FPGAs in DBMS

1st Felix Grenzing
Universität Hamburg
Hamburg, Deutschland
felix.grenzing@studium.uni-hamburg.de

***Zusammenfassung*—Abstract here.**
***Index Terms*—FPGAs, DBMS, Hardware Acceleration**

I. Einführung

Datenbanken sind allgegenwärtig und werden universell vervendet. Von kleinen In-Memory Datenbanken, die in Embedded-Systemen laufen, bis hin zu großen verteilten Datenbanksystemen, die in der Cloud betrieben werden. Große Datenbanksysteme müssen eine hohe Anzahl von Anfragen mit hohem Durchsatz verarbeiten, was eine Herausforderung für die Hardware darstellt. CPUs sind die naheliegende Wahl für die Verarbeitung von Datenbankanfragen, da sie flexibel, leistungsstark und allgegenwärtig sind. Besonders durch die Stagnation der CPU Leistung in den letzten Jahren und der Evolution von Datenbankworkloads, kommen jedoch auch andere Hardwareeinheiten wie FPGAs als Beschleuniger in Betracht. Exeplarisch werden in diesem Paper verschiedene Ansätze vorgestellt, wie FPGAs in Datenbanksystemen eingesetzt werden können und welche Herausforderungen sich dabei ergeben. Die vorgestellten Ansätze sind BitWeaving, IBEX, Caribou, DoppioDB und das REGEXP_LIKE Projekt [1] [2].

II. Begriffe

In [3] und [1] werden verschidene Begriffe verwendet, die Erklärung bedürfen.

A. FPGAs

Ein FPGA (Field Programmable Gate Array) ist ein programmierbarer Schaltkreis, der aus einer Matrix von Logikblöcken besteht, die durch programmierbare Verbindungen miteinander verbunden sind. Die Logikblöcke können durch den Nutzer programmiert werden, um beliebige logische Funktionen zu implementieren. Die Verbindungen zwischen den Logikblöcken können ebenfalls programmiert werden, um die Logikblöcke miteinander zu verbinden. FPGAs sind flexibel und können für eine Vielzahl von Anwendungen eingesetzt werden, die von der Signalverarbeitung über die Kryptographie bis hin zur Datenbankbeschleunigung reichen. FPGAs sind jedoch nicht so flexibel wie CPUs, da sie nur eine begrenzte Anzahl von Logikblöcken und Verbindungen haben. FPGAs sind jedoch in der Lage, spezialisierte Algorithmen sehr effizient zu implementieren, da sie in der Lage sind, die Algorithmen in Hardware zu implementieren, was zu einer hohen Parallelität und Geschwindigkeit führt.

B. Einsatzmöglichkeiten

ung basiert hierbei auf Position des Beschleunigers in Bezug auf die Daten und die CPU.

FPGAs können in Datenbanksystemen auf verschiedene Weisen eingesetzt werden und werden daher häufig auf diese Art kategorisiert. Die Einord

Klassischer Weise gibt es On-the-Side Beschleuniger, wobei die CPU die Daten verwaltet und der FPGA über eine Schnittstelle wie PCIe angebunden ist. Die CPU muss bei dieser Einsatzoption die Daten an den FPGA senden und die Ergebnisse wieder empfangen. Diese Architektur wurde häufig auch mit GPUs umgesetzt.

Eine andere Möglichkeit ist die In Datapath Beschleunigung, bei der der FPGA direkt in den Datenpfad eingebunden ist. Der FPGA kann die Daten in Echtzeit verarbeiten und die Ergebnisse mit gleicher Geschwindigkeit an die CPU zurückgeben. Häufig ist das Ziel die Datenmenge zu reduzieren, da nahe der Datenquelle häufig höhere Bandbreiten zur Verfügung stehen. Die Architektur ist somit mit Smart-Storage Lösungen verwandt.

Eine dritte Möglichkeit ist die Koprozessor Architektur, bei der der FPGA als Koprozessor der CPU fungiert. Der FPGA ist auf demselben Chip wie die CPU und hat häufig auch direkten Speicherzugriff. Flaschenhälse durch langsame Schnittstellen können so vermieden werden, was Vorteile gegenüber On-the-Side Beschleunigern bieten kann.

C. Pipelining und Datenparallelität

Zwei wichtige Arten der Parallelität sind die Pipelineparallelität und die Datenparallelität. Pipelineparalleität beschreibt die Aufteilung der Befehlsabarbeitung in mehrere Stufen, die parallel arbeiten. Jede Stufe bearbeitet dabei Teil des Befehls und gibt die Ergebnisse an die nächste Stufe weiter. Ein Beispiel für eine Pipeline ist in Abbildung 1 zu sehen. Die Befehle werden in vier Stufen aufgeteilt, Fetch, Decode, Execute, Write-back. Die Fetch Stufe liest den nächsten Befehl aus dem ROM, Decode dekodiert den Befehl in Registeraddressen, Opcode oder Operanden, Execute führt Berechnungen durch und Write-back schreibt Ergebnisse in Register oder den RAM. Zunächst bearbeitete die Fetch Stufe den ersten Befehl (grün). Sobald dies geschehen ist, wird der Befehl an die nächste Stufe weitergegeben. Die Fetch Stufe kann nun den nächsten Befehl bearbeiten (violet). Die Befehle werden so stufenweise durch die Pipeline geschoben. Während Befehl 4 (rot) von der Fetch Stufe bearbeitet wird, ist Befehl 3 im Decode, Befehl 2 im Execute und Befehl 1 im Write-back.

Es werden somit vier Befehle parallel bearbeitet, was dazu führt, dass kein Teil des Prozessors unbenutzt bleibt. Die Pipeline kann so die Performance des Prozessors verbessern, da mehrere Befehle gleichzeitig bearbeitet werden können.

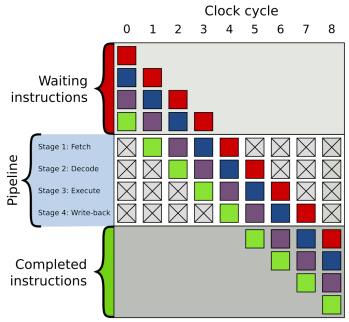

Abbildung 1: Prozessor Pipeline [4]

Datenparallelität beschreibt die Aufteilung der Daten in mehrere Teile, die mit mehreren Bearbeitungseinheiten parallel verarbeitet werden. Die Daten müssen dafür unabhängig von einander sein. Beide Ansätze können die Performance von Algorithmen verbessern, und können auch kombiniert werden um die Vorteile beider Ansätze zu nutzen. Um beide Ansätze zu kombinieren, könnten mehrere Pipelines entworfen werden, die jeweils verschidene Daten bearbeiten.

D. Partial Reconfiguration

Partial Reconfiguration (PR) ist ein Konzept, welches aktuelle FPGAs unterstützen. PR bietet die Möglichkeit, Teile des FPGAs zur Laufzeit zu verändern, ohne das gesamte FPGA neu zu konfigurieren oder das FPGA offline nehmen zu müssen. Dies ermöglicht es, verschiedene Funktionen auf einem FPGA zu implementieren und bei Bedarf zu wechseln. Umkonfiguration während der Laufzeit bietet offensichtlich Pozenzial für Beschleunigung, da mehr Algorithmen implementiert werden können, als gleichzeitig auf dem FPGA Platz haben. Eine Problematik ist jedoch, dass die Umkonfiguration Zeit in Größenordnung von Millisekunden benötigt. Zudem ist die PR-Region nicht dynamisch anpassbar, wodurch bei der Umkonfiguration zu anderen Algorithmen Ressourcen auf dem FPGA verschwendet werden, wenn der Ressourcenaufwand der neuen Algorithmen geringer ist.

Das Projekt DoppioDB [3] nutzt PR, um verschiedene Algorithmen auf einem FPGA zu implementieren und bei Bedarf zu wechseln.

E. BitWeaving

TODO: Faktcheck BitWeaving ist ein Algorithmus, der in [3] vorgestellt wird. Der Algorithmus stellt eine Methode dar, Spaltenscanoperationen durchzuführen, indem die Daten mehrerer Zeilen als Bitvektor codiert in einem Prozessorwort gespeichert und verarbeitet werden. Zwei Varianten des Algorithmus werden vorgestellt, BitWeaving H und BitWeaving V, die sich in der Art und Weise unterscheiden, wie die Daten in den Prozessorworten gespeichert werden. BitWeaving H speichert die Daten zeilenweise in den Prozessorworten, was bedeutet, dass Daten in einem Prozessorwort und nicht über mehrere Prozessorworte hinweg gespeichert werden. BitWeaving V speichert die hingegen Daten spaltenweise in den Prozessorworten. BitWeaving H ist der betrachtete Algorithmus in [1].

BitWeaving stellt ein arithmetisches Framework dar, mit welchem es möglich ist über mehrere Schritte und Bitmasken mit Daten gefüllte Prozessorworte mit einem Prädikatsvektor zu vergleichen. Die Leistung von BitWeaving ist, dass die benötigten Taktzyklen für die Verarbeitung von einer Zeile stark reduziert wird, da in jedem Takt mehrere Zeilen pAro Prozessorwort verarbeitet werden können [3]. Die amortisierte Taktzyklenzahl pro Zeile wird durch BitWeaving zum Teil auf unter 1 reduziert, im Vergleich zu vielen Taktzyklen pro Zeile bei herkömmlichen Methoden.

III. Einordnung der Situation

Die Rolle von FPGAs in Datenbanksystemen ist laut [2] aktuell einem Wandel unterzogen. Während bei FPGAs in der Vergangenheit Herausforderungen die Potenziale überwogen haben, bieten aktuelle Entwicklungen neue Möglichkeiten für den Einsatz von FPGAs in Datenbanksystemen.

[2] stellt diese Entwicklung als Gegensätze von Pessimismus der Vergangenheit und Optimismus der Gegenwart und Zukunft dar.

A. Pessimismus

Der Pessimismus der Vergangenheit begründete sich fundamental in drei Problemen.

Erstens war die Anbindung der On-the-Side Beschleuniger an die CPU ein Flaschenhals, da die Kommunikation in beiden Richtungen über einen Bus erfoglen musste, welcher zu hohe Latenzen und zu geringe Bandbreiten bot. TODO: Quelle On-the-Side Beschleuniger waren die gängige Architektur, um Datenbankoperationen zu beschleunigen, welche nicht im Datenpfad durchgeführt werden können. Koprozessor Architekturen wurden von Chipherstellern nicht wirklich angeboten und waren somit nicht weit verbreitet.

Zweitens profitieren nicht alle Algorithmen von der Beschleunigung durch FPGAs. Iterative Algorithmen, die auf den Ergebnissen vorheriger Iterationen aufbauen, können nicht von der hohen Parallelität von FPGAs profitieren. TODO: Quelle CPUs sind in diesen Fällen performanter, da die hohen Taktraten die itertiven Instruktionen abarbeiten können. Auch weit verzweigte Algorithmen, mit vielen If-Else Anweisungen sind nicht für FPGAs geeignet, da jeder Pfad im FPGA implementiert werden muss, was zu hohen Ressourcenverbrauch führt. Hoher Ressourcenverbrauch wiederrum führt zu geringerer Parallelität, da weniger paralelle Pipelines implementiert werden können. Die Inkompatibilität von Algorithmen wird durch den ersten Punkt noch verstärkt, da die schwierige Kommunikation zwischen CPU und FPGA die Entwickler dazu zwingt, gesamte Algorithmen auf dem FPGA zu implementieren, anstatt nur die Teile, die von der Beschleunigung profitieren, um die Kommunikation zu minimieren.

Drittens ist die direkte Konkurrenz von CPUs ein Hinderniss. CPUs sind sehr flexible, sie können theoretisch jeden Algorithmus ausführen. Hinzu kommt, dass die Leistung der CPUs stetig steigt (Moores's Law) und die Taktraten immer höher werden. Durch den hohe Zeit- und Ressourcenaufwand, welcher die Entwicklung von FPGA Beschleunigern mit sich bringt, war es häufig nicht praktikabel, FPGAs anstelle von CPUs für Datenbankbeschleunigung zu nutzen.

B. Optimismus

Der Optimismus für die Gegenwart und Zukunft, den der Autor motiviert, basiert auf drei wesentlichen Entwicklungen.

Erstens haben sich die Architekturen verändert. Moderne FPGAs sind häufig als Koprozessoren direkt auf dem gleichen Chip wie die CPU integriert, was die Latenzen und Bandbreitenprobleme der On-the-Side Beschleuniger, durch direkten Speicherzugriff des FPGAs, löst. TODO: Quelle Diese enge Integration ermöglicht granulare Beschleunigung, bei der nur die Teile des Algorithmus, die von der Beschleunigung profitieren, auf dem FPGA implementiert werden. Die erhöhte Verfügbarkeit von Koprozessor Architekturen bestärkt nun auch die Erforschung und Entwicklung von FPGA Beschleunigern was zu einer Vielzahl von neuen Anwendungen führt [1] [5].

Ein anschauliches Beispiel ist die Nutzung von FPGAs in Cloud-Umgebungen. Große Anbieter wie Amazon, Baidu und Huawei bieten FPGAs stundenweise an, damit Nutzer individuelle Beschleuniger implementieren können. Microsoft nutzt im Rahmen des Projekts Catapult FPGAs in der Azure-Cloud, um sowohl ihre Infrastruktur als auch maschinelle Lernpipelines zu beschleunigen. Intel experimentiert zudem mit der Integration programmierbarer Elemente direkt in ihre Xeon-CPUs, wodurch spezifische Benutzeranwendungen beschleunigt werden können.

Zweitens haben sich die Workloads verändert. In der Vergangenheit wurden hauptsächlich klassische SQL-Operatoren in Datenbank durchgeführt, welche auf der verfügbaren Hardware häufig Memory-bound waren, also durch die Geschwindigkeit der Datentransfers limitiert. Durch neue Anwendungen wie Maschinelles Lernen und Künstliche Intelligenz sind neue Workloads entstanden, die häufig Compute-bound sind, also durch die Geschwindigkeit der Berechnungen limitiert. Da diese Anwendungen viele Daten benötigen, liegt es nahe, die Anwendungen als neue Datenbank Operatoren einzuführen. Für Compute-bound Workloads werden FPGAs wieder interessant, da sie durch ihre hohe Parallelität und Rekonfigurierbarkeit die Berechnungen beschleunigen können. Auch die Optimierung- und Entscheidungsprozesse innerhalb der Datenbanksysteme könnten durch Maschinelle Lernverfahren verbessert oder abgelöst werden, was ebenfalls Potenziale für FPGAs bietet.

Drittens ermöglichen hybride Ansätze eine bessere Nutzung der Stärken von FPGAs und CPUs. Durch die Kombination beider Technologien können Algorithmen so aufgeteilt werden, dass die parallelisierbaren Teile auf dem FPGA und die sequentiellen Teile auf der CPU ausgeführt werden. Ein Beispiel dafür ist REGEXP_LIKE [5]. TODO: Quelle

IV. Anderes Paper

Im Kontext veränderter Architekturen ordnet sich auch [1] ein. Dort wurde auf einem FPGA ein Column Store implementiert, der den BitWeaving H Algorithmus für einen Spaltenscan verwendet. Es wurden verschidenen Hardwareansätze verglichen und bewertet, um die beste Performance zu erzielen.

A. Architektur

Die Forschung von [1] baut auf der Zynq Ultrascale+ Architektur von Xilinx auf. Die Plattform ist aus zweiteilig aus Steuersystem mit 4 ARM Cortex-A53 Kernen mit 1.2GHz Takt und 4GB DDR4 Speicher und Logikbereich mit FPGA und 512MB DDR4 Speicher aufgebaut [1]. Durch 32-bit – 128-bit Busse kann der Logikbereich auf den Speicher des Steuersystems zugreifen. Diese Busse werden 'direct memory access' (DMA) genannt.

B. Grundlegendes Vorgehen

Die Basis der Implementierung bildet eine Pipeline, die die Daten in mehreren Schritten verarbeitet und die durch den BitWeaving Algorithmus motiviert ist. Die Pipeline besteht aus Lese-Blöcken, Verarbeitungs-Blöcken und Schreibe-Blöcken. Die Lese-Blöcke bekommen sowohl die zu evaluierenden Daten als auch den Prädikatsvektor von einem DMA. Der Prädikatsvektor wird nur zu Begin geladen, da sich dieser nicht verändert. Verarbeitugs-Blöcke führen die eigentliche BitWeaving Operation durch und es gibt so viele Blöcke wie benötigt werden um die Schritte des Algorithmus durchzuführen. Der BitWeaving Gleichheitscheck benötigt beispielsweise 4 Blöcke um die Masken und Verknüpfungen durchzuführen. Die Schreibe-Blöcke schreiben die Ergebnisse zurück in den Speicher des Steuersystems. Alle Blöcke werden pipelinetypisch hintereinander geschaltet, sodass die Daten durch die Pipeline fließen und die Ausgaben des vorherigen Blocks die Eingaben des nächsten Blocks sind.

Um die verschidene Ansätze zu vergleichen, wurde ein Datensatz aus 1 Million Zeilen verwendet. Pro Prozessorwort wurden 16 Zeilen gespeichert. Verglichen wurden in dieser ersten Auswertung die BitWeaving Implementierung auf den ARM Kernen des Steuersystems mit 1 bis 4 Kernen (in Abbildung 2: 'X Cores') und die BitWeaving Implementierung auf dem FPGA mit verschiedenen DMA Busbreiten von 32-bit, 64-bit und 128-bit (in Abbildung 2: 'BASIC X', nach der Breite der verarbeiteten Daten) 1 ARM1 ARM Kern erreichte einen Durchsatz von 1.9 GB/s, 4 Kerne ca. 4.8 GB/s. Der FPGA erreichte mit 32-bit DMA 1.1GB/s, mit 64-bit DMA 1.9GB/s und mit 128-bit DMA 3.9GB/s. Die erste Auswertung zeigt, dass der FPGA mit 128-bit DMA die beste Performance erzielt, was die höchste Datenparallelität innerhalb der Prozessorworte bietet, aber immernoch schlechter abschneidet als 4 ARM Kerne.

C. Verschiedene Ansätze

Um die Parallelität und damit auch den Durchsatz des Systems zu erhöhen, wurde Datenparalell gearbeitet, indem mehrere Pipelines implementiert wurden. Jede Pipeline hat einen eigenen DMA und kann so unabhängig von den anderen Pipelines Daten aus dem Speicher laden. Da bei einem Spaltenscan die Daten unabhängig von einander sind, können so mehrere Zeilen gleichzeitig evaluiert werden. Ansätze mit drei und vier DMAs takten nur mit 200MHz, im Verlgeich zu 250MHz bei zwei oder einem DMA, was erklären wird, warum die Durchsätze für drei und vier DMAs nicht linear zu zwei DMAs skalieren.

Den Ergebnissen aus der ersten Auswertung folgend wurde mit 128-bit DMA gearbeitet, da dieser die beste Performance erzielt hat. Des weiteren wurde einen neue Pipeline Stufe hinzugefügt, welche mehrere Datenworte zu einem längeren Datenwort kombiniert. Diese Combiner-Stufe führt die Pipelines direkt nach den DMA Blöcken zusammen. Die Verarbeitungsblöcke müssen jetzt mit längeren Datenwörtern arbeiten, 256-bit bei zwei DMAs 384-bit bei drei DMAs und 512-bit bei vier DMAs. Durch die erhöhte Menge an gleichzeitig verarbeiteten Daten wurde ein erhöhter Durchsatz erwartet und auch gemessen (In Abbildung 2: 'COMBINED X'). Mit zwei DMAs wurde ein Durchsatz von 7.7 GB/s erreicht, mit drei DMAs 7.5GB/s und mit vier DMAs 6.5GB/s. Zwei DMAs bieten den höchsten Durchsatz, da die DMAs durch die Combiner-Stufe voneinander abhängen, und so gewartet werden muss, bis alle DMAs vom DDR4 Kontroller bedient wurden.

Durch diese Erkenntnis wurde ein weiterer Ansatz (In Abbildung 2: 'INDEP X') eingeführt, bei welchem die Pipelines bis zum Ende getrennt voneinander sind und somit keine Abhängigkeiten zwischen den DMAs bestehen. Der Combiner fällt weg, dafür werden nun 4 paralelle Verarbeitungsblöcke benötigt, welche auf 128-bit Worten arbeiten. Es können auf diese Weise neue Höchstewerte im Durchsatz erzielt werden, 7.8GB/s mit zwei DMAs, 8.9GB/s mit drei DMAs und 8.1GB/s mit vier DMAs. Vier DMAs reduzieren den absoluten Durchsatz, da der DDR4 Kontroller nicht genug Bandbreite für vier DMAs bieten kann und der vierte DMA nicht direkt an den DDR4 Kontroller angeschlossen ist.

Optimale Nutzung von Combiner

Hybridansatz

D. Ergebnisse

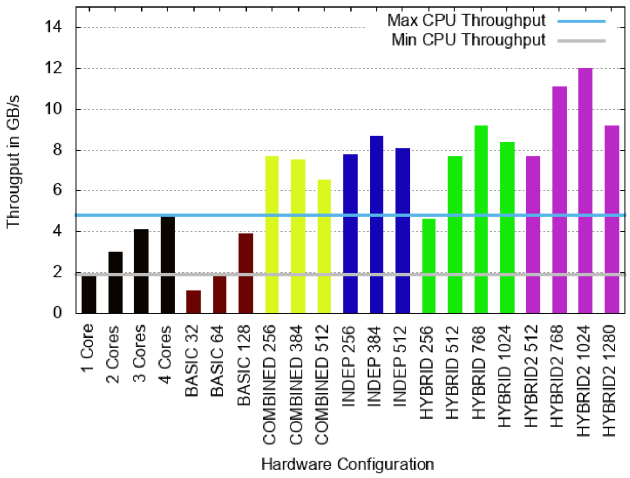

Abbildung 2: Ergebnisse der Evaluation [1]

V. Andere Ansätze

Weitere vorgestellte Ansätze beschäftigen sich mit In-Data-Path Beschleunigern, hybriden Ansätzen und intelligenten Speichersystemen. Diese Ansätze zeigen Beispiele auf, wie FPGAs in Datenbanksystemen eingesetzt werden können und wie die beschriebenen Herausforderungen überwunden werden können.

A. IBEX

IBEX ist ein FPGA-basierter Beschleuniger für Datenbanken, der in [2] vorgestellt wird. IBEX ist ein In-Data-Path Beschleuniger, der direkt in den Datenpfad eingebunden ist, also zwischen SSD und der CPU. Die Daten werden mit gleicher Bandbreite verarbeitet, wie sie von der SSD ankommen. Der FPGA führt SQL-Operatoren aus, welche die Datenmenge reduzieren, also beispielsweise Filter und Aggregationen, nicht aber Operatoren, die die Datenmenge erhöhen können, wie Joins. IBEX verwendet auch einen hybriden Ansatz, um Aggregate zu berechnen. Die Aggregatfunktionen werden auf dem FPGA mit einem Hashtable berechnet, welcher mit fester Größe im RAM verortet ist, was gleiche Laufzeiten, unabhängig von den Daten ermöglicht, da der Hashtable nicht dynamisch vergrößert werden muss, aber dadurch die Anazhl der Gruppen bei Aggregatbildung limitiert. Um dieses Limit zu umgehen, gibt der FPGA Teilaggregate weiter, sobald der Hashtable keinen Platz mehr für neue Gruppen hat. Die CPU vereinigt dann die Teilaggregate zu einem Endergebnis auf eine Weise, das die Bearbeitung von der Arbeit des FPGAs in jedem Fall profitiert. Der FPGA kann so die Datenmenge reduzieren und die CPU kann die reduzierte Datenmenge effizienter verarbeiten.

B. Caribou

Caribou ist ein intelligentes, verteiltes Speichersystem, das speziell für effiziente Datenverarbeitung und -speicherung in modernen Rechenzentren entwickelt wurde. Durch die Integration von Near-Data-Processing wird ein Großteil der Berechnungen direkt an den Speicherknoten durchgeführt, wodurch die Bewegung großer Datenmengen über das Netzwerk minimiert wird. Das System ist auf FPGAs aufgebaut, um eine parallele Verarbeitung und geringe Latenzzeiten sowie eine hohe Durchsatzrate gewährleisten.

Das System nutzt ein einfaches Key-Value-Interface, das über TCP/IP zugänglich ist, und bietet gleichzeitig eine transparente Replikation der Daten, um Fehlertoleranz und Konsistenz sicherzustellen. Caribou zeichnet sich durch ein optimiertes Speichermanagement aus, das auf Kuckuck-Hashtabellen und speicherklassenbasierten Allokationsstrate-

gien basiert, wodurch der Speicherplatz effizient genutzt werden kann, auch bei variablen Datengrößen. Mit Funktionen wie bedingten Abfragen, Scans und Datenkomprimierung bietet Caribou eine flexible Plattform, die nicht nur Daten speichert, sondern auch rechenintensive Aufgaben übernimmt. Caribou ist somit laut [6] eine leistungsstarke, energieeffiziente Lösung für datenintensive Anwendungen in verteilten Systemen. TODO: Faktcheck

C. DoppioDB

Das Projekt DoppioDB zeigt einen solchen Ansatz, indem die Datenbank die FPGA-Steuerung übernimmt und Hardware-Threads nutzt, die wie Software-Threads angesprochen werden. Dabei wurde über SQL hinausgehend Funktionalität für maschinelles Lernen integriert, wie z. B. stochastische Gradientenabstiege und Entscheidungsbaum-Inferenz. TODO: FIXME

D. REGEXP_LIKE

Ein vorgestelltes Projekt befasst sich mit der Optimierung des REGEX_LIKE Operators, welcher reguläre Ausdrücke auf Datenbanken anwendet. So können beispielsweise alle Zeilen einer Tabelle selektiert werden, die in einer Spalte einen bestimmten regulären Ausdruck erfüllen. Der Operator wurde auf einem FPGA implementiert und zeigt die Nutzung von hybriden Ansätzen, indem die relevanten Daten so weit wie möglich auf dem FPGA verarbeitet werden und nur falls nötig auf der CPU nachbearbeitet werden [5]. So können alle REGEXP_LIKE-Anfragen durch die Beschleunigung profitieren, da nicht vollständig passende reguläre Ausdrücke nicht abgewiesen und damit vollständig auf der CPU verarbeitet werden müssen.

Der Reguläre Ausdruck wird an an einer Wildcard-Position aufgeteil, so dass zwei unabhängige reguläre Ausdrücke entstehen. Der FPGA gibt dann, zusammen mit einem Index alle Zeilen zurück, die den ersten regulären Audruck erfüllen. Die CPU überprüft dann, ob die Zeilen vom Index aus auch den zweiten regulären Ausdruck erfüllen. Der verwendete Ausdruck in [5] ist in Listing 1 zu sehen. Der FPGA bearbeitet den Ausdruck bis zur zweiten Wildcard und die CPU muss dann nur noch nach ‘delivery’ suchen.

```
SELECT count(*)
FROM address_tables
WHERE REGEXP_LIKE(address_string,
'(Strasse|Str\.).*(8[0-9]{4}).*delivery');
```

Listing 1: Regulärer Ausdruck aus [5]

VI. The Road that lies ahead

Ein zentraler Aspekt der zukünftigen Datenbankentwicklung ist die Integration programmierbarer Hardware wie FPGAs, die zwar neu konfigurierbar, jedoch nicht so flexibel wie Software sind. Die Kernfrage hierbei ist, wer die Kontrolle über die Beschleunigungsfunktionen übernimmt — das Betriebssystem bzw. der Hypervisor, oder die Datenbank selbst.

A. Betriebssystem- oder Datenbankverwaltung?

Wenn das Betriebssystem die Kontrolle behält, muss die Datenbank Strategien entwickeln, um von hardwarebasierten Beschleunigungsmöglichkeiten zu profitieren, die vom Infrastruktur- oder Cloudanbieter bereitgestellt werden. Hierbei könnten bereits etablierte Techniken zur Code-Optimierung für spezifische CPU-Features, wie SIMD-Einheiten, als Grundlage dienen.

Übernimmt hingegen die Datenbank die Kontrolle, eröffnen sich größere Gestaltungsspielräume. Die Datenbank könnte benutzerdefinierte Hardware-Beschleuniger entwickeln, die exakt auf ihre Bedürfnisse abgestimmt sind, und diese sogar zur Laufzeit synthetisieren (Siehe DoppioDB)

Literatur

[1] N. J. Lisa, A. Ungethüm, D. Habich, W. Lehner, T. D. Nguyen, and A. Kumar, “Column Scan Acceleration in Hybrid CPU-FPGA Systems.” in *ADMS@ VLDB*, 2018, pp. 22–33.

[2] Z. István, “The Glass Half Full: Using Programmable Hardware Accelerators in Analytics.” *IEEE Data Eng. Bull.*, vol. 42, no. 1, pp. 49–60, 2019.

[3] Y. Li and J. M. Patel, “BitWeaving: fast scans for main memory data processing,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’13. New York, NY, USA: Association for Computing Machinery, Jun. 2013, pp. 289–300. [Online]. Available: https://doi.org/10.1145/2463676.2465322

[4] Cburnett, “Generic 4-stage pipeline,” https://en.wikipedia.org/wiki/Instruction_pipelining#/media/File:Pipeline,_4_stage.svg, zugriff: 24-Jan-2025.

[5] D. Sidler, Z. István, M. Owaida, and G. Alonso, “Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures,” in *Proceedings of the 2017 ACM International Conference on Management of Data*, ser. SIGMOD ’17. New York, NY, USA: Association for Computing Machinery, May 2017, pp. 403–415. [Online]. Available: https://doi.org/10.1145/3035918.3035954

[6] Z. István, D. Sidler, and G. Alonso, “Caribou: intelligent distributed storage,” *Proc. VLDB Endow.*, vol. 10, no. 11, pp. 1202–1213, Aug. 2017. [Online]. Available: https://doi.org/10.14778/3137628.3137632