Hata Doğrulama Kodlarının Veri Önbelleklerindeki Geçici Hataları Düzeltmedeki Verimliliğinin Artırılması

Enhancing the Efficiency of Error Correcting Codes in Correcting the Transient Errors in Data Caches

Kadir TUNÇER[1] ve İsmail KADAYIF[2]

[1]Enformatik Bölümü
Çanakkale Onsekiz Mart Üniversitesi
kadirtuncer@comu.edu.tr

[2]Bilgisayar Mühendisliği Bölümü
Çanakkale Onsekiz Mart Üniversitesi
kadayif@comu.edu.tr

Özet

Programların doğru sonuç üretmesi, önemli ölçüde üzerinde işlem yapılan verilerin bozulmamasına bağlıdır. Isının etkisiyle paketleme malzemelerinden ortaya çıkan alfa parçacıkların ve güneş ışınlarından gelen nötron parçacıkların devre elemanlarına çarpmaları önbelleklerde saklanan verilerde hataların ortaya çıkmasına sebep olabilmektedir. Daha da kötüsü silikon teknolojisindeki gelişmelere paralel olarak transistor boyutlarında yaşanan küçülmeler devreleri bu hatalara karşı daha da duyarlı hale getirmektedir. Geçici hata diye adlandırılan bu tip hatalardan korunmak için eşlik kontrolü ve çeşitli hata düzeltme yöntemleri geliştirilmiştir. Bu çalışmamızda ilk önce geçici hatalara karşı bu tip hatadan korunma tekniklerinin mevcut veri saklama yöntemlerinde genelde yetersiz kaldığını göstermeye çalıştık. Daha sonra veri bitlerini önbelleğe araya girmeli şekilde yerleştirdiğimizde hata düzeltme yöntemlerinin hataları bulmada ve düzeltmede yeterli hale getirilebileceğini göstermeye çalıştık. Bunun için önbellek verilerinde rastgele yapay hatalar oluşturarak SimpleScalar Simülatörü ortamında testler yaptık. Testleri SPEC2000 grubundan bazı programlar üzerinde gerçekleştirdik.

Abstract

The ability of programs to generate correct results largely depends on the processed data not being corrupted. The strikes to logic devices by alpha particles dissipating from packaging materials due to heating and by neutron particles from cosmic rays can cause the emergence of data errors in cache memories. To make things worse, the miniaturization of device geometries, in parallel with advances in silicon technologies, makes circuits more vulnerable to these errors. To protect data from such kind of errors, which are referred to as transient (soft) errors, parity check and several error correcting mechanisms were developed. In this study, we first tried to show that such kind of techniques are generally not enough to provide the needed data reliability with current data storage techniques in caches. Then, we also showed that these can be turned into very effective error detection and correction mechanisms if data are stored in caches in an interleaved fashion. To do so, we conducted some experiments with SimpleScalar Tools by generating random errors and injecting them into data stored in the cache. The tests were implemented on some application programs in SPEC2000 suite.

1. Giriş

1960'lı yıllardan beri Moore Kanunu gereği her yeni nesil silikon teknoloji aynı yonga alanına daha fazla transistor sığdırılmasına olanak tanımaktadır. Bunu gelişen yarı iletken teknolojisiyle daha küçük boyutlu transistorların tasarımı mümkün kılmaktadır. Gittikçe küçülen transistor boyutları, daha düşük eşik gerilimi ile çalışma ve performansı artırmak için daha yüksek frekanslar kullanımı devreleri hatalara karşı daha duyarlı hale getirmektedir. Dolayısıyla günümüzde devre güvenirliği performans, düşük enerji harcanması ve az yer kaplama gibi tasarımda önem arz eden unsurlardan biri haline gelmiştir.

Yarı iletken teknolojilerde güvenirliği etkileyen temelde iki tür hata vardır. Bu hatalardan ilki, tasarım sırasında oluşan hatalar olup ilgili devrenin her kullanımında hatalı sonuçların üretilmesine veya depolanan verilerin hatalı okunmasına sebep olan hatalardır. Bu tip hatalar *kalıcı hatalar* (permanent/hard errors) olarak adlandırılırlar [1]. Bunlardan kurtulmanın en basit yolu, test sonucu tespit edilen hata barındıran yongaların çöpe atılması veya hatanın olduğu donanım bileşeninin devre dışı bırakılmasıdır.

Güvenirliğe etki eden ikinci tip hatalar ise *geçici hatalar* (transient/soft errors) [2] diye adlandırılırlar. Bunlar devrelerin hatalı tasarımı sonucu ortaya çıkan hatalar olmayıp, uzaydan gelen yüksek enerjili nötron parçacıkların veya ısısı yükselen paketleme malzemelerinden yayılan alfa parçacıkların transistorlara çarpmaları sonucu transistor yüklerinde meydana gelen değişiklikler sonucu ortaya çıkar. Kalıcı hataların aksine

geçici hataların ortaya çıkmasında çevre koşulları ve çalışma koşulları önem arz etmektedir. Örneğin daha yüksek kesimlerde uzaydan gelen yüksek enerjili parçacıkların etkisi daha yaygın olacağından, uçaklardaki bilgisayarlar yeryüzündeki bilgisayarlara oranla geçici hatalara daha sık maruz kalırlar. Silikon teknolojideki ölçekleme daha küçük boyutlu, daha düşük eşik gerilimli ve daha yüksek hızlarda anahtarlama yapabilen transistorlarla çalışmayı gerektirmektedir. Bu özellikler ise devreleri geçici hatalara karşı daha duyarlı hale getirmektedir [3]. Dolayısıyla gelecek teknoloji nesillerinde geçici hatalar daha da önemli bir güvenirlik problemi olarak karşımıza çıkmaya devam edecektir.

Kalıcı hatalardan daha sık oluşan geçici hatalar, mikroişlemcilerin doğru sonuç üretmelerine engel olabilir. İşlemcinin herhangi bir bileşeni geçici hatalardan etkilenebilmesine rağmen, işlemcinin önemli bir alanını kapladıklarından dolayı bu hatalardan en çok önbellekler etkilenir. Ayrıca önbellekler işlemcilere çok yakın olduğundan ve alan sınırlamasından dolayı minimum boyutlu transistorlarla tasarlandıklarından, bu bileşenlerdeki geçici hatalar yazmaç dosyası yoluyla diğer bileşenlere kolayca yayılabilir [4]. Bu durum beklenmedik sonuçların üretilmesine ve hatta bilgisayar sisteminin tamamen çökmesine neden olabilir. İşlemcilerin yanlış sonuç üretmesine ve sistemin çökmesine engel olmak için geçici hataların tespit edilmesi ve düzeltilmesi çok önem arz etmektedir.

Bu çalışmanın konusu olan geçici hatalara karşı veri güvenirliğini artırmak için geliştirilen yöntemler genel olarak üç kategoriye ayrılabilir. Bunlar sırasıyla işlem teknolojisi, devre çözümleri ve mimari düzeyde çözümlerdir. İşlem teknolojisi temelli çözümlerde aygıtları geçici hatalara karşı korunaklı hale getirmek için silikon yalıtkan (SOI) kullanılır. SOI ile yalıtılan transistorlar yayılan ışınlardan daha az yük toplar. Bu husus, parçacık çarpmalarının devre çıkışlarında daha az bir ihtimalle bit değişikliklerine yol açmasını sağlar. IBM, SOI teknolojisi kullanılmak suretiyle SRAM transistorlarında geçici hata oluşumu sıklığının %80 oranında azaltılabileceğini rapor etmiştir [5]. Devre seviyesi çözümlerde hücre sığası ile besleme ve eşik gerilimleri ayarlanmak suretiyle radyasyona dayanıklı devreler tasarlanır. Bu tip devreler SRAM hücresinin çıkışında bit değişimi yaşanması için gerekli olan asgari yük miktarını artırır.

Veri güvenirliğini artırmaya yönelik mimari düzeyde birçok çözümler vardır. Ancak her bir çözümün getirdiği dezavantaj da vardır. Örneğin, çözümlerden biri her bir komutu iki kez çalıştırıp bulunan iki sonucu karşılaştırmaktır. Eğer iki sonuç da aynı ise bulunan sonuç doğru varsayılarak sonraki komutlar çalıştırılmaya devam edilir; aksi takdirde aynı komutu üçüncü kez çalıştırıp çoğunluğun sonucu doğru olarak kabul edilir. Ancak, hesaplamalarda yapılabilecek hatalı veri üretimini gidermeye yönelik bu faydalı yaklaşım performansa olumsuz yönde etki edecektir. Bellek hiyerarşisinde verilerin saklanması sırasında verilerde geçici hatalar sonucu oluşabilecek veri bozulmalarını önlemenin bir yolu aynı verinin yedek bir kopyasını tutmaktır. Veri okunurken yedeğiyle de karşılaştırılır. Eğer asıl veri ile yedeği farklı ise ilgili verinin geçici hataya maruz kaldığı varsayılıp, hatalı veri bellek hiyerarşisinde bir sonraki seviyede tutulan kopyası ile düzeltilir [6]. Bu yöntemin dezavantajı ise verilerin kopyalarının saklanması için ekstra bellek alanına ihtiyaç duyulmasıdır. RAM ve SRAM'lerde geçici hatalardan kaynaklanan veri bozulmalarına karşı geliştirilen ve pratikte çok sık kullanılan yöntemler ise *eşlik kontrolü* (parity checking) ve *hata düzeltme kodlarıdır* (error correction codes) [7]. Eşlik kontrolü, 8 bitlik veriye ekstra 1 bit eklemek suretiyle veride tek sayıdaki bitlerde oluşan hataların tespitinde kullanılır. Eşlik kontrolü gerçekleştirimi basit olmasına rağmen veride oluşabilecek herhangi bir hatayı düzeltemediği için fazla yararlı değildir. Birçok önbellek tasarımında hata düzeltme yöntemi olarak *tek hata düzeltimi çift hata tespiti* (single error correct double error detect; SECDED) mekanizması kullanılmaktadır. SECDED yöntemiyle 64 bitlik veri için ekstra 8 bit kullanmak suretiyle veride oluşabilecek 1 ve 2 bitlik veri bozulmaları tespit edilebilmekte ve bu bozulmalardan sadece 1 bitlik olanları düzeltilebilmektedir.

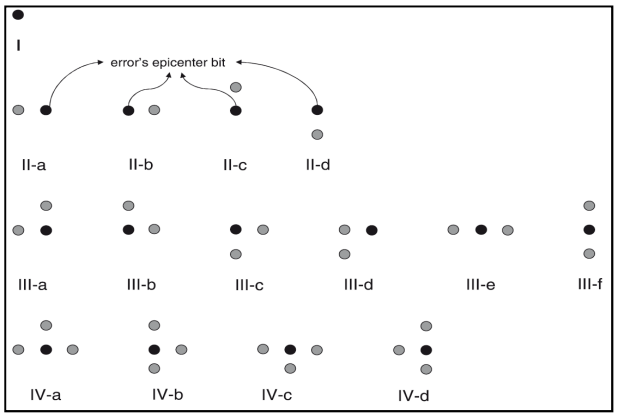
Bu çalışmamızda önbelleklerde hata tespiti ve düzeltimi için yaygın olarak kullanılan SECDED yönteminin geçici hataları bertaraf etmedeki verimliliğini araştırmaya çalıştık. SECDED yöntemi, 2 bitlik hataları tespit edip bunlardan 1 bitlik olanları düzeltebilmektedir. Ancak yarı iletken teknolojideki gelişmelere paralel olarak transistorların boyutlarının küçülmesi sonucu geçici hataların önemli bir kısmı çok bitli hatalar şeklinde ortaya çıkmaktadır. Bu çalışmadaki ilk tespitimiz, veri bitlerini önbelleğe klasik şekilde yerleştirirsek (önce ilk kelimenin bitleri, daha sonra ikinci kelimenin bitleri, ...) SECDED yöntemi çok bitli hataları düzeltmede yetersiz kalmaktadır. İkinci tespitimiz ise, SECDED yönteminin çok bitte oluşan geçici hataları bulmada ve düzeltmede veri bitlerini önbelleğe araya girmeli şekilde yerleştirdiğimizde daha verimli hale getirilebileceğidir. Önbellek olarak birinci seviye veri önbelleğini çalışmamızda dikkate aldık. Aynı çalışmayı ikinci seviye önbellek için de yapmak mümkündür.

Bu çalışmanın geri kalan kısmı şöyle organize edilmiştir. Gelecek bölümde önceki çalışmalardan bahsettik. 3. Bölümde kullandığımız materyal ve yöntemimizi açıkladık. Araştırma bulguları 4. Bölümde verilerek, 5. Bölümle çalışmamızı tamamladık.

2. Önceki Çalışmalar

Phelan [8]'a göre silikon teknolojide yaşanan gelişmelere paralel olarak birçok nedenden dolayı geçici hataların oluşma sıklığı giderek artmaktadır. Bu nedenlerden bazılarını şöyle sıralayabiliriz. Eğer bir tümleşik devrede daha çok bellek ve mantık devresi varsa, parçacığın çarpabileceği daha çok alan mevcut olup geçici hatanın oluşması daha büyük bir ihtimal dâhilindedir. Tasarımlarda daha düşük gerilim değerlerinin ve daha küçük sığaların kullanılması, parçacık çarpması sonucu bir SRAM hücresinin tuttuğu 1 değerinin 0'a ya da 0 değerinin 1'e dönüşme ihtimalini artırır. Küçülen geometrilerden dolayı parçacık çarpmaları, verinin birden çok bitinde bozulmalara yol açabilir [9]. Bu tip hatalara *çok bitli hatalar* (kümelenmiş hatalar) denilmektedir. Daha da önemlisi, daha ince silikon teknolojileri geliştirildikçe parçacık çarpmaların yol açtığı çok bitli veri bozulmaların üstel olarak artacağı vurgulanmaktadır [9], [10].

Çeşitli nedenlerle önbelleklerde oluşabilecek hatalar farklı bit seviyelerinde ortaya çıkabilir. Kadayıf ve ark., yaptıkları çalışmada 1, 2, 3 ve 4 bit hataları farklı tiplerde modellemişlerdir [4]. Şekil 1'de Kadayıf ve ark. önerdiği ve bu



Şekil 1: 1 bit, 2 bit, 3 bit ve 4 bitlik kümelenmiş hata çeşitleri. Hatanın merkez (epicenter) biti siyah ile hatadan etkilenen komşu bitler ise gri olarak gösterilmektedir.

çalışmada da göz önünde bulundurulan çok bitli veri bozulma tipleri gösterilmektedir. Bu modele göre parçacık çarpması sonucu önbellekte veri bozulması birbirine komşu bir küme bitte oluşmaktadır. Hatanın şekli çarpmanın etkisini en fazla hisseden bir bitle belirlenmektedir. Bu bite hatanın *merkez biti* (epicenter bit) denilmektedir. Şekle bakacak olursak, iki bitte bozulmaya sebep olan dört farklı hata tipi vardır. Bunlardan ilki ve ikincisi (II-a ve II-b) iki bitlik bozulmanın aynı önbellek bloğundaki bitlere tekabül ettiğini göstermektedir. Oysa II-c ve II-d geçici hatadan etkilenen iki bitin farklı önbellek bloklarına ait olduğunu göstermektedir. Dolayısıyla SECDED II-a ve II-b hatalarını düzeltemezken, II-c ve II-d hatalarını düzeltebilir. Şekilden de anlaşılacağı üzere 3 bitlik hata 6 farklı, 4 bitlik bir hata 4 farklı şekilde olabilir. IV-a ve IV-c bozulan 4 bitten üçünün aynı önbellek bloğunda, diğer bitin ise farklı önbellek bloğunda olduğunu göstermektedir. Buna göre bozulmuş iki farklı bloktaki verilerden SECDED yöntemi birini (tek biti bozulanı) düzeltebilir. Şekil IV-b ve IV-d, 4 bitlik kümelenmiş veri bozulmasının üç farklı ön bellek bloğunu etkilediğini göstermektedir. Verisi bozulan bu üç önbellek bloklarından ikisinin verisi SECDED tarafından düzeltilebilirken, iki biti bozulmuş olan diğer bloğun verisi SECDED tarafından düzeltilemez.

Çizelge 1: Veri önbelleğinde geçici hata oluşma sıklıkları

Hata şekli	Hata Sıklığı
1 bit hata	1E-6
2 bit hata	1E-7
3 bit hata	1E-8
4 bit hata	1E-8

3. Materyal ve Yöntem

Bu araştırmada SECDED yönteminin veri önbelleklerde geçici hataların düzeltilmesinde ne derecede başarılı olduğu araştırılmıştır. Bunun için program çalışması esnasında (runtime) önbellekte tutulan verilerde rastgele hatalar üretilmiştir. Diğer bir deyişle verilere yapay yolla hatalar enjekte edilmiştir. Verilere ait bitlerden bazıları 0'dan 1'e bazıları ise 1'den 0'a dönüştürülmüştür.

```
do{
    if(random() % geçici_hata_oluşma_oranı == 0) {
        küme = random() % cp->nsets;
        blok = random() % cp->assoc;
        kelime = random() % (cp->bsize / 8);
        hata_merkez_biti = random() % 64;
        hata_çeşidi = random() % hata_çeşidi_sayısı;

        // hesaplanan küme, blok, kelime, hata merkez
        // biti ve hata çeşidine uygun olarak ilgili blok
        // ve blokları bozulmuş veri barındırıyor diye
        // işaretle.
    }
    çevrim_sayısı++;
}while(simülasyon_sonlanmadığı_sürece)
```

Şekil 2: Önbellekte yapay geçici hataların oluşturulmasına ilişkin yalancı kod parçası.

Çizelge 1'de çok bitli değişik hata tipleri için deneylerimizde kullandığımız hata sıklıkları verilmektedir. Çalışmamızda 1 bitten 4 bite kadarki hatalar üzerinde incelemeler yaptık. Kullandığımız hata sıklık oranları [11]'de kullanılanlarla benzerdir. Ayrıca Wrobel ve ark. [12] bulgularına paralel olarak, 1 bitlik veri bozulmaların 2 bitlik veri bozulmalarına göre 10 kat ve 3 bitlik veri bozulmalarına göre 100 kat sıklıkta olduğunu göz önünde bulundurduk.

3.1 Önbelleklerde Hata Oluşumunun Modellenmesi

Bu araştırmada 1, 2, 3 ve 4 bitlik veri bozulmaları üzerinde durulmuştur. 4 bitten daha fazla bitte oluşabilecek hataların oranı çok küçük olduğundan bunları modellemedik. Geçici hatalar rastgele oluştukları için, simülasyon ortamında hataların şekline, oluşma zamanına ve önbellekte bozulmaya sebep oldukları bitlere (blok veri bitleri veya blok etiket bitleri) rastsal (random) sayılar üretmek suretiyle karar vermekteyiz.

Şekil 2 önbellekte yapay geçici hataların oluşturulmasına ilişkin simülasyon koduna eklenen kod parçacığının amacını ve mantığını gösteren yalancı kodu göstermektedir. Kullanılan SimpleScalar [13] simülatörü çevrim doğruluklu (cycle accurate) olduğu için her makine çevriminde önbelleğin neresine erişildiğine, hangi yazmaçlardan okuma hangi yazmaçlara yazma yapıldığı gibi birçok şeyi kontrol eder. Üretilen yapay veri bozulmalarını simülatöre tanıtmak için her makine çevrimi için aşağıdaki işlemler gerçekleştirilmiştir. İlk rastsal sayı kontrol edilerek ilgili makine çevriminde bir yalancı hata oluşup oluşmadığına karar verilmektedir. Bunun için hata oluşma oranlarından faydalanılmıştır. Örneğin, 4 bitlik kümelenmiş geçici hata oluşma oranını önbellekte bir bit için 10^{-8} olarak kabul edersek, üretilen rastsal sayının önbellekteki *toplam bit sayısı* $x10^{8}$'e göre modu 0'a eşitse bu durumda veri önbelleğimizde 4 bitlik bir hata oluştuğu öngörülmüştür. Daha sonra yine beş farklı rastsal sayı üretilerek, hatanın konumun ve şeklinin belirlenmesine karar verilmiştir. Rastsal sayılardan ilk üçüyle merkez bitin tekabül ettiği küme, blok ve kelime tespit edilmiştir. Bu araştırmada kelime boyu 64 bit (8 byte) kabul edildiği için, kelimenin numarası blok boyu 8'e bölünmek suretiyle belirlenmiştir. Örnek olarak, önbellek blok boyunun 64 byte olduğunu düşünürsek, kelime numaraları 0 ile 7 arasında değer alacaktır. Hatanın merkez biti kelime uzunluğu

(64 bit) dikkate alınarak tespit edilmiştir. Son rastsal sayıyla hatanın çeşidine karar verilmiştir. Dört bitlik kümelenmiş hataları göz önünde bulundurursak, bu tip hataların 4 farklı şekli olmaktadır. Bu durumda 0 ile 3 arasında bir rastsal sayı üretilmek suretiyle hatanın hangi şekilde oluştuğuna karar verilmiştir. Daha sonra ön belleği temsil eden veri yapısı dolaşılmak suretiyle hatanın konumu işaretlenmiştir.

Çizelge 2. Sistemimizin temel yapılandırma parametreleri

İşlemci Çekirdeği	
Fonksiyonel Birimler	4 tamsayı ve 4 FP ALU, 1 tamsayı çarpma/bölme birimi, 1 FP çarpma/bölme birimi
LSQ boyu	64
RUU boyu	64
Fetch/decode/issue/commit width	4 komut/cycle
Fetch kuyruk boyu	16 komut
Önbellek ve bellek hiyerarşisi	
L1 komut önbelleği	32KB, 2-yollu, 64 byte blok boyu, 1 cycle gecikme
L1 veri önbelleği	32KB, 4-yollu, 64 byte blok boyu, 1 cycle gecikme, write-back
L2 önbelleği	4MB birleşik, 2-yollu, 128 byte blok boyu, 6 cycle gecikme
Veri/komut TLB	128 giriş, 30 cycle gecikme
Memory	100 cycle gecikme

3.2 Simülasyon Ortamı

Araştırmamızda SimpleScalar simülatörü kullanılmıştır. SimpleScalar günümüzdeki modern RISC temelli süperscalar işlemcilerde programların çalışmasını simüle eden ve Linux türevi platformlarda çalışabilen yazılım araçlarını barındırır. SimpleScalar araçları araştırma ve öğretim için yaygın olarak kullanılmaktadır. Biz bu çalışmada sıra dışı komut çalışımını (out of order execution) modelleyen *sim-outorder* yazılım aracının kodunu değiştirdik. Bu yazılım aracı önbellek, TLB, yazmaç dosyası ve fonksiyonel birimlere erişimi çevrim doğruluklu olarak modelleyebilmektedir. Biz sim-outorder aracının kodunu yukarıda açıklanan yalancı kod doğrultusunda değiştirdik. Yapılan değişikliklerle birinci seviye veri önbelleğinde yapay geçici hataların benzetimi sağlanmıştır. Geçici hataların hem veri bitlerine hem de blokların etiket bitlerine enjekte edilmesine özen gösterilmiştir. Deneylerimizde kullandığımız sistemin yapılandırma parametreleri Çizelge 2'de verilmektedir. İşlemcimiz 4 yollu Alpha benzeri bir RISC mimarisine sahiptir. Kullandığımız uygulama programları ise SPEC2000 [13] takımındandır. SPEC2000 takımından herhangi bir uygulamayı baştan sona kadar simüle etmek çok zaman aldığından, her bir uygulama için ilk 1 milyar komutu atlayarak izleyen 500 milyon komut için istatistikler topladık.

Çizelge 3. SECDED yönteminin geçici hatalara karşı performansı. Önbellekte hem veri bitleri hem de bloklara ait etiket bitlerindeki hatalar dikkate alınmıştır.

Prog.	Enjekte edilen geçici hata sayısı				Okunan bozulmuş kelime sayısı			
	1 bit	2 bit	3 bit	4 bit	1 bit	2 bit	3 bit	4 bit
swim	5479	524	58	57	0	30	5	6
mgrid	5689	535	47	60	0	38	6	8
applu	6272	643	67	65	0	34	6	7
galgel	9762	953	103	96	0	95	16	19
bzip2	5456	591	62	50	0	31	7	5
lucas	4373	485	41	45	0	50	8	10
gzip	4910	540	57	59	0	46	8	9
gcc	6514	713	66	74	0	44	7	9
mcf	5321	478	57	45	0	59	10	13
crafty	5721	634	64	59	0	51	13	11

$$b_{0,0}b_{0,1}b_{0,2}\ldots b_{0,63}b_{1,0}b_{1,1}b_{1,2}\ldots b_{1,63}b_{2,0}b_{2,1}b_{2,2}\ldots b_{2,63}\ldots\ b_{7,0}b_{7,1}b_{7,2}\ldots b_{7,63}$$

Şekil 3. Kelimelerin önbellek satırlarına normal yerleştirim şekli. Her bir kelimenin 64 bit olduğu ve önbellek satır uzunluğunun 64 byte olduğu varsayılmaktadır.

4. Araştırma Bulguları ve Tartışma

Bu bölümde deneylerimizden elde ettiğimiz sonuçları açıklamaya çalışacağız. Çizelge 3, SECDED mekanizmasının kümelenmiş hatalara karşı olan davranışını göstermektedir. Veriler önbellek satırlarına klasik olarak Şekil 3'teki gibi yerleştirildiği kabul edilmektedir. Yani önce 0. kelimeye ait bitler ardışık olarak, sonra 1. kelimeye ait bitler ardışık olarak ve nihayetinde 7. kelimeye ait bitler ardışık olarak tutulmaktadır. Şekil 3'te her bir önbellek satır uzunluğunun 64 byte ve bloklarda tutulan kelime uzunluğunun 64 bit olduğu varsayılmaktadır. Burada $b_{i,j}$, *i.* kelimenin *j.* bitini göstermektedir. Buradaki kelimeden anlatmaya çalıştığımız şey; C, C++ veya Java gibi yüksek seviyeli programlama dillerinde 64 bitlik bir tamsayıyı (long), 64 bitlik double tipinde bir kayan noktalı sayıyı veya 16 bit uzunluklu 4 tane karakteri barındıran 64 bitlik bir önbellek satır alanıdır.

Çizelge 3'ün sol tarafı programın çalışma zamanında kaç tane kümelenmiş hatanın önbelleğe enjekte edildiğini göstermektedir. Çizelgenin sağ tarafı ise oluşan bu hatalardan kaç tanesini SECDED yönteminin düzeltemediğini göstermektedir. Daha önce de bahsedildiği gibi 1 bitlik hataların hepsini SECDED düzeltebilir. Kümelenmiş hata aynı kelimenin birden fazla bitinde bozulmaya sebep olmuşsa bu durumda SECDED bu bitleri düzeltmede yetersiz kalmaktadır. Bu husus çizelgenin sağ tarafının incelenmesiyle de anlaşılabilir. Çizelgenin genelinden de görülebileceği üzere, eğer verileri normal düzende önbelleğe yerleştirirsek kümelenmiş hataların önemli bir kısmını SECDED mekanizması düzeltememektedir. Örneğin, *mgrid* programı için enjekte edilen 535 tane 2 bitlik hatadan 38 tanesi

SECDED tarafından düzeltilememektedir. Yine *gcc* programı için yapay olarak üretilip veri önbelleğine enjekte edilen 66 tane 3 bitlik hatadan 7 tanesi SECDED tarafından düzeltilememektedir. Özet olarak belirtecek olursak; SECDED 4 bite kadarki kümelenmiş hataların önemli bir kısmını düzeltmesine rağmen azımsanamayacak bir kısmını da düzeltememektedir. Geçici hata sıklığının yeni nesil silikon teknolojilerde daha da artacağı göz önünde bulundurulursa bu haliyle SECDED yönteminin gerek duyulan veri güvenirliğini sağlamada yetersiz olduğu açıktır.

$b_{0,0}b_{1,0}b_{2,0}...b_{7,0}b_{0,1}b_{1,1}b_{2,1}...b_{7,1}b_{0,2}b_{1,2}b_{2,2}...b_{7,2}...\ b_{0,63}b_{1,63}b_{2,63}...b_{7,63}$

Şekil 4. Kelimelerin önbelleğe araya girmeli şeklinde yerleştirilmesi. Her bir kelimenin 64 bit olduğu ve önbellek satır uzunluğunun 64 byte olduğu varsayılmaktadır.

Çizelge 4. SECDED yönteminin geçici hatalara karşı performansı. Verilerin önbelleğe araya girmeli şekilde yerleştirildiği kabul ediliyor

Prog.	Enjekte edilen geçici hata sayısı				Okunan bozulmuş kelime sayısı			
	1 bit	2 bit	3 bit	4 bit	1 bit	2 bit	3 bit	4 bit
swim	5479	524	58	57	0	0	0	0
mgrid	5689	535	47	60	0	0	0	0
applu	6272	643	67	65	0	0	0	0
galgel	9762	953	103	96	0	2	0	0
bzip2	5456	591	62	50	0	0	0	0
lucas	4373	485	41	45	0	0	0	0
gzip	4910	540	57	59	0	0	0	0
gcc	6514	713	66	74	0	0	0	0
mcf	5321	478	57	45	0	2	0	0
crafty	5721	634	64	59	0	0	0	0

SECDED hata düzeltme yönteminin hataları tespit etmede ve düzeltmedeki etkinliğini artırmak için verileri önbelleğe Şekil 4'te gösterildiği gibi araya girmeli olarak yerleştirebiliriz. Çizelge 4, verileri bu şekilde önbelleğe yerleştirdiğimizde elde ettiğimiz deney sonuçlarını göstermektedir. Çizelgenin sol tarafından da görüleceği üzere her bir program için çalışma zamanında önceki deneyde olduğu gibi aynı sayıda yapay hata üretmeye özen gösterdik. Çizelge 3 ve 4'ün sağ tarafları incelendiğinde, verileri önbelleğe araya girmeli olarak yerleştirmek SECDED yönteminin hataları tespit edip düzeltmede çok daha verimli hale geldiği görülmektedir. Örneğin *gcc* uygulamasını ele alalım. Veriler önbelleğe normal düzende yerleştirildiğinde SECDED yöntemi 44 tane 2 bitlik hatayı, 7 tane 3 bitlik hatayı ve 9 tane de 4 bitlik hatayı düzeltememektedir. Oysa veriler araya girmeli şekilde önbelleğe yerleştirildiğinde SECDED bu hataların hepsini düzeltebilmektedir. Bu hususu bir örnekle açıklayalım. Şekil 1'de gösterilen 3 bitlik hatalardan III-b'yi göz önüne alalım. Hatanın merkez bitinin önbellek bloğuna yerleştirilen 3. kelimenin 8. bitinde olduğunu düşünelim. Bu durumda hatalı 3 bitten 2 tanesi aynı kelimeye, 1'i ise başka bir kelimeye tekabül edecektir (Hatalı 3 bitten 2 tanesi aynı blokta diğeri ise farklı bir blokta olduğundan). Tek biti SECDED düzeltebildiğinden tek bit hataya sahip kelimeye karşı gelen veri bir problem oluşturmaz. Ancak $b_{3,8}$ ve $b_{3,9}$ bitleri bozulan kelimeye karşı gelen veri SECDED tarafından düzeltilemez. Oysa verileri araya girmeli olarak yerleştirdiğimizde yan yana hatalı iki bit iki farklı kelimenin birer bitine karşı gelecektir. Bu farklı iki kelimenin hata tespiti ve düzeltilmesine ilişkin kontrol bitleri farklı olduğundan bu kelimelerdeki birer bitlik veri bozulmaları SECDED tarafından düzeltilebilir. Sonuçlardan görüleceği üzeri, verilerin önbelleğe araya girmeli olarak yerleştirilmesi SECDED yöntemini 2 bit, 3 bit ve 4 bitlik hataların neredeyse tamamını düzeltebilecek hale getirmektedir. Düzeltilemeyen hatalar (örneğin galgel için 2 tane 2 bitlik hata) iki farklı parçacık çarpmasının aynı kelimenin farklı bitlerine tesadüf etmesinden dolayıdır.

5. Sonuçlar

Yarı iletken teknolojideki gelişmeler sonucu transistor boyutları gittikçe küçülmektedir. Bu husus kümelenmiş geçici hataların oluşumunun üstel olarak artmasına sebep olmaktadır. Oysa yaygın olarak hata bulmada ve düzeltmede kullanılan SECDED yöntemi sadece 2 bitlik hataları tespit edip bunlardan sadece 1 bit olanları düzeltebilmektedir. Bu çalışmamızda verileri önbelleğe araya girmeli olarak yerleştirdiğimizde SECDED yönteminin hataları bulmada ve düzeltmede çok daha verimli hale getirilebileceğini açıklamaya çalıştık.

6. Kaynaklar

[1] Koren, I. and Krishna, C.M., *Fault Tolerant Systems*, Morgan Kaufmann Publishers, San Fransisco, 2007.

[2] Polian, I., Hayes, J.P., Reddy, S.M., and Becker, B., "Modeling and Mitigating Transient Errors in Logic Circuits", *IEEE Transactions on Dependable and Secure Computing*, 8(4), 537-547, 2011.

[3] Gomaa, M., Scarbrough, C., Vijaykumar, T.N., and Pomeranz, I., "Transient-Fault Recovery for Chip Multiprocessors", the *Int. Symposium on Computer Architecture*, 2003, 98-109.

[4] Kadayıf, İ., Şen, H., and Koyuncu, S., "Modeling Soft Errors for Data Caches and Alleviating Their Effects on Data Reliability", *Microprocessors and Microsystems*, 34(6), 200–214, 2010.

[5] Cannon, E.H., Reinhardt, D.D., Gordon, M.S., and Makowenskyj, P.S., "SRAM SER in 90, 130 and 180 nm Bulk and SOI Technologies", *the Int. Rel. Phy. Symp.*, 2004, 300-304.

[6] Akçiçek, D., Koyuncu, S., Şen, H., Kadayıf, İ., "Exploiting Potentially Dead Blocks for Improving Data Cache Reliability Against Soft Errors", *the IEEE International Symposium on Computer and Information Sciences*, 2007, 1-6.

[7] Pradhan, D.K., *Fault-Tolerant Computer System Design*, Prentice-Hall, 2003.

[8] Phelan, R., *Addressing Soft Errors in ARM Core-based Designs*, White Paper, ARM, 2003.

[9] Kim, J., Hardavellas, N., Mai, K., Falsafi, B., and Hoe, J., "Multi-Bit Error Tolerant Caches Using Two-Dimensional Error Coding", *the Int. Symp. on* Microarchitecture, 2007, 197–209.

[10] Slayman, C.W., "Cache and Memory Error Detection, Correction, and Reduction Techniques for Terrestrial Servers and Workstations", *IEEE Transactions on Device and Materials Reliability*, 5(3), 397-404, 2005.

[11] Li, L., Degalahal, V., Vijaykrishnan, N., Kandemir, M., and Irwin, M.J., "Soft Error and Energy Consumption Interactions: a Data Cache Perspective", *the International Symposium on Low Power Electronics and Design*, 2004, 132-137.

[12] Wrobel, F., Palau, J.M., Calvet, M.C., Bersillon, O., and Duarte, H., "Simulation of Nucleon-Induced Nuclear Reactions in a Simplified SRAM Structure: Scaling Effects on SEU and MBU Cross Sections", *IEEE Transactions on Nuclear Science*, 48(6), 1946-1952, 2001.

[13] SimpleScalar Toolset. <http://www.simplescalar.com>

[14] SPEC2000 Benchmark. <http://www.spec.org>