Avrupa Bilim ve Teknoloji Dergisi
Özel Sayı 22, S. 146-151, Ocak 2021

Araştırma Makalesi

www.ejosat.com ISSN:2148-2683

European Journal of Science and Technology
Special Issue 22, pp. 146-151, January 2021

Research Article

Veritabanı Uygulamalarında FPGA Tabanlı Hızlandırıcı Kullanımı

Erdinç Öztürk[1*]

[1*] Sabancı Üniversitesi, Mühendislik ve Doğa Bilimleri Fakültesi, İstanbul, Türkiye, (ORCID: 0000-0003-0456-9213), erdinco@sabanciuniv.edu

(İlk Geliş Tarihi Aralık 2020 ve Kabul Tarihi Ocak 2021)

(**DOI:** 10.31590/ejosat.848995)

ATIF/REFERENCE: Öztürk, E. (2021). Veritabanı Uygulamalarında FPGA Tabanlı Hızlandırıcı Kullanımı. *Avrupa Bilim ve Teknoloji Dergisi*, (22), 146-151.

Öz

Yazılım uygulamalarının performansı, sistem düzeyindeki çeşitli özellikler ile ölçülebilir. Sistem performansı iki ana faktör tarafından belirlenir: veri hesaplama ve hareket kapasitesi. Hesaplama performansı için, merkezi işlem birimi (CPU) mimarisi en önemli faktördür. Bir CPU mimarisi, temel hesaplamaları gerçekleştirmek için aritmetik mantık birimlerini (ALU) kullanır. Bu ALU'ların karmaşıklığı, ilkel aritmetik işlemlerin basamak boyutu ile belirlenir. Bir CPU'nun hesaplama performansı, ALU mimarilerinin performansı ile sınırlıdır. Daha yüksek performans için, CPU çekirdek mimarilerinde artan sayıda ALU kullanılır. Ayrıca, CPU üreticileri, belirli bir sürede gerçekleştirilen işlemlerin sayısını artırmak için çok çekirdekli mimariler tasarlamaktadırlar. Bu iki yaklaşım hesaplama birimlerinin miktarını artırsa da, bu yine de bazı işlem-yoğun uygulamalar için yeterli değildir. Bazı problemler doğası gereği paralelleştirilemezler ve birden çok çekirdek arasında dağıtık olarak çalıştırılmaları zordur. Paralelleştirilebilir sorunlar için bile, çok çekirdekli bir çözüm, çekirdekler arasındaki veri giriş/çıkış sınırlamaları nedeniyle tüm CPU mimarisini yüksek bir yüzdeyle kullanamamaktadır. Veri hareketi için, sistemin bellek hiyearşisinin yapısı baskın faktördür. Farklı düzeydeki önbelleklerin boyutu (L1, L2 ve L3), DDR arayüzünün bant genişliği (DDR4, DDR5, vb.) işlem-yoğun yazılım uygulamalarının performansını belirler. Özellikle veritabanı uygulamaları büyük miktarda veri içerdiğinden, hız için en önemli faktör bellek arayüzü haline gelir. FPGA mimarilerinin hızlandırıcı olarak kullanımı son zamanlarda işlem-yoğun yazılım uygulamalarında popüler hale gelmektedir. Bu makalede, bir FPGA mimarisinde uygulanan basit bir veritabanı sorgulama uygulaması sunulmuştur. Sonuçlar, büyük miktarda veriyle ilgili sorguları içeren veritabanı uygulamaları için önemli bir performans iyileştirmesi elde edilebileceğini göstermektedir.

Anahtar Kelimeler: Veritabanı, FPGA, Hızlandırıcı Tasarımı

FPGA Accelerator Utilization in Database Applications

Abstract

Performance of software applications is determined by various system-level properties. System performance is determined by two major factors: data movement and computational bandwidth. For computational performance, central processing unit (CPU) architecture is the most important factor. A CPU architecture utilizes arithmetic logic units (ALU) to realize basic computations. Complexity of these ALUs is determined by the digit size of primitive arithmetic operations. Computational performance of a CPU is limited by the performance of its ALU architectures. For increased performance, CPU architectures utilize increased number of ALUs in their core architectures. Also, CPU manufacturers build multi-core architectures to increase number of operations realized in a given time. Although these two approaches increase the amount of computational units, this is still not enough for some compute-intensive applications. Some problems are inherently sequential and difficult to schedule amongst multiple cores. Even for parallelizable problems, a multi-core solution is not utilizing the entire CPU architecture with a high percentage, due to data i/o limitations between cores. For data movement, the structure of memory hiearchy of the system is the dominant factor. Size of different level caches (L1, L2 and L3), bandwidth of the DDR interface (DDR4, DDR5, etc.) determine the performance of compute-intensive software applications. Especially for database applications, since there is a large amount of data involved, the most important factor for speed becomes the memory interface. Usage of FPGA architectures as accelerators recently gained attention for compute-intensive software applications. In this paper, a simple database query application implemented on an FPGA architecture is presented. The results show that a significant performance improvement can be achieved for database applications involving queries on large amount of data.

Keywords: Database, FPGA, Accelerator

[*] Sorumlu Yazar: erdinco@sabanciuniv.edu

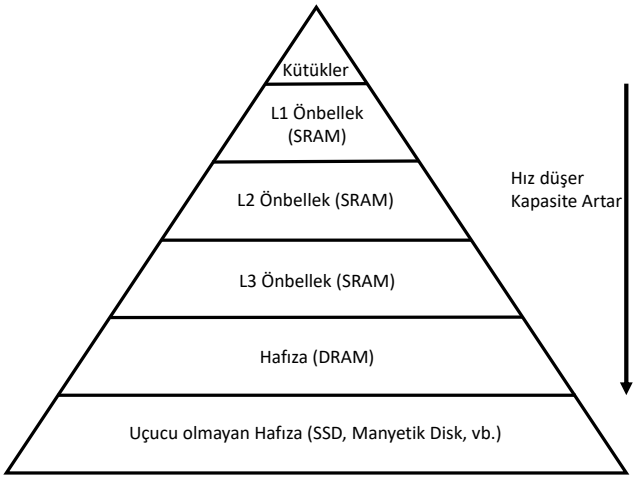
1. Giriş

Modern işlemci (CPU) mimarileri, temelde von Neumann mimarisine dayanmaktadır ve karmaşık algoritmaları gerçekleştirebilmek için temel aritmetik ve mantıksal yapıtaşları içermektedir (Eigenmann & Lilja, 1998). Temel aritmetik yapıtaşları toplama, çıkarma, çarpma ve bölme işlemleri olarak özetlenebilir. Bir CPU nun performansı, bu işlemleri ne kadar hızlı gerçekleştirebildiği ile doğrudan ilişkilidir. Burada hızdan kasıt, birim zamanda mümkün olduğunca fazla işlem yapabilmektir. Modern CPU mimarilerinde aritmetik işlemleri gerçekleştirmek üzere ALU denilen yapılar mevcuttur ve bu yapılar, belli bir basamak boyuna göre tasarlanmıştır (Hennessy & Patterson, 2011). 64 bitlik bir mimaride basamak boyu olarak 2^{64} kullanılmaktadır. Yani, ALU nun içinde 64 bitlik çarpıcı devresi, 64 bitlik toplayıcı devresi, 64 bitlik çıkarma devresi ve 64 bitlik bölme devresi gibi yapılar mevcuttur. Birim zamanda yapılan işlem sayısını arttırmak için işlemci üreticileri kaynak arttırmak yönünde değişik yöntemlere başvurmaktadır. Örnek olarak, modern Intel mimarilerinde her çekirdekte 4 adet ALU bulunmaktadır, her işlemcide de birden fazla çekirdek bulunmaktadır (Hammarlund, ve diğerleri, 2014).

Modern mimarilerde, hafıza hiyerarşisini verimli bir şekilde çalıştırabilmek adına, değişik seviyede değişik boyut ve hızlarda hafıza ve önbellek yapıları bulunmaktadır. Bu hafıza hiyerarşisinin özellikleri, Şekil 1 de özetlenmektedir. Uçucu ve uçucu olmayan hafıza yapılarından oluşan bu hiyerarşinin performansı, bileşenlerine göre değişmektedir (Alpern, Carter, Feig, & Selker, 1994).

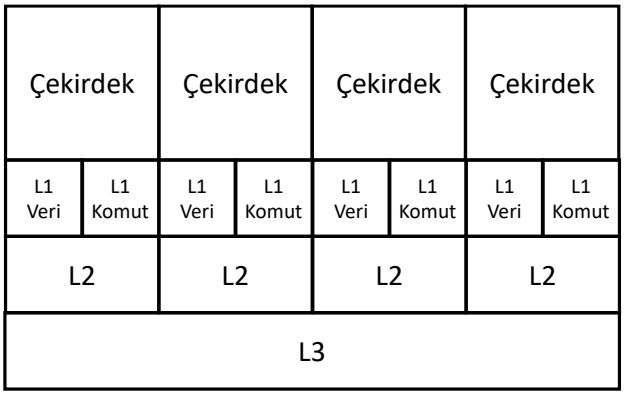
İşlemci üzerinde çalışan programların anlık performansı için en önemli özellik, kullanılan verinin önbellek yapılarında olmasının hızı oldukça arttırıyor olmasıdır. Üzerinde çalışılan veri, eğer L1 seviyesindeki önbellek yapısına sığıyorsa, bundan en yüksek hız verimi elde edilir. Eğer veri büyüyorsa ve L1 önbelleğine sığmıyorsa, işlemler sırasında L2 önbellek seviyesine erişim sağlamak gerekecektir ve bu da hızı düşürecektir. Hafıza hiyerarşisi, bu şekilde çalışmaktadır ve seviye olarak aşağı inildikçe de erişim süresi artacağı için işlemci üzerinde çalışan programların performansı azalacaktır.

Hafıza hiyerarşisini en verimli olarak kullanmanın yöntemi, veriyi L1 önbellek boyutunda parçalara bölerek veri üzerinde yapılacak tüm işlemleri bu parçalar üzerinde çalıştırmaktır. Veritabanı uygulaması özelinde, bir sütun verisi L1 önbellek boyutundaki parçalara bölünür, veri üzerinde çalışacak tüm sorgulamalar gruplanır, ilk parça üzerinde tüm sorgulamalar çalıştırılır, daha sonra ikinci parça üzerinde tüm sorgulamalar çalıştırılır, tüm parçalar bitinceye kadar da bu şekilde devam edilir. Böylelikle verinin hafıza hiyerarşisi üzerindeki hareketi minimize edilmiş olur.



Şekil 1 Hafıza Hiyerarşisi

İşlemci mimarilerinde L1, L2 ve L3 önbellekleri farklı yapılarda bulunmaktadır. Şekil 2 de bir işlemci mimarisinde L1, L2 ve L3 önbellek yapılarının nasıl dağıldığı görülebilir (Molka, Hackenberg, Schöne, & Nagel, 2015).



Şekil 2 İşlemcideki önbellek yapısı örneği

Hafıza hiyerarşisinin zayıf ve güçlü olduğu noktalar mevcuttur. Herhangi bir program, bu hiyerarşinin güçlü olduğu özellikleri kullanabiliyorsa, herhangi bir hızlandırıcı mimarisine ihtiyaç duyulmadığı düşünülebilir. Ancak, bu durumda bile, ok güçlü olan işlemci çekirdeği çok küçük bir hesaplama problemiyle meşgul edilirken, tüm önbellek yapıları dolu olduğu için işlemci başka görevleri hızlı bir şekilde yerine getiremez hale gelebilmektedir. Bu da verinin işlenmesi ile verinin hareket ettirilmesi arasındaki sistemsel dengenin bozulması sonucu oluşmakta, işlemcinin %100 e yakın performansla çalışamamasına yol açmaktadır (Lam, Rothberg, & Wolf, 1991).

Veritabanı uygulamalarında sorgulama işlemlerinde, bir sütun üzerindeki tüm veride sorgulamalar yapılabilmektedir. Sorgulama işlemi oldukça basit olmakla birlikte, sütunun boyutu oldukça büyük olabileceğinde, tüm sütunun hiyerarşinin en alt kısmından en üst kısmına kadar getirilmesi, işlemci üzerinde sorgunun çalıştırılması ve sonucun yine hafızada tutulması gerekmektedir. Burada performans arttırılması için birden fazla sorgunun birleştirilmesi, ve sütun parçaları üzerinde tüm sorguların çalıştırılması yoluna gidilebilir. Ancak, bu yöntem de işlemci için verim sağlayamamaktadır. İşlemciyi ve hafıza hiyerarşisini sadece veritabanı sorguları ile meşgul etmekte, diğer işlemlerin diğer çekirdeklerde bile çalışırken yavaşlamasına yol açmaktadır. Tüm sistem performansı düşünüldüğünde, bu yöntem

pek verimli bir yöntem olmamaktadır (Jayapandian & Jagadish, 2008).

İşlemci ve hafıza hiyerarşisindeki zayıflıklar gözönünde bulundurularak, alternatif olarak bir hızlandırıcı yapısı kullanılabilir. Burada amaç, herhangi bir sorgu çalıştırılacağında, sütundaki tüm veriyi bu hızlandırıcıya göndererek sorgu sonucunu bu hızlandırıcı yapısından geri almaktır. Bu işlem sırasında işlemci yapısı meşgul edilmeyecek, işlemcideki tüm çekirdekler başka görevleri yerine getirebilecektir. Bunun verimli çalışabilmesi için, DMA (Direct Memory Access) denilen yapının kullanılması gerekmektedir. Bu sayede, veri DRAM tabanlı hafıza seviyesinden direk olarak FPGA (*Field Programmable Gate Array*) yapısına aktarılaracak, hafıza hiyerarşisinde herhangi bir etkisi olmayacaktır (Zazo, Lopez-Buedo, Audzevich, & Moore, 2015).

Bu makalede, bu şekilde tasarlanmış bir FPGA hızlandırıcı altyapısı sunulmaktadır. Bu FPGA hızlandırıcı altyapısının çalışma hızının gösterilebilmesi için örnek olarak basit bir veritabanı sorgu yöntemi kurgulanmış, bu yöntemin performans konusundaki bilgileri de sonuçlar bölümünde sunulmuştur.

2. Materyal ve Metot

2.1. Veritabanı Yapıları ve Sorgu Uygulamaları

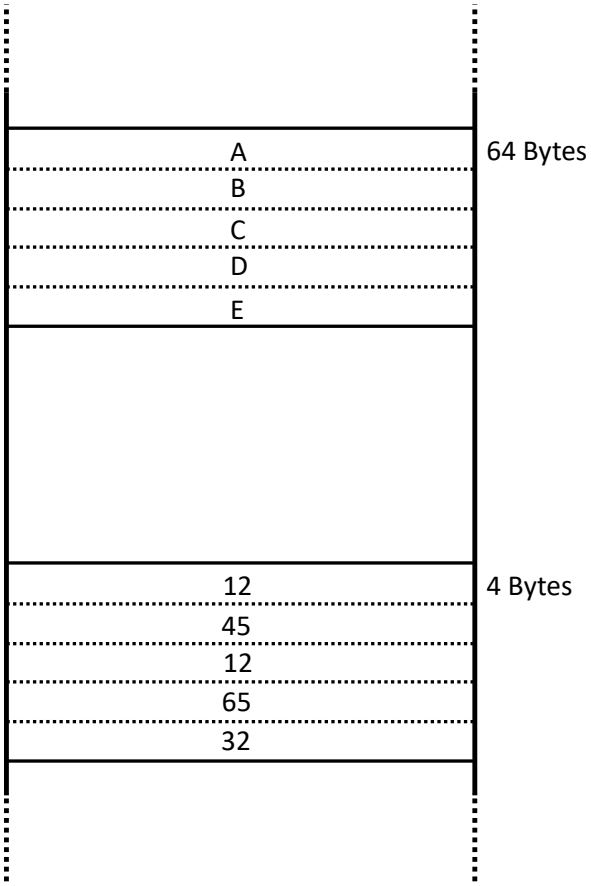
Veritabanı sistemlerinde oldukça karmaşık yapılar kullanılabilmektedir (Gürbüz & Akçin, 2020) (Kurt & İnce, 2020) . Bu yapıların amacı, veritabanına ait olan verilerin verimli bir şekilde saklanarak yine verimli bir şekilde üzerinde sorgu işlemlerinin yapılabiliyor olmasıdır (Kaymaz & ÖZTÜRK, 2020). Veritabanı sistemlerinin karmaşıklığından bağımsız olarak, veri yapıları oldukça basit bir halde düşünülebilir. Bu yapılarda veri, sütunlar halinde saklanmaktadır. Örnek olarak, veritabanı sisteminin herhangi bir bölgesinde yaş bilgisi mevcutsa, ve yaş bilgisi 32 bitlik tamsayı olarak saklandıysa, hafızada bu bilgiler ardışık olarak saklanmaktadır (Navathe & Elmasri, 2001).

Örnek olarak, basit bir veritabanı sisteminde Şekil 3 de görüldüğü gibi veriler olduğunu varsayalım. İsim, 64 Byte lık karakter dizisi olsun.

ID	İsim	Yaş
1	A	12
2	B	45
3	C	12
4	D	65
5	E	32

Şekil 3 Basit bir veri yapısı

Bu veri yapısının sütunları, hafızada Şekil 4 de görüldüğü şekilde tutulmaktadır.



Şekil 4 Veritabanı Sütunlarının Hafızada Saklanması

Bu sütünlara yönelik herhangi bir sorgu yapıldığında, sütundaki tüm verilerin üzerinden geçerek sorgu sonucu elde edilebilir. Sütun boyutları büyüdüğünde, bu işlem oldukça külfetli olabilmektedir. Sorgu, işlem-yoğun bir operasyon olmamakla birlikte, hafızaya oldukça fazla yük getirebilmektedir.

2.2. FPGA Mimarisi

FPGA mimarileri, programlanabilir donanımsal yapılardır. Ayrık olarak tanımlanabilecek ve boole cebiri ile ifade edilebilecek her fonksiyon, kaynaklar yeterli olduğu sürece FPGA üzerinde gerçeklenebilir *(Lysaght, Blodget, Mason, Young, & Bridgford, 2006)*. FPGA mimarileri, pek çok farklı alanda işlem gücü sağlamak ve performans iyileştirmek amacıyla kullanılabilir *(Kaygısız, 2020)*. FPGA üzerindeki kaynaklar şu şekilde özetlenebilir *(XILINX, 2016)*:

- *Configurable Logic Block* (CLB): FPGA içindeki temel fonksiyonel yapıtaşlarıdır. Sayısal tasarımda kullanılan hesaplama yapılarının programlanabilir şekilde gerçeklenebileceği bir *Look-up Table* (LUT) yapısı ile hafıza olarak kullanılan bir *Flip-Flop* yapısını içinde barındırmaktadır. Örnek olarak, 3-bit girişli 1-bit çıkışlı bir LUT yapısı, herhangi bir 3 girişli 1 çıkışlı y=F(a,b,c) boole denklemini gerçekleyebilecek şekilde programlanabilir. FPGA deki CLB sayısı, ürünler arasında farklılıklar göstermektedir.
- *BRAM:* FPGA içinde hafıza yapıları olarak kullanılabilecek olan, oldukça hızlı bir şekilde erişim

sağlanabilen RAM yapıları mevcuttur. Bu RAM yapılarının kapasitesi, mimariden mimariye değişmektedir. RAM yapılarının kapasitesi, FPGA in hafıza-yoğun işlemler cinsinden performansını doğrudan etkilemektedir.

2.3. Hızlandırıcı Mimarisi

FPGA mimarileri, yüksek hızlı bir bağlantı yardımı ile, işlemcilere hızlandırıcı olarak destek olacak yapılar halinde kullanılabilirler. Yüksek hızlı bağlantı seçenekleri arasında en uygunu, PCIe bağlantısı olarak görülmektedir. Sistemde anakarta direk olarak PCIe bağlantısı üzerinden bağlanacak bir FPGA kartı, tasarlanacak olan sürücüler yardımı ile direk olarak sistem ile iletişim kurabilir. Bu yapı için tasarlanmış olan sistem, Şekil 5 de görülebilir. Bu yapı için, (Mert, Öztürk, & Savaş, 2019) deki çalışmalar temel olarak alınmıştır.

Hızlandırıcı mimarisinin çalışma prensipleri, şu şekilde özetlenebilir:

Bilgisayardan FPGA e veri transferi

Hafızada bulunan veri, PCIe bağlantısı üzerinden FPGA e iletilebilir. Bunun için kullanılan fonksiyon

int to_fpga(unsigned char *A, int length)

şeklinde tanımlanmıştır. Burada sütunun pointer değeri ve Byte cinsinden tüm sütunun boyutu, fonksiyone girdi olarak verilmektedir. Verinin transferi tamamen DMA üzerinden yapılmakta, bu sayede veri direk olarak hafıza üzerinden FPGA e gönderilmektedir. CPU nun önbellek yapısı bozulmadığı için, işlemci diğer programlar üzerinde çalışmaya sorunsuz olarak devam edebilir.

FPGA den bilgisayara veri transferi

FPGA de bulunan veri, PCIe bağlantısı üzerinden bilgisayara iletilebilir. Bunun için kullanılan fonksiyon

int from_fpga(unsigned char *A, int length)

şeklinde tanımlanmıştır. Burada FPGA den gelen veri, hafızada A adresine saklanır. Gelecek olan verinin boyutu da bilinmelidir, aksi takdirde PCIe bağlantısı sorunlu olarak çalışmaktadır.

Hızlandırıcı mimarisi, duplex olarak çalıştırılmıştır. Veritabanı uygulamaları için bu elzem bir özelliktir. FPGA üzerindeki hafıza kaynakları kısıtlı olduğu için, büyük ölçekli veritabanı uygulamalarında tüm sütunun FPGA içindeki hafızaya sığması mümkün değildir. Veriyi gönderip FPGA de işleyip sonucu geri almak, uygulanabilecek bir yöntem değildir. Bunun yerine, bir yandan veri gönderilirken, aynı anda FPGA üzerinde işlenerek yine simültane olarak sonuçlar FPGA den bilgisayara gönderilebilecektir. Duplex bir bağlantı olmadan bu mümkün değildir ve FPGA hızlandırıcı yapısının bu özellik olmadan verimli çalışması mümkün değildir.

2.4. Veritabanı Hızlandırıcı Uygulaması

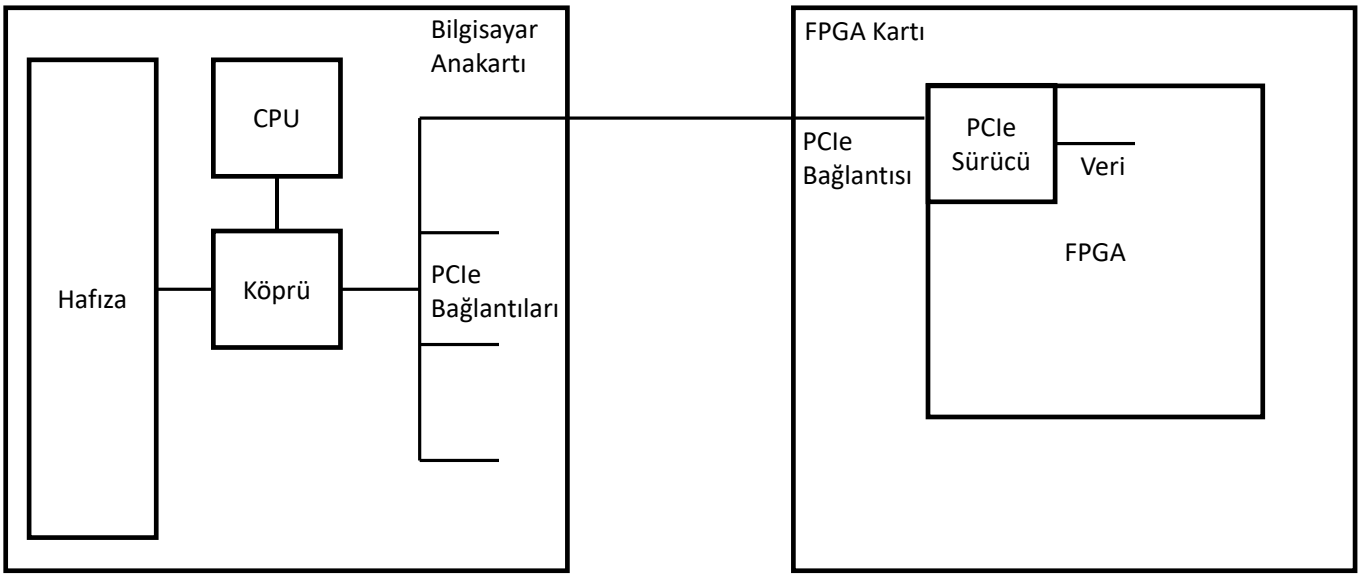
Tasarlanmış ve duplex olarak gerçeklenmiş fpga fonksiyonları, uygun sırayla çağırılarak sütun veya sütunlar üzerinde sorgu işlemleri gerçeklenebilir. Yapılacak olan deneyin basit tutulması amacıyla, yapılacak olan sorgu sütun üzerindeki değerleri aramak ile sınırlandırılmıştır. Örnek olarak, yaş sütununda yaşı 45 e eşit olan kaç kişi vardır sorgusu FPGA tarafından gerçeklenmektedir. Burada, yaşı 45 den büyük olan kaç kişi vardır sorgusu da FPGA de yapılacak sorgulara kolaylıkla eklenebilir. FPGA mimarisi, bu şekilde genişleyebilecek şekilde esnek olarak tasarlanmıştır.

Birden fazla sütun üzerinde de sorgu gerçeklenerek, tüm sorgulara pozitif dönen elemanlar sonuç olarak dönülebilir. Örnek olarak, yaşı 45 olan ve adı B olan kaç kişi vardır? Sorusunun cevabı, FPGA tarafından bulunabilir. Burada dikkat edilmesi gereken konu, her sorgu tipinin ayrı ayrı FPGA de tanıtılması gerektiğidir. Ancak, yine esnek bir mimariye sahip olan tasarımında bu da oldukça kolay olacaktır.

*A adresinde bulunan ve 100 elemanı olan, 32 bitlik tamsayı değerlerinden oluşan sütun için yazılan kod şu şekilde özetlenebilir:

- Sorgu türü, sorgunun içeriği gibi bilgileri hafızada *sorgu adresine yaz
- to_fpga(sorgu, sorgu_boyutu) şeklinde fonksiyonunu başlat
- to_fpga(A, 32*100) şeklinde fonksiyonu yeni bir thread olarak başlat
- from_fpga(sonuc, sonuc_boyutu) şeklinde fonksiyonu yeni bir thread olarak başlat

Sorgu işlemi bittiğinde, sorgunun sonucu *sonuc adresine yazılacaktır.



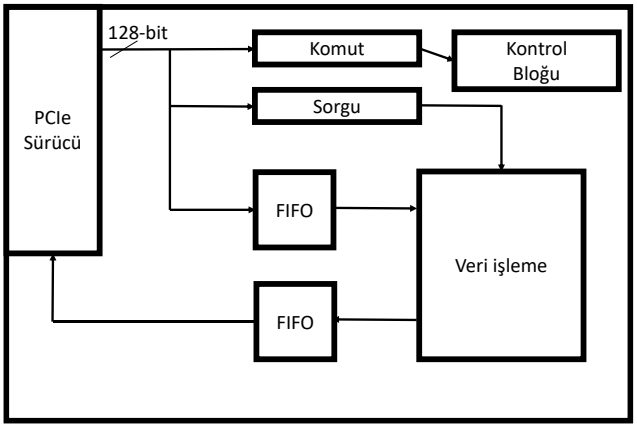
Şekil 5 Hızlandırıcı Destekli Sistem Tasarımı

Tablo 1. Deney Sonuçları

Sütun Eleman Sayısı	Sütun Veri Türü	Sadece CPU zamanlama (cc)	Hızlandırıcı ile zamanlama (cc)
16,777,216	Unsigned char	158,563,828	20,634,364

2.5. FPGA Tasarımı

FPGA e gelen verinin, FPGA tarafından işlenmesi gerekmektedir. Bu işlem için, FPGA in uygun şekilde programlanması gerekmektedir. FPGA programlanması için tasarlanan ve gereklenen mimari Şekil 6 da özetlenmiştir. Veri işleme devresi, unsigned char için özel olarak tasarlanmıştır.



Şekil 6 FPGA mimarisi

FPGA tasarımı, duplex çalışmaya uygun olarak yapılmıştır. Veri işleme modülünün girdi ve çıktılarının FIFO üzerinden yönetiliyor olması, bu yüzdendir. Bilgisayarın hafızasından gelen veri önce FIFO ya koyulacak, daha sonra veri işleme ünitesi tarafından FIFO dan alınacaktır. Komut kütüğündeki bilgiler, FIFO dan alınan verinin özelliğini belirlemektedir. Veri işleme modülü, bu bilgileri kullanarak çalışmakta, kontrol bloğu sayesinde bir veya birden fazla sorgu yapmakta ve cevapları çıkış FIFO suna göndermektedir. Daha sonra çıkış FIFO sundaki veri, PCIe bağlantısı üzerinden bilgisayara gönderilebilir. Burada PCIe sürücüsü hem girdi hem de çıktı işlemlerini paralel olarak gerçekleştirebilir.

3. Araştırma Sonuçları ve Tartışma

3.1. Gerçekleme

Tüm sistem tasarımı ve gerçeklenmesinde kullanılan sistem elemanları, şu şekilde listelenebilir:

- Intel i9-7900X işlemci: 3.3 GHz (Turbo ile 4.3 GHz), 13.75 MB L3 Cache
- 32GB Hafıza: DDR4, 2400MT/s
- XILINX VC707 board
    - XC7VX485T-2FFG1761 FPGA
    - 37,080 Kb BRAMs
    - 75900 CLBs
- RIFFA sürücü (Jacobsen, Richmond, Hogains, & Kastner, 2015): PCIe bağlantısı için RIFFA sürücüsü temel olarak alınmış, veritabanı uygulamalarına uygun olarak modifiye edilmiştir.

Örnek zamanlama için yapılan test sonuçları, Tablo 1 de özetlenmiştir. Yapılan testte, 16,777,216 (16*1024*1024) elemanı olan bir Byte sütunu üzerinde, rastgele değerler aranmıştır. Aynı sorgu, 100 defa çalıştırılarak ortalama alınmıştır. Bu işlem farklı sorgu değerleri için tekrarlandığında oldukça yakın değerler ortaya çıktığı görülmüştür. FPGA üzerinde yapılan işlemlerin performansı, veriden bağımsızdır. FPGA üzerinde yapılan işlemler, oldukça hızlı oldukları için, tüm hızlandırıcı

sistem performansı PCIe hızı ile limitlenmektedir. VC707 kartı için teorik PCIe hız limiti, saniyede 4GB olarak belirlenmiştir. Oldukça büyük veri gönderiminde, iletişim hızı olarak yaklaşık saniyede 30Gb gibi hızlara erişilebilmiştir. Ancak, veritabanı uygulamalarında daha küçük boyutlar ile çalışılacağı düşünülerek, zamanlama da ona göre alınmıştır.

4. Sonuç

FPGA tabanlı bir hızlandırıcı tasarlanıp gerçeklenerek, veritabanı uygulamalarında performans arttırmak için kullanılmıştır. Bu çalışmada yapılan demo gerçeklemeler, büyük veritabanı uygulamalarında FPGA tabanlı hızlandırıcıların verimli bir şekilde kullanılabileceğini göstermiştir. Boyutu ne olursa olsun, herhangi bir sütun üzerinde yapılan sorgunun FPGA üzerinde oldukça verimli bir şekilde çalışabileceği ve performansın tamamen PCIe bağlantısı ile sınırlı olduğu görülmüştür. Bir sütun üzerinde yapılacak olan sorguları tek tek yapmak yerine gruplayarak yapmanın, performans açısından oldukça verimli olacağı görülmektedir. Örnek olarak, önce 45 yaşındaki bireyleri, daha sonra 40 yaşındaki bireyleri sorgulamak yerine, bu eğer tek sorguda yapılırsa FPGA üzerinde aynı zamanı alacak ancak aynı birim zamanda 2 farklı sorgu gerçekleneceği için performans iki katına çıkacaktır. Bu, FPGA kaynakları zorlanarak 100 ila 1000 katlık iyileştirmelere yol açabilir.

FPGA mimarisinde tutulabilecek olan veri boyu sınırlı olduğu için, büyük veriler üzerinde çalışılırken, aynı veri üzerinde defalarca işlem yapan sorguları FPGA üzerinde çalıştırmak, verimsiz olacaktır. Dolayısıyla, veriyi parçalara bölüp FPGA mimarisine sığacak kısım üzerinde tüm seri işlemleri çalıştırmak bir yöntem olarak kurgulanabilir. Bu çalışmanın sonucu olarak, veri ne kadar büyük olursa olsun, akan veri üzerinde yapılacak tüm sorguların FPGA üzerinde verimli bir şekilde çalışabileceği görülmüştür. Gelecek FPGA hızlandırıcı çalışmalarında, bu yöntem üzerinde durulmalıdır.

5. Teşekkür

Bu çalışma, TÜBİTAK tarafından 118E044 numaralı araştırma projesi altında desteklenmiştir.

Kaynakça

Alpern, B., Carter, L., Feig, E., & Selker, T. (1994). The uniform memory hierarchy model of computation. Algorithmica, 72.

Eigenmann, R., & Lilja, D. J. (1998). Von neumann computers. Wiley Encyclopedia of Electrical and Electronics Engineering, 23, 387-400.

Gürbüz, M. Z., & Akçin, O. (2020). Optimizasyon Otomasyonu İçin Cplex Çözücüsünün Web Tabanlı Sunucu Üzerinden Çalıştırılması. Avrupa Bilim ve Teknoloji Dergisi, 943-951.

Hammarlund, P., Martinez, A. J., Bajwa, A. A., Hill, D. L., Hallnor, E., Jiang, H., . . . Osborne, R. (2014). Hammarlund, Per, et al. "Haswell: The fourth-generation intel core processor. IEEE Micro, 6-20.

Hennessy, J. L., & Patterson, D. A. (2011). Computer architecture: a quantitative approach. Elsevier.

Jacobsen, M., Richmond, D., Hogains, M., & Kastner, R. (2015). RIFFA 2.1: A reusable integration framework for FPGA accelerators. ACM Transactions on Reconfigurable Technology and Systems (TRETS) 8, 1-23.

Jayapandian, M., & Jagadish, H. V. (2008). Automated creation of a forms-based database query interface. Proceedings of the VLDB Endowment 1.1, (s. 695-709).

Kaygısız, H. (2020). FPGA Kullanılarak Görüntülerin Gerçek Zamanlı Olarak OTSU Metodu ile Bölütlenmesi. Avrupa Bilim ve Teknoloji Dergisi, 911-917.

Kaymaz, M. E., & ÖZTÜRK, Ö. (2020). ORCA: GO Programlama Dili için ORM/ODM Kütüphanesi. Avrupa Bilim ve Teknoloji Dergisi, 310-317.

Kurt, G., & İnce, G. (2020). ARgent: Web Tabanlı Dinamik İçerik Destekli Artırılmış Gerçeklik Geliştirme Altyapısı. Avrupa Bilim ve Teknoloji Dergisi, EJOSAT, 244-257.

Lam, M. D., Rothberg, E. E., & Wolf, M. E. (1991). The cache performance and optimizations of blocked algorithms. ACM SIGOPS Operating Systems Review, 63-74.

Lysaght, P., Blodget, B., Mason, J., Young, J., & Bridgford, B. (2006). Enhanced architectures, design methodologies and CAD tools for dynamic reconfiguration of Xilinx FPGAs. In 2006 International Conference on Field Programmable Logic and Applications (s. 1-6). IEEE.

Mert, A. C., Öztürk, E., & Savaş, E. (2019). Design and implementation of encryption/decryption architectures for bfv homomorphic encryption scheme. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, 353-362.

Molka, D., Hackenberg, D., Schöne, R., & Nagel, W. E. (2015). Cache coherence protocol and memory performance of the intel haswell-ep architecture. 44th International Conference on Parallel Processing (s. 739-748). IEEE.

Navathe, S. B., & Elmasri, R. A. (2001). Fundamentals of Database Systems with Cdrom and Book. Addison-Wesley Longman Publishing Co., Inc.

XILINX. (2016). 7 Series FPGAs Configurable Logic Block - User Guide.

Zazo, J. F., Lopez-Buedo, S., Audzevich, Y., & Moore, A. W. (2015). A PCIe DMA engine to support the virtualization of 40 Gbps FPGA-accelerated network appliances. In 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig) (s. 1-6). IEEE.