Abbau von Mehraufwand durch Speichervirtualisierung für Kernel-Bypassing-Anwendungen

Vorstellung und Evaluation zweier verschiedener Lösungsansätze

Fabian Müller
Friedrich-Alexander-Universität Erlangen-Nürnberg
fabian0.mueller@fau.de

Zusammenfassung

Hardwarebeschleunigung spielt eine immer größere Rolle. Heterogene Systeme, welche eine Vielzahl dieser Beschleuniger integrieren, werden aus verschiedenen Gründen immer populärer. Sie sparen Energie, entlasten den Hauptprozessor, rechnen parallel und selbstständig und erhöhen die Geschwindigkeit, mit welcher Berechnungen ausgeführt werden können. Zur Erhöhung des Datendurchsatzes kann Kernel Bypassing eingesetzt werden. Dabei werden jedoch neue Performanzprobleme sichtbar. Es zeigt sich, dass die aktuell verwendeten Kommunikationssysteme einen Flaschenhals darstellen, weswegen die Geschwindigkeit begrenzt wird. Daher wurden in der Forschung verschiedene neue Ansätze entwickelt, welche die etablierten Systeme erweitern beziehungsweise ersetzen. Die vorliegende Arbeit untersucht die verbreiteten Systeme und zeigt, an welchen Stellen Mehraufwand entsteht. Danach werden zwei neue Ansätze exemplarisch vorgestellt, die diesen zu reduzieren versprechen, und miteinander sowie mit ihren Vorgängern verglichen.

1 Einleitung

Hardwarebeschleunigung spielt seit jeher eine große Rolle in Systemen. Hierbei übernehmen sogenannte Beschleuniger bestimmte Aufgaben, auf die sie spezialisiert und welche sie effizient durchführen können. Systeme, die neben dem Hauptprozessor weitere, spezialisierte Prozessoren enthalten, welche vom Hauptprozessor bestimmte Aufgaben delegiert bekommen, werden als *heterogene Systeme* bezeichnet.

Es gibt die verschiedensten Arten von Beschleunigern. Zu den ersten weit verbreiteten gehören die mathematischen Coprozessoren für Gleitkommaberechnungen, welche später direkt in die Prozessoren integriert wurden [5]. Ein weiteres bekanntes Beispiel sind Grafikprozessoren, welche im Bereich der Bildverarbeitung stetig weitere Funktionen übernehmen. Anfangs nur zur Ausgabe von Bildern genutzt, wurden sie Anfang der neunziger Jahre zu Beschleunigern für die Berechnung von 2D- und 3D-Modellen ausgebaut [1]. Später wurden zusätzliche Beschleuniger integriert, beispielsweise zur En- und Dekodierung von Videomaterial [4, 10]. Weiterhin wurden Schnittstellen eingeführt, um generische Berechnungen auf Grafikprozessoren durchführen zu können. Dieses *General Purpose Computing on Graphics Processing Unit (GPGPU)* genannte Verfahren ermöglichte es Programmierern, Probleme, die sich effizient nebenläufig lösen lassen, auf eine große Anzahl kleinerer, wenn auch weniger genauerer, Prozessoren zu verteilen und Berechnungen wie beispielsweise die schnelle Fourier-Transformation sehr viel schneller ausführen zu können, als dies auf dem Hauptprozessor möglich gewesen wäre [8, 14, 17].

Wie sich gezeigt hat, können durch den Einsatz von Hardwarebeschleunigern Effizienzsteigerungen sowohl bei der Ausführungszeit als auch bei der Energiesparsamkeit erzielen lassen. Ihr Erfolg zeigt sich vor Allem an der steigenden Menge an Beschleunigern in Systemen. Auch das Ende des *Dennard Scaling* spricht für den verstärkten Einsatz von Beschleunigern [7, 15].

Anwendungen können zur Steigerung der Performanz Geräte direkt im Rahmen des *Kernel Bypassing* nutzen. Dabei entfallen Hardwareabstraktionsschichten, welche im Betriebssystem enthalten sind, was bei Verwendung der Geräte viele Kontextwechsel erfordert. Dies wird durch *gemeinsame Speicherbereiche* mit den Geräten realisiert. Diese Variante wird für Hochperformanzanwendungen bevorzugt, da Zwischeninstanzen, welche zusätzlichen Rechen- und Speicheraufwand verursachen, entfallen können. Es zeigt sich allerdings, dass es weiterhin Flaschenhälse gibt, welche den Datendurchsatz drosseln, zum Beispiel die für die Speichervirtualisierung nötigen Speicherverwaltungseinheiten, welche Zugriffe prüfen und virtuelle Adressen bei Zugriffen übersetzen.

Die vorliegende Arbeit zeigt zunächst auf, wie Prozesse und Beschleuniger im Rahmen des *Kernel Bypassing* miteinander kommunizieren. Das Konzept der *Speichervirtualisierung* wird vorgestellt und die sehr bekannte Implementierung durch *Speicherverwaltungseinheiten* näher betrachtet und deren Schwachstellen im Hinblick auf die Performanz identifiziert. Dann werden zwei Alternativen für den virtualisierten Speicherzugriff durch Beschleuniger vorgestellt und miteinander sowie den vorangegangenen Systemen verglichen.

2 Kommunikation mit Beschleunigern

Beim Einsatz von heterogenen Systemen, in welchen Aufgaben an einen oder mehrere Beschleuniger delegiert werden, stellt sich die Frage, wie die Kommunikation zwischen Programmen und Beschleunigern stattfinden kann und soll. Die Beschleuniger benötige die Daten, auf welche sie ihre Algorithmen anwenden sollen. Weiterhin müssen teilweise die Ergebnisse zurück ins System übergeben werden[1]. Hierzu wird *gemeinsam genutzter Speicher* eingesetzt: Es werden Speicherbereiche angelegt, in welchen die Daten gelegt werden und dann dem Beschleuniger freigegeben werden. Der Beschleuniger seinerseits greift auf die Daten dort zu, führt seine Aufgaben aus und legt gegebenenfalls seine Ergebnisse in einen solchen Speicherbereich.

In früheren Systemen wurden Beschleuniger über den Hauptprozessor an den Speicherbus angebunden. Dadurch wurde bei größeren Datenmengen sehr viel Mehraufwand erzeugt, welcher den Hauptprozessor zusätzlich belastete. Dieser Aufwand kann durch das Umgehen des Hauptprozessors bei der Anbindung von Beschleunigern an den Speicherbus erheblich reduziert werden. Eine Möglichkeit ist es, Beschleunigern direkt Zugriff auf den Arbeitsspeicher zu erlauben. Diese Methode nennt sich *Speicherdirektzugriff* (engl. *direct memory access (DMA)*) [19]. Dieses Zugriffsschema ist in Abb. 1 schematisch dargestellt. Reiner Speicherdirektzugriff ist eine einfache Variante, um Daten mit anderen Geräten auszutauschen, da keine weiteren Einheiten zwischen Gerät und Speicherbus existieren und Anfragen direkt auf dem Speicherbus ausgeführt werden können, hat aber auch einige gravierende Nachteile. Ohne Zwischeninstanzen wird der Speicherzugriff von Geräten auf den Speicherbus nicht kontrolliert. Fehlerhafte oder böswillige Geräte können also auf Speicherbereiche zugreifen, die nicht für sie gedacht sind. Dies ist in mehrerer Hinsicht problematisch. Sensible Daten können nicht vor Zugriffen geschützt werden. Außerdem können die Geräte durch das Beschreiben von Speicherbereichen den Zustand anderer Programme oder auch das Betriebssystem verändern. Ohne sehr großes Vertrauen zu den Geräten zu haben, ist Speicherdirektzugriff ohne eine vertrauenswürdige Zwischeninstanz, welche zum Beispiel durch das Betriebssystem gesteuert wird und fehlerhafte oder falsche Zugriffe verhindert, für deren Anbindung an das System [7, 16].

Aber auch funktional ist Speicherdirektzugriff nicht ausreichend. Multi-Prozess-Betriebssysteme benutzen häufig *Speichervirtualisierung* und *Speicherverwaltungseinheiten* (engl. *Memory Management Units (MMUs)*),

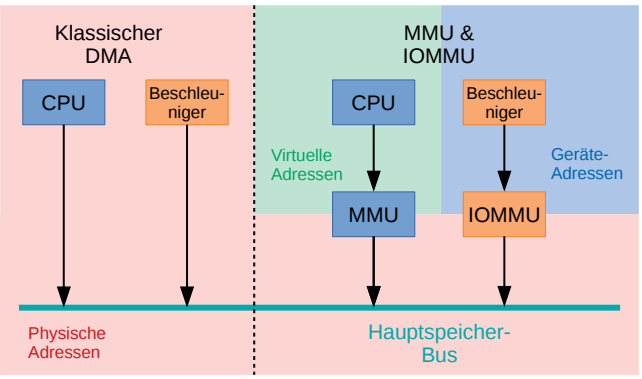

Abbildung 1. Vereinfachte Darstellung der Anbindung auf den Speicherbus von Hauptprozessor und Beschleunigern. Links zu sehen ist der klassische Speicherdirektzugriff, rechts Speicherzugriff unter Verwendung von Speicherverwaltungseinheiten (*MMU*) und *IOMMU*. Es werden die verschiedenen Adressräume farbig dargestellt. Abgeleitet aus [7, Figure 1]

um den Zugriff von Programmen auf Speicherbereiche einzuschränken und heterogene Speichersysteme in einem gemeinsamen Adressraum darzustellen [19]. Der Hauptprozessor greift also nicht mehr direkt per physischer Adressen auf den Speicherbus zu, da zunächst die virtuellen Adressen, die die Prozesse intern nutzen, in physische übersetzt werden müssen. Dies übernimmt die Speicherverwaltungseinheit, welche vom Betriebssystem verwaltet wird. Sie kontrolliert außerdem, auf welche Bereiche im Adressraum des Speicherbusses der laufende Prozess zugreifen darf. Die Verwendung virtueller Adressräume ermöglicht es dem Betriebssystem zudem, Prozesse voneinander zu isolieren und den Speicher lückenlos und effizient auszunutzen.

Beschleuniger allerdings werden vor das Problem gestellt, auf nicht reale, virtuelle Speicheradressen zeigen, die die Programme intern verwenden. Zur Lösung dieses Problems wurden *Input/Output Memory Management Units (IOMMUs)* eingeführt [3, 9, 12], deren Einbindung in das System in Abb. 1 dargestellt ist. Analog zur Speicherverwaltungseinheit, die an den Hauptprozessor angegliedert ist, steht sie als Bindeglied zwischen dem Speicherbus und den Beschleunigern. Das Konzept ist relativ alt. Selbst sehr alte Eingabe-/Ausgabe-Busse wie *AGP* aber auch moderne wie *PCI-Express* greifen darauf zurück [2, 3, 9]. Die *IOMMU* wird genauso vom Betriebssystem verwaltet wie die Speicherverwaltungseinheit des Hauptprozessors und regelt, welches Gerät Zugriff auf welche Bereiche im Hauptspeicher hat. Im Kontext der Beschleuniger spricht man von *Geräteadressen* statt *virtuellen Adressen*. Das Betriebssystem reserviert ein an die *IOMMU* angebundenes System für einen einzelnen

[1] Beschleuniger, die wie zum Beispiel Grafikkarten die verarbeiteten Daten direkt ausgeben, liefern ihre Ergebnisse im Allgemeinen nicht zurück ans System.

Prozess und konfiguriert die *IOMMU* auf die entsprechenden Speicherbereiche.

Die für die Speichervirtualisierung benötigten Verwaltungsdaten liegen im Hauptspeicher und werden *Seitentabellen* genannt. Die Zuordnung von physischen Speicherbereichen zum virtuellen bzw. Geräteadressraum wird von den Speicherverwaltungseinheiten durchgeführt. Verwaltet werden die Bereiche vom Betriebssystem.

3 Performanzprobleme

Verschiedene Forschungsberichte identifizieren mehrere Probleme bei der Verwendung von *IOMMUs*. *Haria et al.* nennen unter anderem die notwendigen Adressübersetzungen, die an den Schnittstellen zwischen den (virtuellen und physischen) Adressräumen erforderlich sind, als ein die Ausführungsgeschwindigkeit von Instruktionen negativ beeinflussendes Element in der Kette [7]. Hinzu kommt, dass die virtuellen Adressräume von Beschleunigern und Prozessen nicht notwendigerweise gleich sind, was bedeutet, dass die Daten zwischen den Adressräumen hin- und herkopiert werden müssen [11, 13]. Außerdem sind Zeiger, die in den verschiedenen Adressräumen verwendet werden, in anderen Adressräumen ungültig, was zusätzliche Übersetzungen notwendig macht. Abb. 1 zeigt, an welchen Stellen Übersetzungen notwendig sind. *Olson et al.* nennen ebenfalls Adressübersetzungen, welche bei jedem Zugriff notwendig sind, sowie das Fehlen von Adresszwischenspeichern auf den Geräten als Probleme [15].

Die Verwendung von Speichervirtualisierung ist ein Kompromiss zugunsten der Sicherheit. *Haria et al.* führen aus, dass die Speicherzugriffstechnologien, die bisher entwickelt wurden, entweder auf Geschwindigkeit oder auf Sicherheit ausgelegt wurden, aber nicht beides [7]. Hierzu vergleichen sie den unsicheren *Speicherdirektzugriff*, welcher direkt physische Adressen nutzt und durch das Fehlen jedweder Kontrollinstanzen, sehr hohe Datendurchsatzraten erzielen kann, mit der *IOMMU*, welche aus Sicherheitsgründen zu bevorzugen ist, aber deutlichen Mehraufwand erfordert.

Die Seitentabellen belegen zusätzlichen Hauptspeicher neben den Nutzdaten. Je nach Anzahl und Größe der Seiten entsteht hier Speichermehraufwand [7, 15].

Bei vielen modernen Betriebssysteme, zum Beispiel Linux, werden *Speicherseiten* Speicher erst bei Bedarf alloziert wird [6, 19]. Bei Anfragen auf nicht allozierte Speicherseiten werden *Seitenfehler* ausgelöst, welche das Betriebssystem dazu veranlassen, eine neue Seite zu reservieren und in der Verwaltungseinheit zu dokumentieren. Dies hat zwar einige Vorteile, unter Anderem lässt sich der Hauptspeicher so lückenlos und effizient ausnutzen, bedeutet aber zeitgleich, dass Verzögerungen durch aufwändige Kontextwechsel in den Kernelraum und zurück erforderlich sind.

4 Neue Ansätze als Alternativen zur IOMMU

Zur Steigerung der Kommunikationsperformanz müssen neue Ansätze gefunden werden, welche weiterhin die Sicherheit garantieren. Wie bereits in Abschnitt 3 ausgeführt wird, entsteht durch den Einsatz von Speichervirtualisierung an mehreren Stellen zusätzlicher Aufwand. In der aktuellen Forschung zeigen verschiedene Arbeiten, dass die größte Schwachstelle hierbei die Speicherverwaltungseinheiten darstellen, die einerseits einen Flaschenhals darstellen, und zudem als generische Lösung nicht an die Speicherzugriffs- und -durchsatzbedürfnisse einzelner Geräte angepasst werden können [7, 15].

Der folgende Abschnitt stellt zwei Systeme vor, die die *IOMMU* erweitern bzw. ersetzen. Dabei bieten sie die selben Schutzgarantien wie diese. Zunächst wird *Border Control* betrachtet, welches leichtgewichtig Geräten den Speicherdirektzugriff mittels physischer Adressen ermöglicht und lediglich bei Lese- und Schreibzugriffen auf den Speicherbus überprüft, ob diese zulässig sind oder nicht. Die Beschleuniger müssen selbstständig ihre virtuellen Adressen in physische übersetzen und Pufferspeicher bereitstellen, um Ergebnisse zwischenzuspeichern, können diese aber dafür bestmöglich ihren Bedürfnissen nach gestalten. Danach wird *Devirtualized Memory* betrachtet, eine Komplettlösung, welche die *IOMMU* funktional erweitert und durch die, mit einigen Ausnahmen, direkte Verwendung physischer Adressen im virtuellen Adressraum sowie Erweiterungen der Speicherverwaltung im System eine Steigerung der Performanz erzielt. Die beiden Systeme unterscheiden sich stark in der Herangehensweise, haben aber ähnliche Zielsetzungen.

4.1 Border Control

Border Control bezeichnet sich als „Sandboxing-System“ für Beschleuniger [15]. Die Beschleuniger selbst müssen virtuelle Adressen zu physischen wandeln und können mit diesen dann auf den Speicherbus zugreifen. Um die Sicherheit zu garantieren, wird zwischen Speicherbus und Beschleuniger jeweils eine *Grenzkontrolle* eingeführt, welche bei jedem Zugriff die Berechtigungen des Beschleunigers überprüft und entweder den Zugriff gewährt oder unter Benachrichtigung des Betriebssystems abweist.

Das System vergleicht sich mit der Grenzkontrolle beim Übergang zwischen Staaten. Hier legen Personen einen Identitätsnachweis vor, mit welchem überprüft wird, ob der Übertritt gewährt werden kann. Liegen Gründe vor, die diesen ausschließen, werden die Reisenden direkt abgewiesen, andernfalls dürfen sie übertreten. Dieser Vergleich bietet sich durchaus an. *Border Control*

selbst garantiert lediglich, dass Zugriffe auf Speicherregionen durch das Betriebssystem autorisiert werden, genau wie Länder an ihren Grenzen den Personenverkehr steuern oder verhindern können. Es kann allerdings keine Aussage getroffen werden, welche Aktivitäten die Beschleuniger nach Gewährung des Zugriffs im Speicher vornimmt, ähnlich wie die Aktivitäten von Reisenden normalerweise vor und nach der Grenze nicht überwacht werden. Daher müssen einzelne Prozesse, welche mit Beschleunigern arbeiten, diesen vertrauen können. Hierbei wird der Vergleich zu Softwarebibliotheken von dritten Parteien gezogen, welchen Programme bei deren Verwendung genauso vertrauen müssen.

Border Control's Ziel ist der Schutz eines Systems vor unzulässigen Speicherzugriffen durch fehlerhafte oder böswillige Beschleunigern mit einhergehender Steigerung der Performanz im Vergleich zu *IOMMU* und anderen Systemen.

4.1.1 Komponenten

Abb. 2 zeigt den Aufbau eines Systems, welches *Border Control* zur Anbindung der Beschleuniger nutzt. Diese müssen einen Teil der Speicherverwaltungseinheiten selbst implementieren. Sie müssen virtuelle Adressen selbstständig bei einem *Adressübersetzungsdienst* auflösen. Mittels der physischen Adressen erfolgt dann der Speicherzugriff über die *Border Control*-Einheit.

Border Control spezifiziert drei Komponenten, welche für den Betrieb benötigt werden. Die *Protection Table* enthält die Zugriffsberechtigungsdaten. Für jeden angebundenen Beschleuniger existiert eine eigene Tabelle. Sie wird im Hauptspeicher angelegt. Weiterhin existiert ein *Border Control Cache*, welcher Teile der *Protection Tables* zwischenspeichert und einen Großteil der verhältnismäßig langsamen Zugriffe auf die *Protection Tables* im Hauptspeicher einspart. Bei Zugriffen auf den Speicherbus wird dieser zunächst konsultiert und die Berechtigungen für die Speicheradresse geprüft. Zusätzlich wird ein *Adressübersetzungsdienst* spezifiziert, welcher den Beschleunigern über eine Schnittstelle bereit gestellt werden muss, sodass sie virtuelle Speicheradressen in physische vor Speicherzugriffen übersetzen können.

Protection Table Abb. 2 zeigt den Aufbau der *Protection Table*. Es handelt sich um eine flache Tabelle, in welcher die Daten, welche zur Überprüfung von Speicherzugriffen notwendig sind, hinterlegt werden. Das System ist so entworfen, dass keine Abbildungen von physischen auf virtuelle Adressen und umgekehrt notwendig sind. Stattdessen wird gespeichert, ob für die Adressen der Speicherseiten Lese- oder Schreibberechtigungen vorhanden sind. Pro physischer Speicherseite werden 2 Bit benötigt, um jeweils die Lese- und Schreibrechte zu kodieren. Selbst für den gesamten physischen Speicherbereich lässt sich so äußerst kompakt eine Tabelle für sämtliche physischen Adressbereiche erzeugen. Die Autoren verdeutlichen dies durch ein einfaches Rechenbeispiel anhand eines Computers mit 64-Bit-Architektur und 16GB Speicher. Bei einer minimalen Speicherseitengröße von 4kB würde maximal rund 1MB pro Beschleuniger und Prozess für die *Protection Table* benötigt. Dies entspricht einem zusätzlichen Speicherbedarf von 0,006%.

Durch das Verwalten des gesamten physischen Adressraumes in einer Tabelle ist zu erwarten, dass große Teile der Tabelle brach liegen. Aufgrund der geringen Größe sei dies jedoch zu vernachlässigen.

Border Control Cache Die *Protection Table* muss bei jedem Speicherzugriff durch einen Beschleuniger abgefragt werden. Da diese im Hauptspeicher verwaltet wird, würden ständige Zugriffe hierauf den Aufwand vervielfachen. Um dieses Problem abzumildern, wird ein Pufferspeicher eingesetzt, welcher die zuletzt verwendeten Berechtigungen vorhält. Dessen Aufbau wird in Abb. 2 gezeigt. Pro Beschleuniger ist ein solcher Pufferspeicher vorzusehen.

Aus Effizienzgründen werden die Informationen blockweise aus dem Hauptspeicher geholt und zwischengespeichert. Die Einträge des Pufferspeichers sind 128 Byte groß. Somit sind jeweils die Berechtigungen von 512 Speicherseiten in einem Eintrag hinterlegt, da pro Seite 2 Bit benötigt werden. Selbst kleine Pufferspeicher können durch die kompakte Darstellung große Speicherbereiche abdecken. Ein Speicher mit 64 Einträgen, welcher 8kB groß wäre, kann bei Seitengrößen von 4kB ganze 128MB des Adressraumes des Speicherbusses abdecken. Bei größeren Pufferspeichern bzw. größeren Seiten steigt die Abdeckung der Adressräume entsprechend.

Address Translation Service Der *Adressübersetzungsdienst* ist eine Komponente, welche für den Betrieb von *Border Control* unerlässlich ist und als vom System bereitzustellend spezifiziert. Ihre Aufgabe ist es, virtuelle Adressen für die Beschleuniger in physische zu übersetzen. Der Dienst wird Beschleunigern über eine Schnittstelle bereitgestellt und vom Betriebssystem verwaltet.

4.1.2 Betrieb

Der folgende Abschnitt erklärt kurz, wie der Betrieb eines Beschleunigers mit *Border Control* aussieht. *Border Control* unterstützt auch Beschleuniger, welche mehrere Prozesse parallel ausführen können. Aus Gründen der Einfachheit beschränken sich die folgenden Ausführungen jedoch auf die Funktionsweise bei Verwendung von Beschleunigern, welche parallel nur einen Prozess bedienen können.

Vor einer Verwendung eines Beschleunigers initialisiert das Betriebssystem zunächst die *Protection Table* für den jeweiligen Beschleuniger. Diese bleibt zunächst *genullt*,

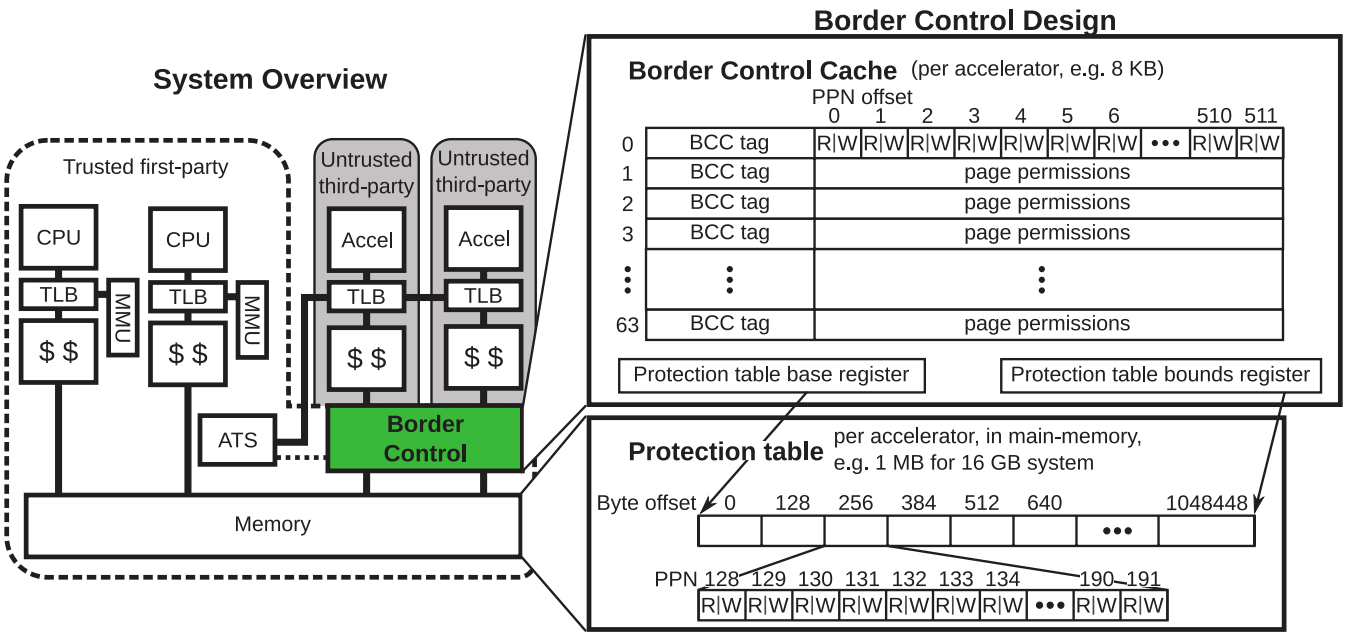

Abbildung 2. Implementierung von *Border Control*. Links wird der logische Aufbau eines heterogenen Systems mit *Border Control* gezeigt. Die rechte Seite zeigt den Aufbau der Kernkomponenten *Protection Table* und *Border Control Cache*. Weiterhin zu sehen ist der von den Beschleunigern benötigte *Address Translation Service*. [15, Figure 2].

d.h. es sind keine Berechtigungen eingetragen, und wird erst später bei Bedarf befüllt.

Das Betriebssystem muss den *Adressübersetzungsdienst* für den Beschleuniger entsprechend konfigurieren, damit dieser für ihn sichtbare *Geräteadressen* in physische übersetzen kann. Das Ergebnis jeder Übersetzung wird sowohl an den Beschleuniger als auch an den *Border Control Cache* weitergegeben. Letzterer prüft, ob er einen zugehörigen Eintrag enthält. Sollte der Eintrag vorhanden sein, aber die Berechtigungen zu niedrig sein, wird der Eintrag aus der *Protection Table* geholt und aktualisiert. Bei Fehlen des Eintrags wird ein neuer Pufferspeicher-Eintrag erzeugt und die Daten werden aus der *Protection Table* angefragt. Dadurch steigt die Wahrscheinlichkeit, dass beim tatsächlichen Zugriff auf den Speicherbus die fraglichen Daten bereits im *Border Control Cache* vorrätig sind.

Jeder Zugriff auf den Speicherbus erfolgt durch die *Border Control*-Schnittstelle. Dieses prüft für jeden Zugriff, ob dieser berechtigt ist. Dazu wird zuerst der *Border Control Cache* befragt. Sollte kein Eintrag vorhanden sein, wird ein solcher angelegt und mit den Informationen aus der *Protection Table* befüllt. Wenn sich herausstellt, dass der Prozess keine Berechtigung für diesen Zugriff hat, wird das Betriebssystem benachrichtigt. Dies funktioniert ähnlich wie bei der *Schutzverletzung* eines Prozesses, welche durch die Speicherverwaltungseinheit ausgelöst werden kann. Das Betriebssystem muss nun entsprechend reagieren. Die Möglichkeiten umfassen beispielsweise das Beenden des Prozesses oder das Deaktivieren des Beschleunigers. In jedem Fall wird die Anfrage von *Border Control* abgewiesen und nicht ausgeführt, d.h. die Lese-/Schreiboperation wird nicht an den Speicherbus weitergegeben.

Während des Betriebs kommt es regelmäßig vor, dass sich die Berechtigungen ändern. Bei Hinzukommen neuer Seiten muss durch das oben genannte Konzept keine Aktualisierung des *Border Control Cache* stattfinden. Bei Änderungen an existierenden Seiten wird zunächst der Beschleuniger darüber informiert, sodass dessen Speicherverwaltung die entsprechenden Pufferspeicher-Einträge als ungültig markiert und eventuelle Inhalte in den Speicher zurückgeschrieben werden. Einträge in den Pufferspeichern, welche übersetzte Adressen enthalten, müssen ebenfalls invalidiert werden, damit diese bei zukünftigen Zugriffen neu abgefragt werden werden. Danach wird die *Protection Table* angepasst und der *Border Control Cache* aktualisiert. Selbst wenn ein Beschleuniger seine eigene Speicherverwaltung nicht aktualisiert, ist der Schutz gewährleistet, da ungültige Schreibzugriffe von *Border Control* abgefangen werden.

Bei Beendigung eines Prozesses sollen, um Datenverlust zu vermeiden, alle Pufferspeicher der Beschleuniger und der *Border Control Cache* geleert werden, damit die enthaltenen Daten zurückgeschrieben werden und die Zugriffsrechte des Beschleunigers zurückgesetzt werden.

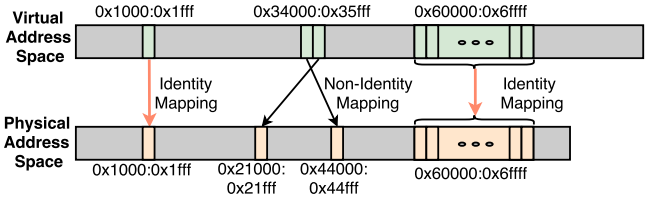

Abbildung 3. Beispiel für das Layout eines Prozess-Adressraums mit *Identity Mapped* sowie regulär allozierten virtuellen Speicherseiten. [7, Figure 3].

Weiterhin kann der Speicher der *Protection Table* vom Betriebssystem wieder freigegeben werden.

4.1.3 Evaluation

Das Hauptargument für die Verwendung von *Border Control* ist, dass Beschleuniger dadurch, dass sie selbst Strukturen der Speicherverwaltung implementieren können, große Freiheiten beim Speicherzugriff gewinnen. Es können entsprechend angepasste Zwischenspeicher eingesetzt werden, welche zu ihren Zugriffsmustern passen. Dies ist allerdings auch ein großer Nachteil. Beschleuniger werden komplexer, was die Fehleranfälligkeit erhöht, und werden zwangsläufig teurer, da zusätzliche Komponenten integriert werden müssen.

Die *Protection Tables* belegen zusätzlichen Hauptspeicher neben den ohnehin existierenden Seitentabellen, welche zur Adressübersetzung notwendig sind. Es entsteht also zusätzlicher Mehraufwand im Hauptspeicher. Aufgrund der geringen Größe wird argumentiert, dass dies zu vernachlässigen sei.

4.2 Devirtualisierter Speicher nach Haria et al.

Einen anderen Ansatz verfolgt das Papier *Devirtualizing Memory in Heterogenous Systems* [7]. Die Autoren schlagen ein neues Konzept des *Devirtualisierten Speichers* (engl. *Devirtualized Memory (DVM)*) vor, bei welchem versucht wird, physischen Speicher mittels seiner physischen Adresse direkt nutzen zu können. Speicher wird also so alloziert, dass die Adressen von Speicherbereichen im virtuellen Adressraum eines Beschleunigers oder Prozesses deren physischen Adressen im Hauptspeicher entspricht. Dieses Konzept wird als *Identity Mapping* bezeichnet.

Abb. 3 zeigt ein Beispiel für Layout des Adressbereichs eines Prozesses mit sowohl *Identity Mapped* als auch regulären virtuellen Speicherseiten. *Devirtualized Memory* stellt eine Erweiterung des Konzepts des virtuellen Speichers dar. Die konventionelle Adressübersetzung wird weiterhin unterstützt. Nach Möglichkeit wird jedoch das effizientere *Identity Mapping* eingesetzt, bei welchem größere Speicherbereiche zusammen alloziert und verwaltet werden. Die virtuellen Adressen der *Identity Mapped* Bereiche entsprechen den physischen Adressen.

4.2.1 Komponenten

DVM besteht aus zwei Komponenten. Im Betriebssystem wird eine neue Form der Seitentabelle, die *Compact Page Table*, eingeführt, welche zur Verwaltung der *Identity Mapped* Speicherbereiche eingesetzt wird. Auf Hardwareseite wird ein neuer Pufferspeicher eingeführt, welcher die Pufferspeicherkomponenten der *IOMMU* in sich vereint.

Compact Page Table Die Seitentabelle wird um einen neuen Typ für Blatteinträge erweitert, welcher *Permissions Entry (PE)* genannt wird. Diese ersetzen Einträge auf allen Ebenen der Seitentabelle. Sie beinhalten 16 Berechtigungseinträge, welche jeweils 2 Bit umfassen und 4 verschiedene Berechtigungszustände kodieren. Je nachdem, auf welcher Ebene in der Seitentabelle sie eingesetzt werden, werden verschieden große Speicherbereiche verwaltet. Je höher die Ebene, desto größer werden die verwalteten Bereiche. Die Größe der *Identity Mapped* Bereiche sind variabel, da mehrere kleinere Felder zu einem größeren Bereich kombiniert werden können. Durch die zusammenhängende kontinuierliche Allokation der Speicherbereiche wird garantiert, dass alle Speicherbereiche, die die *PEs* adressieren, *Identity Mapped* sind. Da sie jeden Knoten innerhalb einer hierarchischen Seitentabelle ersetzen können, können sie ganze Unterbäume einsparen und somit die Größe einer Seitentabelle signifikant reduzieren. Dadurch werden große Mengen an Hauptspeicher eingespart, was bei heute üblichen Speichergrößen allerdings oft zu vernachlässigen ist.

Access Validation Cache Durch die kompaktere Darstellung von Berechtigungen können Einträge der Seitentabelle effizienter zwischengespeichert werden. *DVM* führt daher einen neuen Pufferspeicher-Typ ein. Der *Access Validation Cache* ersetzt die verschiedenen unabhängigen Pufferspeicher innerhalb der *IOMMU* und speichert Einträge auf sämtlichen Ebenen der Seitentabelle zwischen. Da durch den Einsatz der *Permission Entries* insgesamt weniger Einträge auf der untersten Ebene benötigt werden, kann selbst ein kleiner *AVC* die Verwaltungsdaten für große Speicherbereiche vorhalten.

4.2.2 Betrieb

Speicheranfragen werden wie gehabt über eine *IOMMU* geführt. Dort reduziert sich allerdings der Aufwand deutlich. Bei Speicheranfragen wird wie gehabt die *Seitentabelle* abgelaufen. Sofern ein *Permissions Entry* angetroffen wird, werden die Berechtigungen geprüft und

der Speicherzugriff entweder zugelassen oder abgewiesen. Sollte ein regulärer Seitentabelleneintrag gefunden werden, werden die Adressen übersetzt und danach die Zugriffskontrolle ausgeführt.

Bei der Durchführung des nötigen Ablaufens der Speicherseiten in der Tabelle wird durch die optimierte und kompakte Seitentabelle die Wahrscheinlichkeit größer, dass alle notwendigen Einträge bereits im *AVC* vorhanden sind. Hierbei kommt zu tragen, dass der *AVC* alle Ebenen einer Seitentabelle zwischenspeichert und nicht, wie beispielsweise bei Verwendung eines *Translation Lookaside Buffers* zusammen mit einem *Page Walk Cache*, mehrere Zwischenspeicher involviert sind. Bei Fehlen von Einträgen im Zwischenspeicher wird die Seitentabelle herangezogen.

4.2.3 Evaluation

Das System bietet verschiedene Vorteile. Durch das Wegfallen von Adressübersetzungen und das effizientere Zwischenspeichern steigt die Ausführungsgeschwindigkeit von Anfragen an den Speicherbus. *Identity Mapped*-Bereiche bieten ferner die sogenannte *Zeiger-ist-ein-Zeiger-Semantik*, das heißt Zeiger können sowohl vom Hauptprozessor als auch von Beschleunigern direkt genutzt werden, da sie einen gemeinsamen Adressraum benutzen, was weitere Übersetzungen überflüssig macht. Zudem sinkt der Speichermehraufwand, da für *Identity Mapping* deutlich weniger Verwaltungsdaten benötigt werden.

Es gibt jedoch auch signifikante Nachteile. Da Speicher für *Identity Mapping* zusammenhängend alloziert werden muss, kommt es wahrscheinlicher zu Speicherfragmentierung. Sollte kein ausreichend großer Bereich mehr frei sein, kann *Identity Mapping* nicht mehr eingesetzt werden, da mehrere unabhängige Bereiche nur über Speichervirtualisierung als zusammenhängend dargestellt werden können. Weiterhin wird *Copy-on-Write* unterstützt, es wird jedoch darauf hingewiesen, dass beim Schreiben die entstehende Kopie nicht mehr direkt adressiert werden kann, weswegen auf normalen virtuellen Speicher zurückgegriffen werden muss.

Identity Mapping wird von *Haria et al.* nur bei *Heap*-Speicheranfragen durchgeführt, da es dort am Einfachsten zu implementieren ist, was die Komplexität im Rahmen der Untersuchung gering hielt, ist theoretisch jedoch auch z.B. für *Stack*-Speicher möglich. Insgesamt ist *DVM* weniger flexibel als klassischer virtueller Speicher. Durch die Einschränkungen beim Speicher-Layout ist *Address Space Layout Randomization* nur bedingt möglich. Die Autoren stellen jedoch insgesamt in Frage, ob dies einen großen Nachteil darstellt, da bereits wirksame Angriffe darauf beschrieben wurden [18]. Tiefergehende Sicherheitsanalysen werden jedoch auf zukünftige Arbeiten vertagt.

Laut *Haria et al.* sind für den praktischen Einsatz nur geringe Änderungen an der *IOMMU* sowie am Betriebssystem notwendig. Die Autoren stellen ferner eine Möglichkeit vor, *DVM* auch für den Hauptprozessor zu implementieren, um die gleichen Vorteile dort auch nutzen zu können.

5 Vergleich der vorgestellten Systeme

Tabelle 1 vergleicht die beiden vorgestellten Systeme *Border Control* und *Devirtualized Memory* mit der *IOMMU* und Speicherdirektzugriff. Es wird der Implementierungsaufwand der Systeme im Vergleich zu einem Standard-Linux-System, wie es heute eingesetzt werden könnte, betrachtet. Zudem werden weitere Kriterien aus den in den Papieren aufgezählten Vor- und Nachteilen der verschiedenen Systeme analysiert. Hierzu zählen die Performanz, Sicherheit und Unterstützung von Speichervirtualisierung. Zuletzt wird betrachtet, wie frei Beschleuniger bei der Implementierung von Speicherzugriffen sind.

5.1 Implementierungsaufwand (System & Betriebssystem

Der Implementierungsaufwand auf Hardware- und Softwareseite wird hier in Relation zum Einsatz einer *IOMMU* auf einem Linux-System angegeben.

Der Aufwand zur Implementierung ist bei *DMA* am Geringsten. Speicherdirektzugriff bildet die Basis aller vorgestellten Systeme. Da bei reinem *DMA* keine Speichervirtualisierung eingesetzt wird, werden keine entsprechenden Einheiten zur Adressübersetzung benötigt. Der Aufwand ist also als gering einzuschätzen.

Border Control benötigt größere Änderungen auf Hardwareebene, da hier komplett neue Strukturen eingeführt werden, weswegen der Implementierungsaufwand als höher eingeschätzt wird.

Die *IOMMU* ist bereits weit verbreitet und wird von allen großen Betriebssysteme unterstützt und steht somit zwischen *DMA* und *Border Control*. Bei *Devirtualized Memory* muss auf Betriebssystemebene die Seitentabelle geringfügig angepasst sowie einige kleinere Änderungen an der *IOMMU* vorgenommen werden. Auf Softwareseite können die Allokations-Schnittstellen transparent angepasst werden, sodass existierende Programme nicht angepasst werden müssen. Aufgrund des geringeren Implementierungsaufwands von *DVM* im Vergleich zu der *IOMMU*, aber des höheren Aufwands verglichen mit reinem *DMA*, wird der Aufwand neutral bewertet.

5.2 Implementierungsaufwand (Beschleuniger)

Reiner *DMA* unterstützt das grundlegende Konzept der Speichervirtualisierung, das die anderen Systeme teilen, nicht. Weitere Komponenten für den Speicherzugriff sind nicht notwendig, die Anfragen werden direkt und ohne

Kriterium	DMA	IOMMU	Border Control	Devirtualized Memory
Implementierungsaufwand (System & Betriebssystem)	✓	–	✗	–
Implementierungsaufwand (Beschleuniger)	–	✓	✗	✓
Sicherheit	✗	✓	✓	✓
Performanz	✓	✗	✓	✓
Unterstützt Speichervirtualisierung	✗	✓	✓	✓
Flexibler Speicherzugriff auf Beschleunigerseite	✓	✗	✓	✗

Tabelle 1. Qualitative Bewertung von *Border Control* und *Devirtualized Memory* anhand einiger ausgewählter Kriterien

Zwischeninstanz mittels der physischen Adressen an den Bus gestellt.

IOMMU und *DVM* erlauben Beschleunigern, Speicherzugriff mittels virtueller Adressen auf dem Speicherbus direkt durchzuführen, da die Adressübersetzung automatisch von der Schnittstelle übernommen wird. Dies ist ein Vorteil gegenüber *Border Control*, welches von Beschleunigern erwartet, die Adressübersetzung vor einem Zugriff selbstständig mittels des bereitgestellten *Adressübersetzungsdienstes* zu übernehmen und Ergebnisse selbstständig zwischenzuspeichern, was zusätzliche Komplexität innerhalb der Geräte selbst erforderlich macht.

5.3 Unterstützung von Speichervirtualisierung

Reiner *DMA* unterstützt Speichervirtualisierung nicht. Zugriff auf den Speicher kann nur durch die Verwendung physischer Adressen stattfinden. Die entsprechenden Zwischeneinheiten, die notwendig wären, sind nicht vorhanden.

IOMMU, *Border Control* und *DVM* unterstützen alle das Konzept der virtuellen Adressräume und unterscheiden sich nur in der Herangehensweise, beispielsweise bei der Allokation von Speicher oder der Implementierung von Zwischenspeichern.

5.4 Sicherheit

Speicherdirektzugriff ist unsicher, da Zwischeneinheiten fehlen, die Zugriffe kontrollieren und gegebenenfalls verhindern könnten. Speichervirtualisierung wird nicht unterstützt, Prozesse und Beschleuniger können auf den gesamten Adressraum unbegrenzt zugreifen.

Die *IOMMU* unterstützt Speichervirtualisierung und somit sämtliche daraus hervorgehenden Sicherheitsaspekte. Zugriffe werden auf ihre Berechtigung hin überprüft und können auch abgelehnt werden.

Border Control und *Devirtualized Memory* versprechen, Zugriffe auf den Speicherbus schneller durchzuführen, bieten aber den gleichen Schutz vor fehlerhaften oder böswilligen Beschleunigern wie die *IOMMU*, indem sie den Zugang zum Speicherbus kontrollieren und nur für zuvor explizit freigegebene Bereiche zulassen.

5.5 Performanz

Da keine Adressübersetzungen oder Zugriffskontrollen durchgeführt werden, kann durch Speicherdirektzugriff im Vergleich zu den anderen Systemen insgesamt ein höherer maximaler Datendurchsatz erzielt werden. Dies ist positiv zu werten.

Bei der *IOMMU* entsteht, wie in Abschnitt 3 gezeigt wird, Mehraufwand durch die Adressübersetzungen und Zugangskontrollen bei jedem Speicherzugriff. Es werden viele Verwaltungsdaten benötigt, und bei jedem Zugriff müssen aufwändige Adressübersetzungen durchgeführt werden, welche sogar zu zusätzlichen Speicherzugriffen führen können, wenn beispielsweise Daten für die Zwischenspeicher aus der Seitentabelle nachgeladen werden müssen. Dies führt zu Performanzeinbußen, welche negativ bewertet werden.

Border Control und *Devirtualized Memory* erhöhen die Effizienz beim Übersetzen von Adressen oder eliminieren diese ganz. Die Zwischenspeicherung von Zugriffsrechten und Adressabbildungen wird durch die Reduzierung der Datenmengen verbessert. Dadurch können teure zusätzliche Zugriffe auf Berechtigungs- und Seitentabellen eingespart werden. Der Mehraufwand zur Laufzeit wird auf ein Minimum reduziert, was sich positiv auf die Performanz auswirkt.

5.6 Flexibler Speicherzugriff auf Beschleunigerseite

Bei *DMA* können Zugriffe auf den Speicherbus unter Berücksichtigung der Vorgaben des Busses frei durchgeführt werden. Es können Zwischenspeicher entsprechend der Spezifikation implementiert werden, und in Größe und Layout frei definiert werden. *Border Control* geht in eine ähnliche Richtung. Dadurch, dass die Adressen selbstständig über eine Schnittstelle übersetzt werden müssen, bleibt es dem Beschleuniger überlassen, wie diese zwischengespeichert werden. Es können zum Beispiel mehrstufige oder verschieden große Pufferspeicher eingesetzt werden, solange sie die Spezifikation des Speicherbusses einhalten. Für Spezialanwendungen, zum Beispiel wenn der Zugriff auf stark fragmentierte Speicherbereiche notwendig ist, kann dies Vorteile haben, da die Pufferspeicher bei entsprechender Struktur die

Adressen besser vorhalten und somit Anfragen an den *Adressübersetzungsdienst* eingespart werden können.

Bei der *IOMMU* und *Devirtualized Memory* müssen Beschleuniger Speicherzugriffsanfragen mittels der ihnen bekannten virtuellen Adressen durchführen. Die Systeme kümmern sich automatisch um die Übersetzung vor dem Zugriff. Es handelt sich also um generische Systeme. Sie können also vom Beschleuniger nicht beeinflusst werden, die bei der Implementierung getroffenen Annahmen sind zu berücksichtigen. Es ist jedoch hinzuzufügen, dass sich diese fehlende Flexibilität unter Einsatz von *Identity Mapping* aber nicht weiter auswirkt, sondern nur zum Beispiel bei starker Fragmentierung und den daraus resultierenden vielen Adressübersetzungen, welche Pufferspeicher überfordern und somit Zugriffe auf andere Speicher zum Nachladen von Daten notwendig machen können.

6 Fazit

Die beiden vorgestellten Ansätze zur Erhöhung der Performanz bei der Verwendung von geteiltem Speicher in *Kernel-Bypassing*-Anwendungen reduzieren deutlich den Mehraufwand, den die etablierte Lösung, die *IOMMU*, benötigt, und erreichen so eine Steigerung der Performanz bei der Kommunikation zwischen Prozess und Beschleuniger.

Border Control und *Devirtualized Memory* sprechen verschiedene Zielgruppen an. Während ersteres größeren Spielraum bei der Implementierung des Speicherzugriffs auf Seiten des Beschleunigers bietet, bietet *Devirtualized Memory* eine generische Schnittstelle zur Anbindung von Beschleunigern und übernimmt die meiste Arbeit.

Die vorgestellten Konzepte sind bisher noch Thema der Forschung und werden noch nicht produktiv eingesetzt. Einige Fragen, insbesondere bei *Devirtualized Memory*, sind noch offen und könnten in der nahen Zukunft Thema der Forschung sein.

Literatur

[1] James H. Clark. 1982. The Geometry Engine: A VLSI Geometry System for Graphics. *Computer Graphics* 16 (July 1982).

[2] Intel Corporation. 1998. *Accelerated Graphics Port Interface Specification.* Technical Report. Intel Corporation.

[3] Intel Corporation. 2014. *Intel Virtualization Technology for Directed I/O, revision 2.3.* Technical Report. Intel Corporation.

[4] NVIDIA Corporation. 2020. *NVIDIA Video Codec SDK.* Technical Report. https://developer.nvidia.com/nvidia-video-codec-sdk

[5] W. E. Ferguson. 1991. Selecting math coprocessors. *IEEE Spectrum* 28, 7 (July 1991), 38–41. https://doi.org/10.1109/6.83469

[6] Mel Gorman. 2004. *Understanding the Linux Virtual Memory Manager.* https://pdos.csail.mit.edu/~sbw/links/gorman_book.pdf

[7] Swapnil Haria, Mark D. Hill, and Michael M. Swift. 2018. Devirtualizing Memory in Heterogeneous Systems *(ASPLOS '18)*. Association for Computing Machinery, New York, NY, USA, 637–650. https://doi.org/10.1145/3173162.3173194

[8] Liang Hu, Xilong Che, and Si-Qing Zheng. 2016. A Closer Look at GPGPU. *ACM Comput. Surv.* 48, 4, Article Article 60 (March 2016), 20 pages. https://doi.org/10.1145/2873053

[9] Advanced Micro Devices Inc. 2011. *AMD I/O Virtualization Technology (IOMMU) Specification, revision 2.00.* Technical Report. Advanced Micro Devices Inc.

[10] Advanced Micro Devices Inc. 2012. *White Paper AMD Unified Video Decoder (UVD).* Technical Report. Advanced Micro Devices Inc. https://web.archive.org/web/20170830001223/https://www.amd.com/Documents/UVD3_whitepaper.pdf

[11] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, and et al. 2017. In-Datacenter Performance Analysis of a Tensor Processing Unit. *SIGARCH Comput. Archit. News* 45, 2 (June 2017), 1–12. https://doi.org/10.1145/3140659.3080246

[12] ARM Ltd. 2013. *ARM System Memory Management Unit Architecture Specification, SMMU architecture, version 2.0.* Technical Report.

[13] T. Nowatzki, V. Gangadhar, N. Ardalani, and K. Sankaralingam. 2017. Stream-dataflow acceleration. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 416–429. https://doi.org/10.1145/3079856.3080255

[14] Akira Nukada, Yasuhiko Ogata, Toshio Endo, and Satoshi Matsuoka. 2008. Bandwidth Intensive 3-D FFT Kernel for GPUs Using CUDA *(SC '08)*. IEEE Press, Article Article 5, 11 pages.

[15] L. E. Olson, J. Power, M. D. Hill, and D. A. Wood. 2015. Border control: Sandboxing accelerators. In *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. 470–481. https://doi.org/10.1145/2830772.2830819

[16] L. E. Olson, S. Sethumadhavan, and M. D. Hill. 2016. Security Implications of Third-Party Accelerators. *IEEE Computer Architecture Letters* 15, 1 (Jan 2016), 50–53. https://doi.org/10.1109/LCA.2015.2445337

[17] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips. 2008. GPU Computing. *Proc. IEEE* 96, 5 (May 2008), 879–899. https://doi.org/10.1109/JPROC.2008.917757

[18] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. 2004. On the Effectiveness of Address-Space Randomization *(CCS '04)*. Association for Computing Machinery, New York, NY, USA, 298–307. https://doi.org/10.1145/1030083.1030124

[19] Andrew S. Tanenbaum. 2002. *Moderne Betriebssysteme.* Pearson Studium.