chip gleichzeitig aktiviert sein können, da die produzierte Wärme nicht abgeführt werden kann [11]. Der Prozentsatz an deaktivierten Transistoren auf einem Chip, dem sog. „Dark Silicon", steigt daher rasant an. Der Grund hierfür ist, dass Leckströme nur verhindert werden können, wenn Transistoren komplett von der Stromversorgung getrennt werden. Die einzige Möglichkeit, die höhere Anzahl an Transistoren dennoch zu nutzen, ist die Verwendung anwendungsspezifischer Schaltungen. Durch die sehr unterschiedlichen Bedürfnisse verschiedener Anwendungsfälle führt dies auch zu großen Steigerungen in Rechenleistung und Effizienz [11].

Die Integration unterschiedlicher, spezialisierter Recheneinheiten auf einem Chip bezeichnet man als heterogene Rechnerarchitektur („Heterogeneous System Architecture", HSA). Dies kann verschiedene Arten von Spezialisierungen beschreiben, vom Verwenden verschieden gearteter CPU-Kerne (big.LITTLE) über das effiziente Einbinden von GPGPUs (general-purpose GPUs) bis hin zu Field-Programmable Gate Arrays (FPGAs), Coarse-Grained Reconfigurable Arrays (CGRAs) und Application-Specific Integrated Circuits (ASICs) wie beispielsweise digitale Signalprozessoren (DSPs) [11].

Jedoch spielt das Verwenden unterschiedlicher CPU-Kerne im HPC-Bereich keine sonderlich große Rolle und das Einbinden von FPGAs, CGRAs und ASICs beginnt erst langsam. Das Einbinden von

GPGPUs ist hingegen aktuell voll im Gange. AMD hat bereits 2010 die Ära der Accelerated Processing Unit (APU) engeläutet, welche eine Verbindung aus CPU und GPU darstellt [1, 6]. AMD hatte aber wegen ihrer aktuell schwachen CPU-Mikroarchitektur in den letzten Jahren große Probleme im Serverbereich, weshalb sich dieses Konzept bisher noch nicht durchsetzen konnte. Diese Verbindung hat jedoch großes Potential, da die sequentielle Rechenkraft einer CPU mit komplexen Rechenkernen und out-of-order Berechnung mit der parallelen Datenverarbeitung einer GPU kombiniert werden kann. Es gibt zwar bereits einige Ansätze, um die Rechenleistung einer externen GPU zu nutzen (Nvidia CUDA, OpenCL, C++ AMP), jedoch limitiert die komplexe Kommunikation und Verwaltung des Speichers die effiziente Kombination von CPUs und GPUs [13]. Die Integration beider Recheneinheiten auf einem Chip kann diesen Flaschenhals beseitigen. Neue Programmieransätze können zudem viele der Entwicklungsschwierigkeiten von GPU-Berechnungen auf heterogenen Rechnerarchitekturen lösen [13].

In Abschnitt 2 dieser Arbeit werden die Vorteile von CPUs und GPUs gegenübergestellt und Vorteile der Integration beider Einheiten auf einer APU erläutert. Anschließend wird in Abschnitt 3 beispielhaft die erste APU von AMD (Llano) beschrieben und deren Vorteile dargelegt. Im darauf folgenden Abschnitt 4 werden Einzelheiten und Verbesserungen der Speicherverwaltung in heterogenen Systemarchitekturen, insbesondere in Hinblick auf uniformen Hauptspeicher, beschrieben. Abschnitt 5 behandelt die Programmierung heterogener Systemarchitekturen und Abschnitt 6 quantifiziert schließlich die Vorteile, welche die Kombination von CPU und GPU in verschiedenen Anwendungen bringt. Abschnitt 7 fasst diese Arbeit zusammen und gibt einen Ausblick auf künftige Entwicklungen in diesem Bereich.

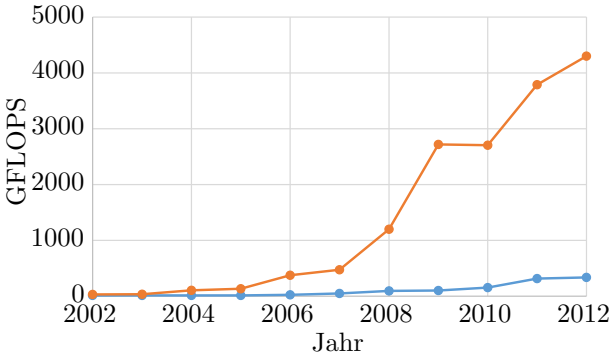


Abbildung 1: GFLOPS (floating point operations per second) von CPUs (blau) und GPUs (orange) von 2002 bis 2012. Die Leistung von GPUs wächst deutlich schneller als die von CPUs. Daten aus [14].

2 Verbindung von CPUs und GPUs

Die Rechenleistung moderner CPUs und GPUs ist in den letzten Jahren immer weiter auseinander gedriftet. Abbildung 1 zeigt diesen Trend, an dem man sehen kann, dass der Einsatz der GPU die Rechenleistung signifikant steigern kann.
Jedoch wurde in den letzten Jahren ebenfalls gezeigt, dass die Vorteile von GPUs in der Gleitkomma-Rechenleistung sehr anwendungsspezifisch ist. Lee et al. haben in [10] gezeigt, dass dieser Vorsprung über verschiedene Anwendungen hinweg im Durchschnitt nur zu einer Leistungssteigerung von 2.5X führt, wenn man für beide Prozessortypen sorgfältig optimiert. Beide Architekturen haben sehr unterschiedliche Eigenschaften, die in verschiedenen Anwendung unterschiedlich zum Tragen kommen. Die Unterschiede spiegeln sich vor allem in der Prozessierung und dem Speicherzugriff wieder.

Prozessierung: CPUs haben eine hohe sequentielle Performanz durch superskalare out-of-order Prozessierung. Für parallele Datenverarbeitung setzen sie auf SIMD-Vektoreinheiten (Single Instruction, Multiple Data). GPUs hingegen verwenden sehr einfache Recheneinheiten und eine niedrigere Taktrate. Dadurch nimmt jede Recheneinheit sehr we-

nig Platz ein und es können viele davon auf einem Chip untergebracht werden. Dies ermöglicht die Verarbeitung von sehr vielen Daten gleichzeitig mittels SIMT (Single Instruction, Multiple Threads). Während bei SIMD ein einzelner Thread eine Operation auf mehrere Daten gleichzeitig anwendet, wenden bei SIMT viele leichtgewichtige, parallele Threads in unterschiedlichen Recheneinheiten dieselbe Operation auf verschiedene Daten an [10].

Speicherzugriff: CPUs verwenden große Caches, um die Latenzen von Speicheroperationen zu verbergen. Zusätzlich werden oft Hardware-Prefetcher verwendet, um Speicherzugriffe im Voraus auszuführen und so Verzögerungen noch weiter zu reduzieren. Dies liegt daran, dass bei CPUs geringe Speicherlatenzen sehr wichtig sind, da diese auf Anwendungen mit vielen Verzweigungen (Branching) spezialisiert sind. Deshalb wird zudem Hauptspeicher mit möglichst geringen Zugriffslatenzen verwendet. Um weiters eine gute Flexibilität zu gewährleisten, werden für den Hauptspeicher austauschbare Speicherriegel verwendet. Deshalb ist die Bandbreite des Hauptspeichers (z.B. DDR4) jedoch deutlich geringer als die des Grafikspeichers (z.B. GDDR5). GPUs können diese höhere Bandbreite deutlich effektiver ausnutzen, da diese typischer Weise dieselbe Operation auf viele zusammenhängende Daten anwenden. Weiters wird eine eigene Verwaltungseinheit mit Warteschlange eingesetzt, die Speicherzugriffe verwaltet und optimiert. Die höhere Speicherlatenz wird versteckt, indem GPUs eine Vielzahl von Threads pro Recheneinheit ausführen, um während Speicherzugriffen andere Operationen auszuführen [10].

Diese Unterschiede führen dazu, dass CPUs und GPUs für verschiedene Anwendungsszenarios unterschiedlich gut geeignet sind. Dies ist an dem je nach Anwendung sehr unterschiedlichen Speedup bei Verwendung einer GPGPU erkennbar, wie in Abbildung 2 dargestellt. In einigen Anwendungen können sie sich sehr gut ergänzen und zusammenarbeiten. In den meisten anderen Anwendungen können sie sich zudem gegenseitig unterstützen, beispielsweise durch das Prefetching von Daten [17]. Dies setzt voraus, dass die Kommunikation zwischen CPU und GPU sehr effizient ist. Das ist jedoch in einem normalen System, in dem diese mittels PCI Express geschieht, nicht der Fall. Eine solche Zusammenarbeit benötigt also eine enge Verschaltung, wie sie in APUs gegeben ist.

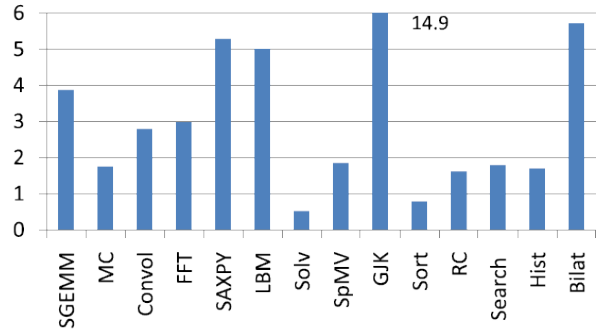


Abbildung 2: Speedup in unterschiedlichen Anwendungen bei Verwendung einer GPU im Vergleich zur Verwendung einer CPU. Aus [10].

Die Integration beider Recheneinheiten auf einem Chip ermöglicht sogar, dass sich beide Recheneinheiten denselben Speicher und Adressraum teilen. Dadurch können CPU und GPU auf denselben Daten arbeiten und bei der Verarbeitung direkt kooperieren. Zudem vereinfacht dies die Arbeit des Programmierers wesentlich, da Daten nicht mehr zwischen den Einheiten kopiert und kohärent gehalten werden müssen. Weitere Details und Auswirkungen eines solchen uniformen Speichers werden in Abschnitt 4 beschrieben.

3 Beispielarchitektur: Llano

Die Prozessorarchitektur Llano wurde 2011 von AMD eingeführt. Als erste APU ist ihr Design beispielhaft für diese Prozessorart und wird in vielen Publikationen referenziert. Die Chipfläche beträgt 227 mm^2 und wurde im 32 nm silicon-on-insulator (SOI)-Prozess hergestellt. Ein Blockdiagramm von Llano ist in Abbildung 3 dargestellt. Llano ist die erste von AMD hergestellte APU. Als solche integriert sie vier CPU-Kerne der K10-

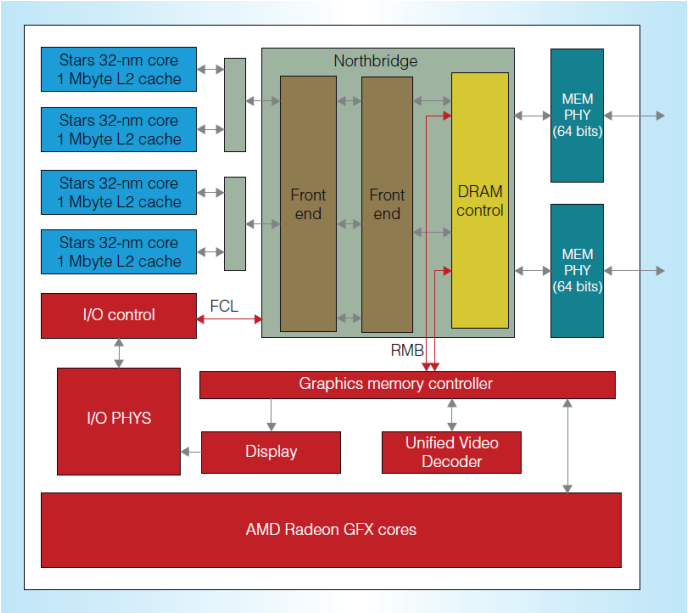


Abbildung 3: Blockdiagramm der Llano Prozessorarchitektur. Aus [5].

Mikroarchitektur und 16 SIMD-Einheiten in der GPU. Jede SIMD-Einheit besteht aus 16 VLIW-5 (very long instruction word) Kernen und jeder dieser Kerne besteht unter anderem aus vier normalen Stream-Kernen und einem speziellen Stream-Kern. Daraus ergeben sich 400 Stream-Kerne, die eine gemeinsame Rechenleistung von 480 GFLOPS erbringen. Zudem integriert Llano die Video-Dekodierer, die Northbridge und mit dieser verschiedene I/O-Controller auf demselben Chip [5].

Zwei Besonderheiten hat AMD mit seiner ersten APU eingeführt: Leistungsmanagement mittels „AMD Turbo CORE“ und ein Speichermanagement, das die Nutzung des selben Hauptspeichers durch CPU und GPU ermöglicht. Ein uniformer Speicher war dies jedoch noch nicht.

„AMD Turbo CORE“ ist eine Technologie, die je nach Anwendungsfall (CPU-lastig, GPU-lastig, OpenCL) den Takt und die Spannung der CPU- und GPU-Kerne anpasst und somit deren Rechenleistung und Stromverbrauch reguliert (dynamic voltage/frequency scaling, DVFS). Intel nennt diese Technologie „Intel Turbo Boost“. Auf diese Art und Weise kann die verfügbare Leistung immer dort eingesetzt werden, wo sie am meisten benötigt wird. Zudem kann die thermische Trägheit des Systems ausgenutzt werden, um kurzfristig eine sehr hohe Rechenleistung zu erzielen und es können deaktivierte Teile der APU als Wärmesenke verwendet werden, um anderswo mehr Wärme produzieren zu können als eigentlich erlaubt. Dies ist in Abbildung 4 dargestellt [5].

In Llano haben die CPU und die GPU unterschiedliche virtuelle Adressräume und eigene Translation Lookaside Buffer (TLB). Die GPU kann zudem keinen page fault behandeln. Sie können allerdings über Speicher kommunizieren, der zuvor in einer bekannten Position festgesetzt wird (pinning and locking). Hierzu muss das Betriebssystem zuvor die entsprechende Seite in den Hauptspeicher laden und dort festsetzen [3].

Wie in Abschnitt 2 beschreiben, besitzen CPU und GPU wesentliche Unterschiede darin, wie sie auf Speicher zugreifen und dessen Latenzen ausgleichen. CPUs verwenden hierfür Caches, da sie vor allem zufällige Zugriffe aus skalarem, sequentiellen

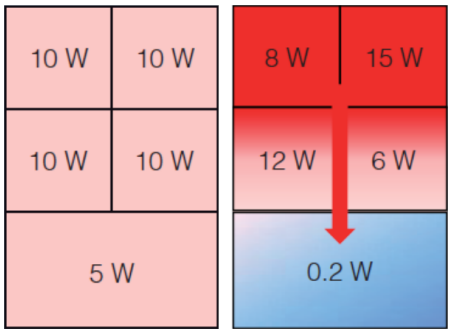


Abbildung 4: Schematische Darstellung der Funktionsweise von AMD Turbo CORE. Die vier CPU-Kerne sind oben dargestellt, die GPU unten. Auf der linken Abbildung verbrauchen alle Einheiten gleichmäßig Strom. Rechts verbrauchen einzelne CPU-Kerne deutlich mehr Strom als sie dies bei gleichermäßiger Belastung dürften. Dies wird durch die anderen, sparsameren Kerne und durch die GPU ausgeglichen. Aus [5].

Code behandlen. GPUs sind hingegen spezialisiert auf datenintensiven, vektorisierten Code und verstecken die Latenz durch Multi-Threading.
Deshalb wird der Hauptspeicher bei Llano in zwei Bereiche eingeteilt. Neben dem gewöhnlichen Systemspeicher gibt es einen bei Systemstart fixierten Bereich, der als Videospeicher der GPU verwendet wird. Der Zugriff auf diesen wird von der GPU geregelt, weshalb diese hier die volle Bandbreite des DRAM-Busses nutzen kann.
Für die gemeinsame Hauptspeicherverwaltung wurden in Llano weiters zwei neue Busse eingeführt: Der Fusion Control Link (FCL) untersucht für die GPU die CPU-Caches und ermöglicht somit einen kohärenten Zugriff der GPU auf den Systemspeicher. Der Radeon Memory Bus (RMB) erlaubt der GPU den Zugriff auf den Hauptspeicher über die Unified North Bridge (UNB).
Um die verschiedenen Speicherzugriffe zu beschreiben, muss man einerseits zwischen CPU- und GPU-Zugriffe und andererseits zwischen im CPU-Cache gepufferten, darin nicht gepufferten und Videospeicher unterscheiden [3]. Die verschiedenen Speicherzugriffe sind in Abbildung 5 dargestellt und nachfolgend beschrieben. Die erzielbaren Speicherbandbreiten sind in Tabelle 1 aufgelistet.

CPU:

1. **Speicher mit Cache-Einsatz**: Normaler Zugriff über L2-Cache.

2. **Speicher ohne Cache-Einsatz**: Schreibzugriff via Write Combine-Buffer (WC). Dieser optimiert Schreibzugriffe auf zusammenhängende Speicherbereiche.

3. **Videospeicher**: Zugriff via WC-Buffer. Da die GPU einen eigenen virtuellen Adressraum verwaltet, muss zudem vor dem Zugriff bei der GPU die physikalische Adresse angefragt werden. Hierfür wird die UNB verwendet. Zudem muss der Grafiktreiber sicherstellen, dass im Moment des Zugriffs der Hauptspeicher kohärent ist, da die CPU nicht den GPU-Cache überprüfen kann. Hierfür wird der GPU-Cache geflusht.

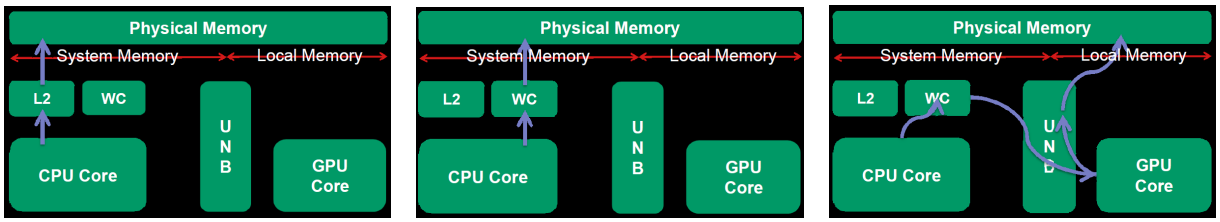
GPU:

1. **Speicher mit Cache-Einsatz**: Mittels FCL wird erst der CPU-Cache durchsucht, bevor auf den Hauptspeicher zugegriffen wird. Für beides ist Kommunikation via UNB nötig.

2. **Speicher ohne Cache-Einsatz**: Direkter Zugriff via UNB.

3. **Videospeicher**: Direkter Zugriff via UNB.

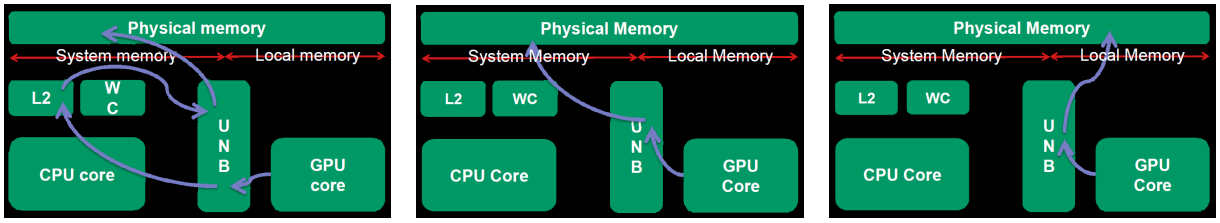
Daga et al. haben 2011 die Vorteile einer APU im Vergleich zu einer externen GPU untersucht [8]. Hierbei haben sie eine APU der Zacate Architektur verwendet. Dies ist die mobile Version der Llano Architektur und damit eng verwandt.
Für diese Untersuchung haben sie erst festgestellt, dass die Verwendung von Beschleunigern wie GPUs einen Overhead durch Kommunikation erzeugt, der in Amdahls Gesetz nicht berücksichtigt ist. Der Speedup ergibt sich dann als



(a) CPU-Zugriff auf Cache-fähigen Speicher.

(b) CPU-Zugriff auf nicht Cache-fähigen Speicher.

(c) CPU-Zugriff auf Videospeicher.



(d) GPU-Zugriff auf Cache-fähigen Speicher.

(e) GPU-Zugriff auf nicht Cache-fähigen Speicher.

(f) GPU-Zugriff auf Videospeicher.

Abbildung 5: Zugriffsmethoden auf verschiedene Speicherbereiche durch CPU (oben) und GPU (unten). „Local Memory“ entspricht dem Videospeicher. Aus [3].

Tabelle 1: Bandbreiten für unterschiedliche Speicherzugriffe. Daten aus [3].

	Cache	Ohne Cache	Video
CPU:			
Lesen	8-13 GB/s	<1 GB/s	<1 GB/s
Schreiben	8-13 GB/s	8-13 GB/s	8 GB/s
GPU:			
Lesen	4.5 GB/s	6-12 GB/s	17 GB/s
Schreiben	5.5 GB/s	6-12 GB/s	12 GB/s

$$S' = \frac{1}{s + p' + o}, \tag{1}$$

mit dem seriellen Anteil s, dem beschleunigten parallelen Anteil p' und dem Overhead des Beschleunigers o. Wegen der unterschiedlichen Kerne einer GPU gilt *nicht* $p' = p/N$ (mit der Anzahl an Prozessoren N), im Gegensatz zu symmetrischen Multikernsystemen. Dies führt zu einem Verhalten der Laufzeit wie in Abbildung 6 dargestellt.
Daga et al. haben durch mehrere Benchmarks (Molekulardynamik, FFT, Scan, Reduction) fest-

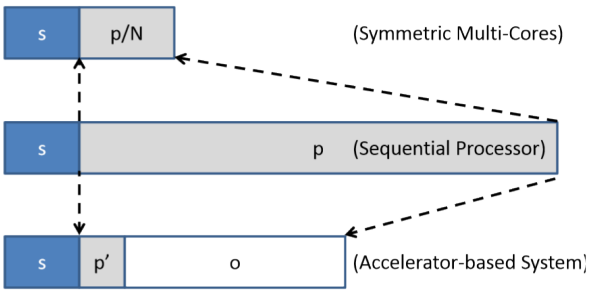


Abbildung 6: Speedup durch Mehrkernprozessoren (Symmetric Multi-Cores) und Beschleuniger wie GPUs (Accelerator-based System) im Vergleich zur sequentiellen Berechnung (Sequential Processor). Aus [8].

gestellt, dass der Overhead bei Verwendung einer APU dank des performanteren Blocktransfers um einen Faktor 6 reduziert werden kann. Bei Anwendungen, in denen die Kommunikation wesentlich ist, kann dies zu einer deutlichen Beschleunigung führen.

4 Speicherarchitektur

Bei Llano arbeiten zwar CPU und GPU mit demselben Hauptspeicher, jedoch sind deren Zugriffsmethoden sehr unterschiedlich, wie in Abschnitt 3 beschrieben. Dies macht bis zu einem gewissen Punkt auch Sinn, da deren Anforderungen üblicher Weise sehr unterschiedlich sind. Jedoch geht hierdurch der große Vorteil des uniformen Speichers verloren. Hechtman und Sorin zeigten 2013 in [9], dass kohärenter Speicher (Cache-coherent shared virtual memory, CCSVM) zu wesentlichen Zugewinnen führen kann.
Hierfür verglichen sie eine eigens entwickelte Systemarchitektur mit CCSVM und mit einer Llano APU und einer CPU von AMD. Ihre Architektur verwendete einen L1-Cache für jeden CPU- und GPU-Kern und einen globalen L2-Cache. Die GPU-Kerne verwenden eine Alpha-ähnliche Befehlssatzarchitektur (ISA) und können jeweils 8 Operationen pro Takt ausführen. In den Benchmarks (Matrixmultiplikation, All-Pairs Shortest Path, Barnes-Hut) zeigte sich, dass ihre Architektur trotz fehlender Optimierungen durchgehend deutlich schneller war als Llano. Insbesondere bei kleinen Problemgrößen zeigte sich der große Vorteil von Kohärenz und der vereinfachten Speicherverwaltung. Zudem führte diese Art der Verwaltung zu wesentlich weniger DRAM-Zugriffen.
Es stellt sich hierbei natürlich die Frage, inwiefern diese Vorteile auch für echte Szenarien gelten, da der Speicherzugriff von GPUs auf anderen Prinzipien beruht als der von CPUs. Dennoch hat kohärenter Speicher mehrere signifikante Vorteile:

- Kein Kopieren von Daten zwischen Speichersystemen nötig (zero-copy)
- CPU und GPU können effizient auf denselben Daten arbeiten und kooperieren
- Bessere Effizienz, da ein Großteil des normalen Kommunikations-Overheads entfällt
- Deutlich einfachere Programmierung
- Keine Synchronisierung in Software notwendig
- Parallelisierte CPU-Algorithmen können direkt auf die GPU portiert werden, ohne auf Kohärenz achten zu müssen

Die Performanz von kohärentem Speicher in heterogenen System kann zudem noch deutlich verbessert werden. Power et al. haben 2013 in [12] festgestellt, dass GPUs durch ihre parallelen Zugriffe mit hoher benötigter Speicherbandbreite CPUs überfordern können. Dies liegt daran, dass solche Anfragen von üblichen CPU-Cache-Strukturen schlecht gefiltert werden und deshalb sehr viele Anfragen an das Cache Directory nötig machen. Sie haben daher ein System basierend auf Region Coherence vorgeschlagen. Bei einem solchen System reservieren Caches nicht einzelne Cache-Lines, sondern große Regionen des Speichers [7]. Mit einem Region Directory konnten sie die benötigte Bandbreite an das Cache Directory in 10 verschiedenen Benchmarks im Durchschnitt um 95 % reduzieren. Die dadurch verringerte Latenzzeit des Speicherzugriffs verdoppelte die Leistung der verwendeten Benchmarks im Durchschnitt.
Aus diesen Gründen ist kohärenter Speicher auch eines der Kernprinzipien der durch die HSA Foundation entwickelten heterogenen Systemarchitektur [13]. AMD sah dies ebenfalls ein und stellte mit Kaveri 2014 eine Architektur vor, die kohärenten Speicher unterstützt [4]. AMD nennt seine Implementierung hiervon „heterogeneous Uniform Memory Access“ (hUMA). Im Gegensatz zu Llano verwenden in Kaveri CPU und GPU denselben virtuellen Adressraum, weshalb die GPU mit Pointern der CPU umgehen kann. Zudem kann in Kaveri die GPU page faults behandeln und ist deshalb nicht mehr auf festgesetzten Speicher eingeschränkt. Daher kann sowohl die CPU also auch die GPU auf den gesamten Speicherbereich zugreifen [14].

5 Programmierung heterogener Architekturen

Wie in Abschnitt 2 beschrieben, ist die Berechnung auf GPUs zwar schneller, aber nicht wesentlich. Bordawekar et al. haben zudem gezeigt [2], dass

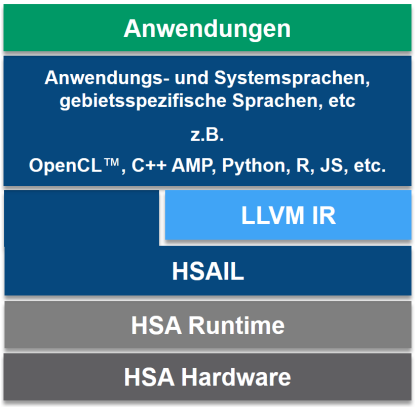


Abbildung 7: Der HSA Software Stack. Die HSA Runtime ermöglicht heterogene Parallelität mittels Warteschlangen. Die Architektur wird durch den HSA Intermediate Layer (HSAIL) abstrahiert und der Programmcode per LLVM optimiert. Aus [15].

dieser Geschwindigkeitsvorteil mit einer wesentlich komplexeren und zeitaufwändigeren Software-Entwicklung erkauft wird. Hierfür haben sie mehrere Anwendungen manuell für Intel Westmere und IBM Power7 Prozessoren parallelisiert und anschließend mittels CUDA auf eine Nvidia Fermi GPU portiert. Der erreichte Speedup lag zwischen 2 und 7, bei einer beinahe Verzehnfachung der Entwicklungszeit.

Die Vereinfachung der Software-Entwicklung und -Optimierung ist daher eine wesentliche Voraussetzung für die Verwendung von heterogenen Rechnerarchitekturen. Dies ist das Hauptziel der HSA Foundation [13]. Ein wesentlicher Punkt hierbei ist natürlich der bereits besprochene uniforme Speicher. Allerdings müssen die Programmierwerkzeuge auch die massive Parallelität von GPUs unterstützen und die genauen Eigenschaften der unterschiedlichen Rechenwerke abstrahieren.

Um dies zu erreichen, wird bei HSA eine Syntax verwendet, durch die Aufgaben asynchron an die passendsten Rechenwerke verteilt werden kann. Hierfür verwaltet die HSA Runtime Warteschlangen für alle Rechenkerne. Diese sind in Hardware umgesetzt und unterstützen Parallelität sowohl innerhalb einer einzelnen als auch zwischen mehreren Warteschlangen. Zudem wird hierdurch imlizit präemptives Multitasking unterstützt. Jede Aufgabe besitzt einen einfachen Parallelitätsspezifizierer (parallelism specifier), der Informationen für die parallele Ausführung wie den Bereich bzw. das Gitter und die Bearbeitungsgruppe enthält. Aufgaben aus diesen Warteschlangen kann der Kernel zwischen Rechenkernen verschieben und somit die Arbeitslast effizient verteilen [15].

Um heterogene Parallelität unabhängig von der jeweiligen Architektur zu unterstützen, wird zudem die Architektur durch eine Zwischenschicht namens HSAIL (HSA Intermediate Layer) abstrahiert. Hierbei handelt es sich um eine virtuelle Befehlssatzarchitektur (Instruction Set Architecture, ISA), welche durch einen JIT Kompilierer in die eigentliche ISA übersetzt wird. Eine Übersicht über den HSA Software Stack gibt Abbildung 7 [13, 15].

Der HSA Software Stack ist unabhängig von der jeweiligen Programmiersprache. Somit werden OpenCL und alle zugehörigen Werkzeuge direkt unterstützt. Neben OpenCL gibt es noch einige andere Programmiersprachen und Werkzeuge, die die Verwendung von GPUs ermöglichen und vereinfachen (z.B. C++ AMP). Durch die besprochenen Vorteile von APUs sind diese hier besonders einfach zu verwenden und zu entwickeln [13].

Weiters gibt es mit BOLT[1] eine eigene Bibliothek für die Verwendung von HSA, die viele übliche und fortgeschrittene Routinen unterstützt und mit OpenCL und C++ AMP zusammenarbeiten kann. Diese unterstützt sämtliche Vorteile der HSA (zero-copy) und vereinheitlicht als eine der ersten Bibliotheken den Code von CPU und GPU.

Java kann zudem mittels APARAPI[2] parallelisiert werden. Hierfür wird Java Bytecode in OpenCL übersetzt oder mittels Thread-Pool parallelisiert. Auf HSAs wird hierbei bereits HSAIL verwendet. AMD und Oracle arbeiten weiters gerade am Sumatra-Projekt, durch das die HSA künftig direkt von der Java Virtual Machine (JVM) unterstützt wird.

[1] https://github.com/HSA-Libraries/Bolt
[2] http://aparapi.github.io/

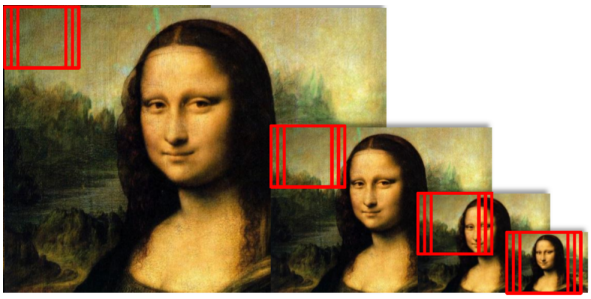


Abbildung 8: Bei der Gesichtserkennung mittels Haar-Filter wird ein Suchquadrat an jede Stelle des Bildes gesetzt, mit unterschiedlichen Skalierungen. Aus [13].

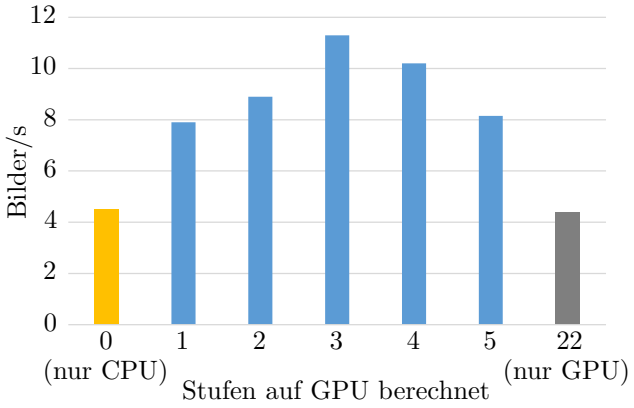


Abbildung 9: Performanz der Gesichtserkennung bei Berechnung einer unterschiedlichen Anzahl an Erkennungsstufen auf der GPU. Daten aus [13].

6 Anwendungen

Die Kombination von CPU und GPU bringt Vorteile in vielen Anwendungen, die sowohl aus parallelen also auch aus sequentiellen Teilen bestehen. Ein Beispiel hierfür ist die Gesichtserkennung mittels Haar-Filter, wie in [13] beschrieben.
Hierbei wird ein Suchquadrat an jede Stelle des Bildes gesetzt. Dies geschieht mit unterschiedlichen Skalierungen, um unterschiedlich große Gesichter zu erkennen (siehe Abbildung 8). Die Suche in jedem Suchquadrat ist in mehrere Stufen unterteilt. In jeder Stufe werden unterschiedliche Merkmale untersucht und in jeder Stufe kann abgebrochen werden. Diese Anwendung erfordert anfangs eine extrem hohe Datenrate, jedoch erreichen nur sehr wenige Suchquadrate die späteren Stufen. Deshalb kann eine GPU alleine diese Berechnung kaum beschleunigen. Wird jedoch eine GPU für die ersten Stufen der Berechnung verwendet und eine CPU für die späteren, so ist ein Speedup von ungefähr 2.5X erreichbar (siehe Abbildung 9), und das bereits mit APUs der Llano-Architektur, welche die Funktionen der HSA nur teilweise umsetzt.
Doch APUs bringen nicht nur dann Vorteile, wenn beide Recheneinheiten zusammenarbeiten. Yang et al. haben in [17] untersucht, wie die CPU verwendet werden kann, um von der GPU benötigte Daten im Voraus in den gemeinsamen L3-Cache zu laden (Prefetching). Der so optimierte Speicherzugriff führte in mehreren Benchmarks zu einem Leistungsgewinn von durchschnittlich 21.4 %.
Wang et al. haben zudem in [16] untersucht, wie die Arbeitslast für beliebige Anwendungen so verteilt werden kann, dass eine maximale Leistungseffizienz und Rechenleistung erzielt wird. Hierfür haben sie einen einfachen Algorithmus entwickelt, der Aufgaben abhängig von der Recheneffizienz zuteilt und die lokale Frequenz und Spannung und die Anzahl aktiver CPU-Kerne steuert (dynamic voltage/frequency/core scaling, DVFCS). Hiermit konnten sie über mehrere Benchmarks hinweg einen durchschnittlichen Leistungsgewinn von 13 % im Vergleich zu einer fixen Leistungs- und Lastverteilung erzielen. Im Vergleich zur reinen Verwendung der CPU oder GPU ergab sich ein noch wesentlich höherer Leistungszuwachs.

7 Zusammenfassung und Ausblick

In dieser Arbeit wurden die wesentlichen Vorteile der Kombination von CPU und GPU im Detail beschrieben und begründet. Diese basieren vor allem auf uniformem Speicher und der einfacheren Kooperation zwischen CPU und GPU. Um die Möglichkeiten der Kooperation aufzuzeigen, wur-

den die Unterschiede von CPU und GPU herausgearbeitet. Beispielhaft wurde die Llano Systemarchitektur vorgestellt und einige Details zur Speicherarchitektur in APUs untersucht. Zudem wurden Prinzipien der Programmierung von APUs und HSAs im Allgemeinen vorgestellt. Zuletzt wurden die Vorteile von APUs mittels Anwendungen und Benchmarks dargelegt.
In Zukunft wird die Integration von CPU und GPU auf einem Chip zum Standard avancieren. Die Vorteile werden durch schnelle Speichertechnologien wie High Bandwidth Memory (HBM) und bessere Cache-Architekturen noch ausgebaut. Zudem dürfte das Programmierkonzept der HSA Foundation in Zukunft immer wichtiger werden. ASICs und andere spezialisierte Rechenwerke werden zunehmend auf CPUs und Systems on a Chip (SoCs) integriert und dieses Konzept kann solche Einheiten problemlos verwalten.

Literatur

[1] AMD. *APU 101: All about AMD Fusion Accelerated Processing Units*. 2011. URL: `http://amd-dev.wpengine.netdna-cdn.com/wordpress/media/2012/10/apu101.pdf` (besucht am 19. 12. 2015).

[2] R. Bordawekar, U. Bondhugula und R. Rao. "Can CPUs Match GPUs on Performance with Productivity?: Experiences with Optimizing a FLOP-intensive Application on CPUs and GPU". In: *IBM Research Report* (2010).

[3] P. Boudier und G. Sellers. *Memory system on fusion APUs*. 2011.

[4] D. Bouvier und B. Sander. *Applying AMD's Kaveri APU for Heterogeneous Computing*. 2014.

[5] A. Branover, D. Foley und M. Steinman. "AMD Fusion APU: Llano". In: *IEEE Micro* 2 (2012), S. 28–37.

[6] N. Brookwood. "AMD Fusion family of APUs: enabling a superior, immersive PC experience". In: *Insight* 64.1 (2010), S. 1–8.

[7] J. F. Cantin u. a. "Coarse-grain coherence tracking: RegionScout and region coherence arrays". In: *IEEE Micro* 26.1 (2006), S. 70–79.

[8] M. Daga, A. M. Aji und W.-c. Feng. "On the Efficacy of a Fused CPU+GPU Processor (or APU) for Parallel Computing". In: *2011 Symp. Applicat. Accelerators in High-Performance Computing*. IEEE Comp. Soc., 2011, S. 141–149.

[9] B. A. Hechtman und D. J. Sorin. "Evaluating cache coherent shared virtual memory for heterogeneous multicore chips". In: *2013 IEEE Int. Symp. on Performance Analysis of Systems and Software*. IEEE Comp. Soc., 2013, S. 118–119.

[10] V. W. Lee u. a. "Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU". In: *37th Int. Symp. Computer Architecture*. ACM Press, 2010, S. 451–460.

[11] T. Mitra. "Heterogeneous Multi-core Architectures". In: *Information and Media Technologies* 10.3 (2015), S. 383–394.

[12] J. Power u. a. "Heterogeneous system coherence for integrated CPU-GPU systems". In: *46th IEEE/ACM Int. Symp. Microarchitecture*. IEEE Comp. Soc., 2013, S. 457–467.

[13] P. Rogers. *Heterogeneous system architecture overview*. 2013.

[14] P. Rogers, J. Macri und S. Marinkovic. *heterogeneous Uniform Memory Access*. 2013.

[15] V. Tipparaju und L. Howes. *HSA for the Common Man*. 2012.

[16] H. Wang u. a. "Workload and power budget partitioning for single-chip heterogeneous processors". In: *21st Int. Conf. Parallel Architectures and Compilation Techniques*. ACM Press, 2012, S. 401–410.

[17] Y. Yang u. a. "CPU-assisted GPGPU on fused CPU-GPU architectures". In: *IEEE 18th Int. Symp. High Performance Computer Architecture*. IEEE Comp. Soc., 2012, S. 1–12.