Donanım Hızlandırıcılı Bulut Bilişim Sistemleri için Yonga-üstü Anahtar Mimarisi
An On-chip Switch Architecture for Hardware Accelerated Cloud Computing Systems

Fatih Yazıcı[†], Ayhan Sefa Yıldız[*†], Alper Yazar[*†], Ece Güran Schmidt[†]

[†] Elektrik Elektronik Mühendisliği Bölümü, ODTÜ, Ankara, Türkiye
{fatih.yazici, yildiz.ayhan, alper.yazar, eguran}@metu.edu.tr
[*] ASELSAN A.Ş., Ankara, Türkiye
{asyildiz, ayazar}@aselsan.com.tr

***Özetçe* —Bu bildiride, donanım hızlandırıcılı bulut bilişim sistemleri için ölçeklenebilir bir yonga-üstü paket anahtarı mimarisi önerilmektedir. Mimarimiz FPGA üzerinde yeniden yapılandırılabilir bölgeleri, 40 Gbps Ethernet arayüzlerini ve bir PCIe arayüzünü birbirine bağlamaktadır. Anahtar örgüsü, ölçeklenebilirliği sağlamak için hat hızında çalışmaktadır. Önerdiğimiz yeni bir algoritmayla giriş çıkış hat çiftlerinin anahtar örgüsüne erişimi atanan önceliklendirmeye göre sağlanmaktadır. Tasarımımız Xilinx Zynq 7000-SoC üzerinde uygulanmıştır ve 40 Gbps hızında çalışabilmektedir. Simülasyon sonuçlarımız, veri çıkışını düşürmeden bağlantıların önceliklendirilebildiğini göstermektedir.**

***Anahtar Kelimeler*—bulut bilişim, yonga-üstü anahtar, anahtar örgüsü çekişme çözümlemesi.**

***Abstract*—In this paper, we propose a scalable on-chip packet switch architecture for hardware accelerated cloud computing systems. Our proposed switch architecture is implemented on the FPGA and interconnects reconfigurable regions, 40 Gbps Ethernet interfaces and a PCIe interface. The switch fabric operates at line speed to achieve scalability. We propose a new algorithm that grants access to the fabric according to the allocated prioritization to input-output port pairs. The switch is implemented on Xilinx Zynq 7000-SoC and can work at 40 Gbps rate. Our simulation results show that our proposed algorithm achieves desired prioritization without degrading the throughput.**

***Keywords*—cloud computing, on-chip switch, switch fabric arbitration.**

I. Giriş

Günümüz bulut bilişim servislerinde bellek, işlemci (CPU) ve disk gibi kaynakların yanı sıra donanım hızlandırıcılarının (hardware accelerators-HA) da kullanıcılara sunulması akademik ve uygulama alanında ilgi gören bir konudur [1], [2]. Bu kapsamda bulut veri merkezindeki sunuculara FPGA hızlandırıcı (FPGA Accelerator Card-FAC) kartları eklenebilir ya da bir sunucuya bağlı olmadan bu kartları tek başlarına bulut haberleşme ağına bağlı kullanarak donanım hızlandırıcı çalıştırılabilir. Donanım hızlandırıcılar programlanmak üzere kullanıcıya sunulabildiği gibi, bulut yöneticisi tarafından kullanıcıdan gelen işlerin istenen başarımda çalıştırılabilmesi için CPU'ya alternatif olarak kullanılabilir [3].

Kısmi yeniden yapılandırma (partial reconfiguration) ile FPGA üzerinde birden fazla Yeniden Yapılandırılabilir Bölge (YYB) tanımlanarak donanım kaynakları farklı öğe boylarında atanabilir. Bu kullanım şeklinde FPGA üzerindeki arayüzlerin, işlemci benzeri bileşenlerin ve YYBlerin birbirleri ile haberleşmeleri gerekmektedir.

Bu bildiride, bulut bilişim sistemlerinde kullanılacak FAC'ler için FPGA üzerinde gerçeklenen bileşenler arasında haberleşmeyi sağlayacak *ölçeklenebilir* bir *yonga-üstü paket anahtar* tasarımı, gerçekleştirilmesi ve başarım değerlendirilmesi sunulmaktadır. Donanım gerçekleştirimi Xilinx Zynq-7000 SoC XC7Z100 ürünü [4] üzerindedir. Bildiride sunduğumuz yeni anahtar örgüsü çekişme çözümleme (switch fabric arbitration) yöntemi, CreditArbiteR (CAR), bileşenler arasında *anahtarlama servisi farklılaşması* (switching service differentiation) sağlayarak önceliklendirilmiş haberleşme sağlamaktadır. Anahtar başarımı farklı yük senaryoları altında yazılım simülatörü ile ölçülmüştür.

II. Önceki Çalışmalar

Hızlı geniş alan bilgisayar ağlarında anahtarlama paketlerin sabit boyutlu hücrelere bölünmesi ile gerçekleştirilir. Yüksek sayılarda hatlara ve hızlara ölçeklenme, anahtar örgüsünün hat hızlarında çalıştırılması ile sağlanır. Anahtar giriş ve çıkış hatlarının yüksek veri çıkışını (throughput) sağlayacak şekilde bire bir bağlantısı için çekişme çözümleme gereklidir. [5], [6] yöntemlerinde giriş ve çıkışlar arasında istek/onay mesajları göndererek, [7] mesajlarda giriş kuyruklarındaki paket sayılarını da göndererek veri çıkışını artırmaktadır. [8]'de giriş-çıkış eşlemesi ağırlıklı yapılarak bağlantılara bant genişliği ayarlaması yapmaktadır. Bu çalışma (iş koruyan - work-conserving) bir yaklaşım değildir ve kapasite mevcut olsa bile ayırılan bant genişliğinin üzerine çıkılmamaktadır.

Yonga-üstü anahtarlar örgü (mesh) topolojisi ile bağlı çok

978-1-7281-7206-4/20/$31.00

sayıda haberleşme düğümü için az sayıda hatla tasarlanmaktadır. Ağ anahtarlarından farklı olarak kontrol ve anahtarlama bir saat çeviriminde taşınan veri miktarı olan *flit* ölçeğinde yapılmakta ve sanal kanalların (virtual channel-VC) bağlanması ile gerçekleştirilmektedir. [9], [10] çalışmalarında hatlara öncelik veren yonga-üstü asenkron anahtar yapıları önerilmektedir. Gerçekleme devre tasarımı şeklindedir. [11], arabellek yönetimini bağlantılı-liste olarak özel bir yazmaç yapısı ile gerçekleştirerek sanal çıkış kuyruğu (Virtual Output Queue-VOQ) kullanmadan kuyruk başı bloklanmasını (Head of Line blocking-HOL) aşmaya çalışmakta, bu şekilde gecikme ve veri miktarını iyileştirmeyi amaçlamaktadır. Gerçekleştirim donanım olmadan simülatör yoluyla yapılmıştır.

III. YONGA-ÜSTÜ ANAHTAR

Öngördüğümüz yapıda FAC, 40 Gbps Ethernet ara yüzü ile hem sunucunun hem de YYBlerin diğer bulut sunucuları ve genel ağ ile haberleşmesini gerçekleştirmektedir. Buna göre anahtar $N = 8$ hatlı olarak tasarlanmıştır. 40 Gbps genel ağ arayüzü, 40 Gbps sunucu-FAC arayüzü, 4 YYB, SoC işlemci ve PCIe arayüzlerini haberleştirmektedir. 40 Gbps ağ arayüzleri Ethernet (IP Çekirdeği [12]) olarak gerçeklenmiştir. Bütün hatlar 40 Gbps hızda çalışacak şekilde gerçeklenmiştir.

A. Blok Mimari ve Donanım Gerçeklemesi

Anahtarın blok mimarisi Şekil 1'de görülmektedir. Giriş hattı i ve çıkış hattı j için blok tasarım detaylandırılmıştır. Anahtar örgüsü ölçeklenebilir şekilde 40 Gbps'te çalışabilmesi için hat hızında çalışacak şekilde tasarlanmıştır. Kuyruk başı bloklanmasını önleyerek veri çıkışını artırmak için her giriş i'de her çıkış j için ayrı bir FIFO yapısı ile sanal çıkış kuyrukları oluşturulmuştur [5], [6]. Bu kuyrukları `FIFO_i_j`, $i, j = 0 \cdots 7$ olarak isimlendirmekteyiz. Her saat çevriminde iletilen veri miktarı 40 Gbps IP çekirdeği arayüzü ile uyumlu olarak seçilmiştir. Buna göre flit boyutumuz 256 bittir. `Anahtar Örgüsü`, bütün girişleri her çıkışa çoklayıcı (multiplexer) ile bağlayarak gerçekleştirilen bir çaprazlayıcı anahtar (crossbar) yapısıdır. `Giriş Kontrolcüsü_i`, giriş hattı i'ye gelen veriyi çıkış hattı j'ye göre `FIFO_i_j`'ye yazar. `Bit Haritası Yaratıcısı_i` FIFO yapıların sağladığı `empty` sinyalini kullanarak, her giriş hattı i'de hangi çıkışlar için hazır veri olduğunu bir bitharitası ile `Çekişme Çözümleyicisi`'ne iletir. Bu modül anahtar örgüsü için bire-bir bağlantı konfigürasyonuna *karar* vererek çoklayıcıların seçim girişlerini (select inputs) ayarlar. Mevcut tasarımdaki çekişme çözümleyicisi (arbiter), literatürdeki İkili-Çevirimsel-Sıralı (Dual Round Robin-DRR) [6] yaklaşımı ile gerçekleştirilerek hem donanım tasarımımızın hem de simülatör yazılımımızın ön doğrulaması amaçlanmıştır. Karar hesaplaması ve örgü üzerinden veri iletimi boru hattı (pipeline) yaklaşımı ile gerçekleştirilmiştir. DRR için karar 4 saat çeviriminde hesaplanmaktadır. `VOQ Kontrolcüsü_i` çekişme çözümleyicisi kararına göre anahtar örgüsünün giriş portuna ilgili sanal giriş kuyruğu FIFO'sunda depolanan flitlerin iletilmesini sağlar.

Hem boru hattı verimini yüksek tutmak hem de bağlantı konfigürasyonunu yeterince sık değiştirerek paket gelişine uygun kararlar verebilmek için anahtarlama 320 baytlık *hücreler* ile yapılmaktadır. Bağlantı konfigürasyonunun hesaplanması ve hücrelerin anahtarlanması için harcanan zamana *anahtar döngüsü* adını vermekteyiz. Bir hücrenin anahtar örgüsü üzerinden taşınması için gerekli saat çevirim sayısını *hücre zamanı* olarak tanımlamaktayız. Mevcut tasarımda boru hattı tasarımı ile bir anahtarlama döngüsü, hücre zamanı olan $320 \cdot 8/256 = 10$ saat çevirimine eşittir. Giriş hattı i'den çıkış hattı j'ye iletilen hücreler yeniden birleştirme (reassembly) `RA Kontrolcüsü_j` tarafından `FIFO_j_i`'de tekrar paket haline getirilerek anahtardan çıkarılır.

TABLO I: CAR İÇİN NOTASYON TABLOSU

KabulTamKredi(i,j)	Kabul işaretçisi i için çıkış j'ye atanan kredi değeri
OnayTamKredi(i,j)	Onay işaretçisi j için giriş i'ye atanan kredi değeri
$T_{IN,i}$, $T_{OUT,j}$	Giriş i ve çıkış j için anahtarlama döngüsü
KabulKredi(i)	Kabul işaretçisi i için kalan güncel kredi miktarı
OnayKredi(j)	Onay işaretçisi j için kalan güncel kredi miktarı
KabulİşaretÇi(i)	Kabul aşamasında giriş i'nin önceliklendirdiği çıkış
OnayİşaretÇi(j)	Onay aşamasında çıkış j'nin önceliklendirdiği giriş
Kabul(i)	Giriş i için anahtarlanmak üzere seçilen çıkış

YYB üzerinde koşan uygulamanın ve üretilen verinin tasarımcı kontrolünde olmasına karşılık diğer arayüzlerden yapılan veri haberleşmesinde böyle bir kontrol olmadığını varsaymaktayız. Buna göre, Bölüm III-C'de sunulan simülatör sonuçlarına ve FPGA donanımın ayrıklı olmasından kaynaklı minimum öğe boylarına uygun olarak YYBlere bağlı hatlardaki her FIFO boyutu 16 flit = 1.6 hücre, diğer hatlarda 512 flit = 51.2 hücre olarak belirlenmiştir.

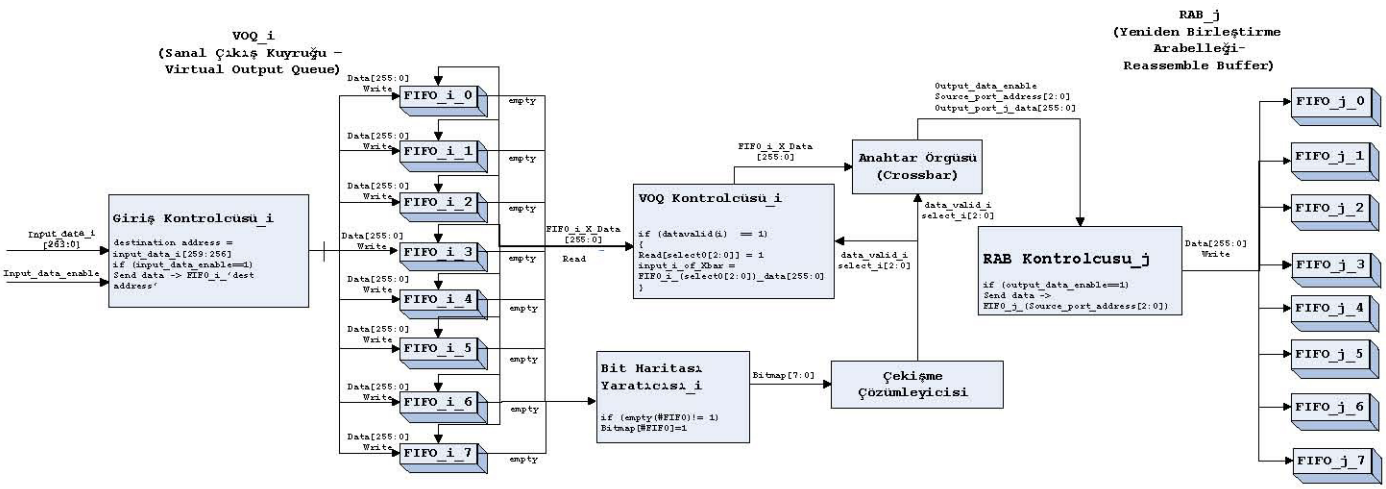
Anahtar tasarımımız Vivado Design Suite 2016.4 aracı kullanılarak XC7Z100 SoC PL üzerinde gerçekleştirilmiştir. Çalışma frekansı 156.25 MHz ile 40 Gbps veri çıkışını desteklemektedir. Sırasıyla FPGA LUT, LUTRAM, FF, BRAM kaynaklarının kullanım oranları sırasıyla %12, %10, %14 ve %34'tür. Vivado Design Suite 2016.4 aracı ile hesaplanan toplam güç tüketim tahmini 645 mW'tır.

B. CreditArbiteR (CAR) Çekişme Çözümleyicisi

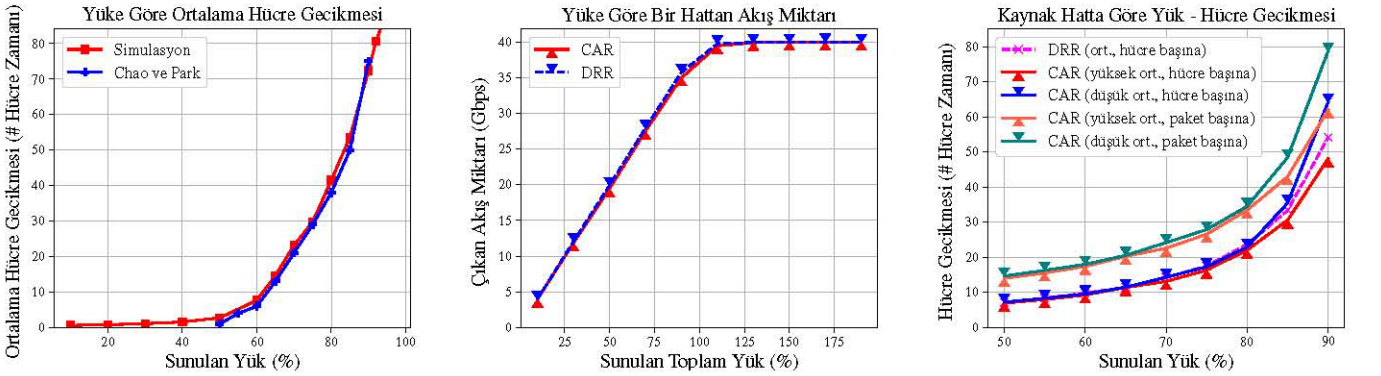
Çekişme Çözümleyicisinin kararları, hatların anahtar örgüsüne erişimini kontrol etmektedir. DRR sadece veri çıkışını artırmayı amaçlamaktadır. Bu bildirimizde her giriş-çıkış hattı i, j çifti için *anahtarlama servisi farklılaşması* (switching service differentiation) sağlayabilen, iş koruma özellikli (work conserving), yeni bir çekişme çözümleyicisi CreditArbiteR (CAR) önermekteyiz. CAR çevirimsel sıralı ve giriş çıkışlara dağıtılmış karar hesaplaması ile ölçeklenebilir gerçekleştirimdedir ve Şekil 1'deki mimariye uyumludur.

Her i, j çiftini bir *bağlantı* olarak tanımlamaktayız. CAR'da her bağlantı için sırasıyla giriş i ve çıkış j hatlarında istenen servis kalitesine uygun olarak $KabulTamKredi(i,j)$ ve $OnayTamKredi(i,j)$ *kredi* atanmaktadır. Giriş i için bir *çekişme çevirimini* sırasıyla bütün $j = 0 \cdots (N-1)$ çıkışlara öncelik verilen $T_{IN,i} = \sum_j KabulTamKredi(i,j)$ (çıkış için $T_{OUT,j} = \sum_i OnayTamKredi(i,j)$) anahtarlama döngüsü olarak tanımlayalım. Buna göre, i'deki (j'deki) çekişme çözümleyici $T_{IN,i}$ ($T_{OUT,j}$) boyunca j'ye (i'ye) $KabulTamKredi(i,j)$ ($OnayTamKredi(i,j)$) çevirim döngüsü öncelik vermektedir. Anahtarlama döngüsü boyunca giriş i'deki (çıkış j'deki) *güncel kredi miktarını* $KabulKredi(i)$ ($OnayKredi(j)$) ile gösteriyoruz. Notasyonda kullanılan semboller Tablo I'de verilmiştir.

CAR algoritmasında, her giriş i kuyrukladığı hücrelerin çıkışlarına *istek* gönderir. Bunu takiben her çıkış j aldığı istekler arasından $OnayKredi(j)$ değerine göre seçtiği girişe



Şekil 1: Anahtar Mimarisi.



(a) Simülatör doğrulaması. (b) CAR iş koruması. (c) CAR servis farklılaşmasının gecikmeye etkisi.

Şekil 2: Servis kalitesi simülatör deneyleri.

onay gönderir. Son aşamada her giriş i $KabulKredi(i)$ değerlerine göre seçtiği çıkışa *kabul* gönderir. $OnayKredi(j)$ ve $KabulKredi(i)$, i, j arasında anahtarlanan her hücre için azaltılır. Her çıkış j'de ve giriş i için çevirimsel sıralı çekişme çözümleyiciler güncel anahtarlama çeviriminde ilk öncelikle onay gönderilecek giriş ve kabul gönderilecek çıkış hattını işaretçilerle (pointer) belirlerler. Herhangi bir anda bir onay (kabul) işaretçisi öncelik verdiği girişe ek olarak bu tercih için güncel krediyi de tutar. Güncel kredi sıfıra indiğinde bir sonraki tercih edilen girişe (çıkışa) geçilir ve kredi buna göre yenilenir. İstek-kabul-onay dizisi birden fazla yineleme (iteration) tekrarlanarak bağlanan giriş çıkış sayısı artırılır. Giriş ve çıkışların birlikte çalışması Algoritma 1'de gösterilmektedir.

CAR her çekişme çeviriminde her bağlantıya en az kredisi kadar hücre çıkarma fırsatı verir. Kredi tercih önceliği belirttiği için [8]'den üstün olarak CAR iş-koruyan yapıdadır. Hücre bulunduran bir kuyruğun yalnızca kredisini bitirmiş olması nedeniyle çıkış yapamaması söz konusu değildir. İşaretçide öncelikli olarak gösterilmeyen giriş veya çıkış kendinden daha yüksek öncelikli bir bağlantının kullanmadığı anahtarlama kapasitesinden faydalanabilir.

C. Simülasyon Gerçeklemesi ve Başarım İncelemesi

Gerçekleştirdiğimiz anahtar yapısının ve başarımını ölçmek için Bölüm III-A'daki gerçekleştirimi donanıma yakın detayda modelleyen olay-tabanlı bir simülatörü C++ dilinde geliştirdik. Simülatörün ürettiği zaman damgalı olay kaydı dosyalarını Python dilinde işleyerek istatistiksel ve diğer ölçümleri elde ettik. Simülatörümüzü doğrulamak için [6]'da Chao ve Park tarafından sunulan 16x16 anahtar üzerinde uniform yükle elde edilen DRR deneyini tekrarladık. Burada yük hat kapasitesine oranla o hatta gönderilen toplam trafiği göstermektedir. Sonuçların benzerliği Şekil 2 (a)'da görülmektedir.

Algoritma 1'i simülatörümüzde gerçekledik. 8x8 anahtarın her hattı 40 Gbps hızda çalıştırıldı. YYB arayüzlerine düşük öncelik verilecek şekilde 1, kalan arayüzlere yüksek öncelik verecek şekilde 9 kredi atadık. Bu öncelik seçiminini yaparken tasarımcı kontrolünde olmayan arayüzlerden gelebilecek çok miktarda veriyi hızlı biçimde anahtarlamayı hedefledik. Deneylerde %95 maksimum yük altında ortalama 16.21 hücrelik kuyruk oluşabildiği gözlenmiştir. Bu sonuçlar FIFO boyutlarının seçilmesinde etken olmuştur.

Yollanan paketler %1 ihtimalle 40 bayt ve %99 ihtimalle

OnayİşaretÇi$(j) \leftarrow 0$
OnayKredi$(j) \leftarrow$ *OnayTamKredi*$(0, j)$
KabulİşaretÇi$(i) \leftarrow 0$
KabulKredi$(i) \leftarrow$ *KabulTamKredi*$(i, 0)$
for *yineleme*$\leftarrow 0$ **to** *YinelemeSınırı* **do**
  // İstek aşaması
  İstek$(i, j) \leftarrow 0$
  for (i, j), $i, j \in 0 \ldots (N-1)$ **do**
    if *Giriş* i *ve çıkış* j *bağlı değil* **then**
      if *Giriş* i*'de çıkış* j *için hücre var* **then**
        İstek$(i, j) \leftarrow 1$
  // Onay aşaması
  Onay$(i, j) \leftarrow 0$
  for $j \in 0 \ldots (N-1)$ **do**
    for $i' \in 0 \ldots (N-1)$ **do**
      $i \leftarrow (i'+$*OnayİşaretÇi*$(j))$ **mod** N
      if *İstek*$(i, j) = 1$ **then**
        Onay$(i, j) \leftarrow 1$
        break
  // Kabul aşaması
  Kabul$(i) \leftarrow$ BOŞ
  for $(i, j') \in 0 \ldots (N-1)$ **do**
    $j \leftarrow (j'+$*KabulİşaretÇi*$(i))$ **mod** N
    if *Onay*$(i, j) = 1$ **then**
      Kabul$(i) \leftarrow j$
      i, j bağlandı
      if *OnayKredi*$(j) > 1$ **then**
        OnayKredi$(j) \leftarrow$*OnayKredi*$(j) - 1$
      else
        $i_{sonraki} \leftarrow (i + 1)$ **mod** N
        OnayİşaretÇi$(j) \leftarrow i_{sonraki}$
        OnayKredi$(j) \leftarrow$*OnayTamKredi*$(i_{sonraki}, j)$
      if *KabulKredi*$(i) > 1$ **then**
        KabulKredi$(i) \leftarrow$ *KabulKredi*$(i) - 1$;
      else
        $j_{sonraki} \leftarrow (j + 1)$ **mod** N
        KabulİşaretÇi$(i) \leftarrow j_{sonraki}$
        KabulKredi$(i) \leftarrow$*KabulTamKredi*$(i, j_{sonraki})$
  if *Yeni bağlantı yok* **then**
    break
Kabul edilen bağlantı hücrelerini anahtarla:
Kabul$(i) \rightarrow j$

Algoritma 1: CAR sözde kodu

1500 bayt boyutunda, anahtar akış miktarı her hat için maksimum 40 Gbps olacak şekilde ayarlanmış ve DRR [6] ile karşılaştırmalı olarak değerlendirilmiştir. Hem DRR hem CAR en fazla 3 yineleme ile çalışmaktadır. Şekil 2 (b) CAR ve DRR'nin %200'e kadar aşırı yüklenmesi halinde aynı şekilde hat maksimum hızı olan 40 Gbps'e ulaştığını göstermektedir. Bu deneyden CAR'ın *iş koruyan* yapıda çalıştığı ve *hat eşlemelerini kısıtlamadığı* görülmektedir.

CAR ile yapılan önceliklendirmenin ortalama hücre ve paket gecikmelerine etkisi ise Şekil 2 (c)'de görülebilir. Anahtarın yüksek ve düşük öncelikli hatları kendi aralarında benzer sonuçlar vermiştir. Yüksek yüklerde artan çekişme nedeni ile önceliklendirmenin gecikmelere etkisi belirginleşmiştir. DRR ile eşit şartlarda karşılaştırma hücre gecikmeleri ile yapılmıştır. DRR gecikmelerinin CAR altında yüksek ve düşük öncelikli bağlantılar arasında kalması CAR'ın önceliklendirmeyle yaptığı ödünleşimi göstermektedir.

IV. Sonuç ve Gelecekteki Çalışmalar

Bu bildiride bulut servisleri için FPGA yeniden yapılandırılabilir bölgelerinde gerçeklenen donanım hızlandırıcılar ve diğer donanım modülleri arasında haberleşme sağlayan bir yonga-üstü anahtar mimarisi önerilmiştir. Mimari Zynq-7000 SoC XC7Z100 üzerinde 40 Gbps hat hızında gerçeklenmiştir. Ölçeklenebilirlik için anahtar örgüsü hat hızında çalıştırılmıştır. Anahtar örgüsü çekişme çözümlemesi için yeni bir yöntem olan CAR yöntemi önerilmiştir. Bu yöntem, bağlantılara farklı önceliklerde anahtar örgüsü erişimi vermektedir. CAR başarımı yazılım simülatörü ile incelenmiştir. Sonuçlar, önceliklendirme yapmanın anahtar örgüsünde bağlantı kurulmasını kısıtlamadığını ve CAR yönteminin iş-koruyan yapıda çalıştığını göstermektedir. Sadece veri çıkışını artırmayı hedefleyen DRR algoritması ile karşılaştırıldığında önceliklendirmenin özellikle yüksek yükler altında gecikme farklılaşması sağladığı gözlenmiştir. Araştırmamızın bir sonraki adımında CAR yöntemi anahtar mimarimizde FPGA üzerinde gerçeklenecektir.

TEŞEKKÜR

Bu çalışma, 117E667-117E668 nolu proje kapsamında TÜBİTAK tarafından desteklenmektedir. Yazarlar desteklerinden dolayı TÜBİTAK'a ve ASELSAN A.Ş.'ye teşekkür eder.

Kaynaklar

[1] A. M. Caulfield, E. S. Chung, A. Putnam, H. Angepat, J. Fowers, M. Haselman, S. Heil, M. Humphrey, P. Kaur, J.-Y. Kim *et al.*, "A cloud-scale acceleration architecture," in *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 2016, p. 7.

[2] A. Yazar, A. Erol, and E. G. Schmidt, "Accloud (accelerated cloud): A novel fpga-accelerated cloud archictecture," in *2018 26th Signal Processing and Communications Applications Conference (SIU)*. IEEE, 2018, pp. 1–4.

[3] N. U. Ekici, K. W. Schmidt, A. Yazar, and E. G. Schmidt, "Resource allocation for minimized power consumption in hardware accelerated clouds," in *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2019, pp. 1–8.

[4] "Zynq-7000 SoC data sheet: Overview." [Online]. Available: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf

[5] N. McKeown, "The islip scheduling algorithm for input-queued switches," *IEEE/ACM transactions on networking*, no. 2, pp. 188–201, 1999.

[6] H. J. Chao and J.-S. Park, "Centralized contention resolution schemes for a large-capacity optical atm switch," in *1998 IEEE ATM Workshop Proceedings.'Meeting the Challenges of Deploying the Global Broadband Network Infrastructure'(Cat. No. 98EX164)*. IEEE, 1998, pp. 11–16.

[7] B. Hu, F. Fan, K. L. Yeung, and S. Jamin, "Highest rank first: A new class of single-iteration scheduling algorithms for input-queued switches," *IEEE Access*, vol. 6, pp. 11 046–11 062, 2018.

[8] D. Stiliadis and A. Varma, "Providing bandwidth guarantees in an input-buffered crossbar switch," in *Proceedings of INFOCOM'95*, vol. 3. IEEE, 1995, pp. 960–968.

[9] R. R. Dobkin, R. Ginosar, and A. Kolodny, "Qnoc asynchronous router," *Integration*, vol. 42, no. 2, pp. 103–115, 2009.

[10] T. Bjerregaard and J. Sparso, "A router architecture for connection-oriented service guarantees in the mango clockless network-on-chip," in *Design, Automation and Test in Europe*. IEEE, 2005, pp. 1226–1231.

[11] C. Li, D. Dong, Z. Lu, and X. Liao, "Rob-router: A reorder buffer enabled low latency network-on-chip router," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 9, pp. 2090–2104, 2018.

[12] "40Gbps Ethernet solution." [Online]. Available: http://hiteksys.com/pdf/40G-Ethernet-Verification-Report.pdf