ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ

VERİ ÖNBELLEĞİ GÜVENİLİRLİĞİNİN MODELLENMESİ VE GELİŞTİRİLMESİ

Hande ŞEN

Danışman:
Yrd. Doç. Dr. İsmail KADAYIF

Nisan, 2008
ÇANAKKALE

VERİ ÖNBELLEĞİ GÜVENİLİRLİĞİNİN MODELLENMESİ VE GELİŞTİRİLMESİ

Çanakkale Onsekiz Mart Üniversitesi Fen Bilimleri Enstitüsü

Yüksek Lisans Tezi

Bilgisayar Mühendisliği Anabilim Dalı

Hande ŞEN

Danışman:

Yrd. Doç. Dr. İsmail KADAYIF

Nisan, 2008

ÇANAKKALE

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

HANDE ŞEN, tarafından **YRD.DOÇ.DR.İSMAİL KADAYIF** yönetiminde hazırlanan **"VERİ ÖNBELLEĞİ GÜVENİLİRLİĞİNİN MODELLENMESİ VE GELİŞTİRLMESİ"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

Doç.Dr. İsmail TARHAN

Yönetici

Yrd.Doç.Dr. İsmail KADAYIF

Jüri Üyesi

Yrd.Doç.Dr. İbrahim TÜRKYILMAZ

Jüri Üyesi

Sıra No:……

Tez Savunma Tarihi:…../…../……

Müdür

Fen Bilimleri Enstitüsü

TEŞEKKÜR

Bu çalışmama deneyim ve bilgisi ile danışmanlık yapan sayın hocam Yrd. Doç. Dr. İsmail KADAYIF'a ve bana da tezin hazırlanmasında her zaman yardımcı olan Arş. Gör. Selçuk KOYUNCU'ya sonsuz teşekkürlerimi sunuyorum.

Hande ŞEN

SİMGELER VE KISALTMALAR

ACE (Architecturally Correct Execution): Mimarisel Doğru Çalışabilirlik

APEC (Active Period Error Contribution): Aktif Periyot Hata Katkısı

AVFC (Architectural Vulnerability Factor for Caches): Önbellekler için Mimari Bozulabilirlik Faktörü

CVF (Cache Vulnerability Factor): Önbellek Bozulabilirlik Faktörü

D-Cache (Data Cache): Veri Önbelleği

DP(Dead Period): Ölü Periyot

DRAM (Dynamic RAM): Dinamik RAM

DRE (Detected Recoverable Error): Sezilebilen Düzeltilebilen Hata

DUE (Detected Unrecoverable Error): Sezilebilen Düzeltilemeyen Hata

ECC (Error Correction Code): Hata Doğrulama Kodu

FIT (Failure In Time): Hata Zamanı

I-Cache (Instruction Cache): Komut Önbelleği

L1 (Level 1): Birinci Seviye

L2 (Level 2): İkinci Seviye

LRU (Least Recently Used): Son Zamanlarda En Az Kullanılan

MITF (Mean Instruction to Failure): Hataya Olan Ortalama Komut

MTBF (Mean Time Between Failure): Hatalar Arası Ortalama Zaman

MTTF (Mean Time To Failure): Hataya Olan Ortalama Zaman

PC: Program Counter

SECDED (Single Error Correction Double Error Detection): Tekli Hata Düzeltme Çoklu Hata Sezme

SEU (Single Event Upsent): Tek Bitlik Bozulmalar

SER (Soft Error Rate): Soft Error Oranı

SC (Shadow Cache): Gölge Önbellek

SDC (Silent Data Corruption): Sessiz Veri Bozulması

SRAM (Static RAM): Statik RAM

VERİ ÖNBELLEĞİ GÜVENİLİRLİĞİNİN MODELLENMESİ VE GELİŞTİRİLMESİ

ÖZET

Enerji yüklü parçacığın çarpmasıyla oluşan soft errorlar, özellikle gürültülü ortamlarda bilgisayar sistemleri için önemli güvenilirlik sorunlarına neden olurlar. Geçici hatalar sonucu oluşan bu problemleri teknolojinin büyüklüğü ve agresif sızıntı kontrol mekanizmaları daha da ciddi bir hale getirir. Bu nedenle, işlemci veya bellek dizaynında soft errorlara karşı daha güvenilir geliştirme mekanizmaları sunulmalıdır. Bunu yapmak için önce soft error'ları modellemeliyiz, daha sonra modele dayanarak değişik güvenilirlik geliştirme tekniklerinin maliyet/güvenilirlik karşılaştırmasını yapmalı ve bu tekniklerden sistem gereksinimlerini sağlayan bir tanesini seçmeliyiz.

Günümüzde önbellekler çip üzerinde çok büyük bir oranda yer kapladığından ve gelecek tasarımlarda da bu yerin büyüyeceği varsayıldığından, bilgisayar sisteminin diğer bileşenlerine kıyasla önbellekler soft errorlara karşı daha kolay etkilenir hale gelmişlerdir. Bu çalışmada, önce L1 veri önbelleği için soft error modeline odaklanacağız, sonra değişik güvenilirlik geliştirme tekniklerini araştıracağız. Bu esnada önbellekte oluşan bir hatanın programın sonuç çıktısında görülmesinin olasılığını gösteren AVFC (Architectural Vulnerability Factor for Caches) ölçüsünden bahsedeceğiz. Oluşturacağımız modele dayanarak, soft errorların varlığında güvenilirliği arttıran 3 mimarisel yaklaşım önereceğiz. İlk yaklaşımımız, soft errorın bellek hiyerarşisinde daha alt seviyelere yayılmadan önlenmesidir. Bu kirli blokların (dirty block) L1 önbelleğinden L2 önbelleğine geri yazılırken değişmemiş veri kelimelerinin (data words) geriye yazılmamalarıyla

sağlanır. İkinci yaklaşım, önbellek bloklarının herhangi bir soft errora yakalanma şansını azaltmaya yöneliktir ve etkilenir periyotlarını azaltmak için seçilmiş önbellek bloklarını iptal etme şeklinde gerçekleştirilmeye çalışılır. Deney sonuçları, bu iki yaklaşımın da AVFC ölçüsüne dayanarak L1 veri önbelleğindeki soft errorları hafifletici etki yaptığını göstermektedir. Özellikle, ilk yaklaşımı kullanarak hiçbir performans kaybına sebep olmadan AVFC ölçüsünü %32 geliştirmek mümkündür. Diğer taraftan, ikinci yaklaşımla önbellek bloklarının ne kadar agresifçe iptal edileceğine bağlı olarak, performans azalımı %0 ile %21,3 arasında değişirken, AVFC ölçüsü %60 ila %97 arasında gelişir. Önbellek bloklarının iptal edilmesine bağlı olarak yaşanan performans yükünü azaltmak için önerdiğimiz üçüncü yaklaşım önceden getirme ile iptal edilmiş bloğun yeni bir kopyasının önbelleğe getirilmeye çalışılmasıdır. Deney sonuçlarımız gösteriyor ki, bu yaklaşım sayesinde deney testleri içindeki tüm uygulamalar için performans maliyetini (yükünü) %1 den daha az azaltır. Ancak bunun için ikinci yaklaşımdaki güvenilirlik gelişiminden tolere edilebilir ölçüde vazgeçmek gerekir.

Anahtar sözcükler: soft error, güvenilirlik, AVFC, veri önbelleği

MODELING AND IMPROVING DATA CACHE RELIABILITY

ABSTRACT

Soft errors arising from energetic particle strikes pose a significant reliability concern for computing systems, especially for those running in noisy environments. Technology scaling and aggressive leakage control mechanisms make the problem caused by these transient errors even more severe. Therefore, it is very important to employ reliability enhancing mechanism in processor/memory designs to protect them against soft errors. To do so, we first need to model soft errors, and then study cost/reliability tradeoffs among various reliability enhancing techniques based on the model so that system requirements could be met.

Since cache memories take the largest fraction of on-chip real estate today and their share is expected to continue to grow in future designs, they are more vulnerable to soft errors, as compared to many other components of computing system. At this work, we first focus on a soft error model for L1 data caches, and then explore different reliability enhancing mechanisms. More specifically, we define a metric called AVFC (Architectural Vulnerability Factor for Caches), which represents the probability with which a fault in the cache be visible in the final output of the program. Based on this model we then propose three architectural schemes for improving reliability in the existence of soft errors. Our first scheme prevents an error from propagating to the lower levels in the memory hierarchy by not forwarding the unmodified data words of a dirty cache block to the L2 cache when the dirty block is to be replaced. The second scheme proposed selectively invalidates cache blocks to reduce their vulnerable periods, decreasing their chances of catching any soft errors. Based on the AVFC metric, our experimental results show that these two schemes are very effective in alleviating soft errors in the L1 data cache. Specifically, by using our first scheme, it is possible to improve the AVFC metric by 32% without any performance loss. On the other hand, the second scheme enhances the AVFC metric between 60% and 97%, at the cost of a performance degradation which varies from 0% to 21.3%, depending on how aggressively the cache blocks are

invalidated. To reduce the performance overhead caused by cache block invalidation, we also propose a third scheme which tries to bring a fresh copy of the invalidated block into the cache via prefetching. Our experimental results indicate that this scheme can reduce the performance overheads to less than 1% for all applications in experimental suite, at the cost of giving up a tolerable portion of the reliability enhancement the second scheme achieves.

Keywords: soft errors, cache reliability, AVFC, data cache

İÇERİK **Sayfa**

YÜKSEK LİSANS TEZİ SINAV SONUÇ BELGESİ i
TEŞEKKÜR ii
SİMGELER VE KISALTMALAR iii
ÖZET iv
ABSTRACT vi

BÖLÜM 1 – GİRİŞ 1

BÖLÜM 2 – ÖNBELLEKLERE İÇİN SOFT ERROR MODELİ 6
2.1. Hata Üretimi.....8
2.2. Hata Yayılımı 8
2.3. AVFC.....9
2.4. Örnek.....11

BÖLÜM 3 – DENEYSEL DÜZENEK VE YAKLAŞIMLARIMIZA GÖRE SONUÇLAR 13
3.1. Simülasyon Ortamı 13
3.2. Temel Durum İçin AVFC Değerleri.....15
3.3. Yaklaşımımıza Göre Sonuçlar.....15
3.3.1. Sadece Değişmiş Kelimeleri Geri Yazmanın Etkisi 19
3.3.2. İnaktif (Aktif Olmayan) Satırların Geçersiz Kılınmasının Etkisi 22
3.3.3. İnaktif Satırların Yenilenmesinin Etkisi 28

BÖLÜM 4 – SONUÇ VE TARTIŞMA 32

KAYNAKLAR …….34

Tablolar..... I
Şekiller II
Yaşam Öyküsü.....……..IV

BÖLÜM 1

GİRİŞ

Donanımda oluşan geçici hatalar, alfa parçacıkları gibi paketleme materyallerinden yayılan ışımalar ve kozmik ışınlardan gelen nötron parçacıkların, işlemcinin güvenilir bir biçimde çalışmasını engellemesi sonucu oluşur. Ziegler (1996)'e göre bu olaylar nedeniyle toplanan yük latch, SRAM hücreleri, kapılar gibi mantık devrelerinin durumunu değiştirebilir, yanlış çıktılar doğurabilir veya uygulama veya sistem yazılımı çökmelerine neden olabilir. Transistörlerin boyutlarını küçültmek (daha küçük kapasitans), daha az kaynak ve eşik voltajı ve daha yüksek saat hızlarında işlem yapmak, işlemcilerin geçici hatalardan yani soft errorlardan daha çabuk etkilenmesine neden olur. Önbellekler çip üzerinde önemli bir yer teşkil ettikleri için, CPU'nun diğer kısımlarına oranla önbellekler soft errorlara karşı daha duyarlıdırlar. Önbellekler CPU ya daha yakın olduklarından yazmaç dosyaları üzerinden soft errorlar diğer bileşenlere de kolayca yayılabilirler. Bu da önbelleklerdeki geçici hataları önlemeye çalışmak için başka bir nedendir.

DRAM üreticilerinin, bit başına düşen SER (soft error rate)'i geliştirmek için uyguladıkları yöntemler, SRAM cihazları için geçerli değildir ve gerçekte SRAM hücresi başına düşen soft error oranı ya aynı kalmaktadır yada çok az gelişmektedir (Mukherjee ve diğ., 2005). Bir yandan, VLSI teknolojisindeki gelişmeler önbellek güvenilirliğinde pozitif etki yaratabilir. Örneğin, transistör boyutlarının küçülmesi parçacıkların belirli bir transistöre çarpma olasılığını düşürür. Diğer taraftan işlemci teknolojisindeki yeni gelişmeler (daha küçük kapasitanslar, daha az kaynak ve eşik voltajı ve daha yüksek saat hızlarında işlemler) önbellek güvenilirliğini tam tersi yönde etkileyebilir, önbelleği hatalara karşı çok daha hassas hale getirebilir. Örnek olarak, eğer ön bellek düşük sızıntı için optimize edildiyse daha az miktardaki yük hataya neden olabilir. Bu ters etkiler kabaca birbirini dengeler ve transistör başına düşen hata oranı hemen hemen aynı kalır. Sonuç olarak önbelleklerin hata oranı, üzerlerinde bulunan transistör sayısıyla doğru orantılı olarak artar (Hareland ve diğ., 2001; Karnik ve diğ., 2001). Ayrıca teknolojinin gelişmesi doğrultusunda değişik

devre tipleri için SER ölçüsünü araştıran araştırmalarda bulunmaktadır (Karnik ve diğ., 2004; Nguyen ve diğ., 2003; Shivakumar ve diğ., 2002; Irom ve diğ., 2002).

Pratikte, güvenilirliği belirleyen ve sık kullanılan iki ölçü vardır: FIT (Failure in Time) ve MTBF (Mean Time Between Failure). FIT, 1 milyar saatte yakalanan hata sayısıdır. MTFB, FIT'in tersidir (FIT = 1/MTFB).

Soft errorlar temel olarak iki kategoriye ayrılırlar: saptanan (detected) ve saptanamayan (undetected) soft errorlar. Saptanamayan hatalar aynı zamanda Silent Data Corruption (SDC) olarak da bilinirler, en sinsi hatalardır ve sistemin yanlış çıktılar üretmesine neden olurlar. Saptanan hatalarınsa iki tipi vardır: Saptanan kurtarılabilir hatalar (Detected Recoverable Error - DRE) ve saptanan kurtarılamayan hatalar (Detected Unrecoverable Errors - DUE). DRE'ler bazı güvenlik mekanizmaları tarafından hem saptanabilir hem de düzeltilebilir. DUE'ler ise saptanabilir fakat düzeltilemez. Bu tip hatalarla uğraşmanın en iyi yolu saptandıkları yerde programın çalışmasının durdurulmasıdır. SDC ve DUE oranları da SER'de olduğu gibi FIT/MTBF olarak tanımlanırlar. İşlemci üreticileri genel olarak ürünlerinin SER bütçelerini hedef pazarların güvenilirlik ihtiyaçlarını karşılayabilecek şekilde ayarlarlar. Örneğin; IBM'in Power4 sistemi için hedefleri 114 SDC FIT (1000 yıl MTTF), 4566 sistem çökerten DUE FIT (25 yıl MTTF) ve 11415 uygulama çökerten DUE FIT (10 yıl MTTF)'dir (Mukherjee ve diğ., 2005).

Bir çipin SER'ini azaltan pek çok teknik önerilirken, bu teknikler performans, yer, güç veya dizayn zamanı gibi ekstra maliyetlere neden olurlar. Donanım seviyesinde güvenilirliği arttırma teknikleri üç ana kategoriye ayrılır: süreç teknolojisi, devre ve mimari çözümleri (Mukherjee ve diğ., 2005). Süreç teknolojisi tabanlı çözümlerde cihazı hatalardan korumak için silikon-yalıtkan SOI kullanılır. SOI ile kaplanan cihaz daha ince silikon katmanı nedeniyle radyasyondan daha az yük çeker ve çarpma mantıksal devrenin durumunu daha az değiştirir. IBM'in raporlarına göre kısmen tüketilmiş/boşaltılmış SOI teknolojisi kullanmak SRAM cihazlarında SER'in beş kat azalmasını sağlar (Cannon ve diğ., 2004). Devre seviyesinde çözümlerde, hücre kapasitansı ve/veya kaynak voltajı gibi cihaz

parametrelerini ayarlayarak radyasyondan korunaklı hücreler oluşturulur. Bu durumda da hücrenin değişmesi için gereken minimum yük artar, böylece SER değeri azalır. Ancak, tasarımda radyasyon korumalı hücreler kullanmak, oldukça büyük yer ve güç masrafına yol açar. Süreç ve devre seviyesindeki çözümler bu çalışmanın ilgi alanı olmadığından, bu çalışmada bunlardan daha fazla bahsetmeyeceğiz. Ancak bu iki kategorideki çözümlerin, mimari çözümleriyle birlikte kullanılarak güvenilirliği daha da artırılabileceği göz ardı edilmemelidir.

Önbelleğin güvenilirliğini arttırmak için kullanılan mimari çözümleri, eşlik kontrolü ve hata doğrulama kodu (Error Correcting Codes - ECC) (Pradhan, 2003), Π bit (Weaver ve diğ., 2004), N birimsel artıklık (N Modular Redundancy - NMR) gibi bileşenlerin replikalarına dayanan tutarlılık kontrollerini içerir. Byte-parity mekanizması yani her 8 bit veriye ekstra 1 bit ekleyen mekanizma, önbelleklerde hata yakalama için sıkça kullanılır. Eşlik basit ve güçlü olmasına rağmen %12,5 alan masrafına neden olur. Dahası tek sayılı hataları yakalayabilir ancak onları düzeltemez. Sonuç olarak, özellikle yüksek seviyede güvenilirlik isteniyorsa veya gürültülü ortamlarda çalışılıyorsa çoğu sistem için yalnız başına geçerli bir çözüm olmaktan uzaktır. Chen ve Hsiao (1992)'nın yaptıkları çalışmalarda da bahsettikleri üzere popüler bir ECC olan SECDED, 64-bit giriş için ekstra 8-bit kullanır. Eşlik bitinin tersine SECDED'in gerçekleştirilmesi daha zordur ve tek bitlik hataları düzeltebilir. Ancak önbelleğe erişimi geciktirir, özellikle yüksek saat hızlarında çalışan işlemciler için, önbelleğe erişimin 1 devirde tamamlanmasını engeller ve güç tüketimini arttırır. NMR tabanlı teknolojiler, kesinlikle veri güvenirliği sağlamasına rağmen, daha fazla yer masrafına neden olurlar. Örneğin; Carmichael (2001)'in bahsettiği gibi Triple Modular Redundancy'e bağlı alan masrafı %200'dür, bu da alan açısından kısıtlı olan sistemler için oldukça pahalıdır. Π bit, yanlış DUE'ları yok etmeye yarayan hata yayılım mekanizmasıdır (DUE hataları çıktıyı etkilemez.). Hata yakalanır yakalanmaz makine denetimli istisna mekanizmasını harekete geçirmek yerine, etkilenen komutların Π biti set edilir. Daha sonra pipelinenın commit safhasında, eğer hatalı komutun yanlış yoldaki bir komut olduğuna karar verilirse, Π bit uyarılır ve yanlış DUE olayı önlenir. Aynı komutun birden fazla kopyasını çalıştırmaya dayalı gereksiz çoklu kullanım teknikleri, hata yakalama ve düzeltme

için önerilmiştir (Gomaa ve diğ., 2003; Vijaykumar ve diğ., 2002; Ray ve diğ., 2001; Reinhardt ve Mukherjee, 2000). Ancak, bu teknikler işlemcinin kaynakları üzerine ekstra yük bindirir. Kaynak fazlalığının azaltılmasını Kumar ve Aggarwal (2006) çalışmışlardır. Pipelinenın boşta bekleyen kaynaklarından ve L2 ıskalarından faydalanılarak issue kuyruğundaki soft error oranı ve performans arasındaki ilişkileri belirleyen bazı çalışmalar Gomma ve Vijaykumar (2005) tarafından yapılmıştır. Çeşitli donanım birimlerinin AVF' sini ölçmek için hata enjeksiyonuna dayalı çeşitli çalışmalarda mevcuttur (Kim ve Somani, 2002; Wang ve Patel, 2003). Bu fikirlerden birini uygulayarak işlemcinin hataya en dayanıksız kısmını belirleyen Wang ve diğ. (2004) bu kısmı geçici hatalara karşı koruyan teknikler önermişlerdir.

Bu çalışmada önce önbelleklerde geçici hataları modelleyen bir yaklaşım önereceğiz. Yaklaşımımızı L1 veri önbelleğinin içeriği açısından anlatmamıza rağmen bu yaklaşımı biraz değiştirmek suretiyle L2/L3 önbelleklerinin yanında L1 komut önbelleği için de uygulamak zor değildir. Modelimize dayalı olarak, fikirsel olarak Mukherjee ve diğ. (2003) çalışmasında tanıtılan AVF' ye oldukça benzeyen bir kavram olan önbellekler için mimari bozulabilirlik faktörünü (Architectural Vulnerability Factor for Caches-AVFC) bu çalışmada tanımladık. AVFC, önbellek bileşeninde oluşan bu hatanın yüzde kaç ihtimalle programın çıktısında görülebilir bir hataya sebep olabileceğini belirler. Daha sonra değişik veri güvenirliği geliştirme yöntemlerini inceleyeceğiz. Şunu vurgulamakta yarar var, önerdiğimiz yöntemlerin amacı soft errorları yakalamak veya onları düzeltmek değildir. Yaklaşımlarımız, ECC ve parity gibi klasik hatadan korunma yaklaşımlarından bağımsızdır ve önbelleğin soft errorları yakalama şansını azaltmaya çalışır. Daha açıkça söylemek gerekirse, önerdiğimiz yaklaşımlar, parity ve ECC tabanlı yaklaşımları değiştirmek anlamında değildir. Aksine, veri güvenilirliğinin maksimize edilmesinde geleneksel hata koruma yaklaşımlarıyla birlikte kullanılması kastedilmektedir. İlk yaklaşımımız, yer değiştirme işlemi sırasında tüm satır yerine önbellek satırının değişmiş veri kelimesini geri yazarak, hatanın hiyerarşide bir sonraki seviyedeki önbelleğe yayılmasını azaltmaktır. İkinci yaklaşımımız, geçici hatalara yakalanma şansını azaltmak için hareketsiz önbellek satırlarını geçersiz kılmaktır. İlişkili üçüncü yaklaşımda, hareketsiz önbellek satırlarını geçersiz kıldığımızda oluşan performans

kaybını azaltmak için geçersiz kılmayı takip eden önceden getirme ile hareketsiz önbellek satırının yeni bir kopyasını önbelleğe getirmekteyiz. Bu üç yöntemi, geçici hata modelimizi ve Spec2000'nin kodlarını kullanarak değerlendireceğiz. Deneysel gözlemlerimize göre, ilk yaklaşım L1 önbelleğinden L2 önbelleğine hataların yayılmasını %57 azaltmakta ve herhangi bir performans kaybı olmadan veri önbelleği tutarlılığını %32 geliştirmektedir. Diğer yandan, ikinci yaklaşım önbellek bloklarının ne kadar sıklıkta geçersiz kılındığına bağlı olarak %0'dan %21,3'e değişen oranda performans azalması sağlarken %60 - %97 arasında veri güvenilirliğini artırmaktadır. Üçüncü yaklaşımımızla veri güvenilirliğinin arttırılmasından göz ardı edilebilir bir oranda vazgeçilmesiyle, performans kaybını %1'den aşağıya düşürmek mümkün olabilmektedir.

BÖLÜM 2

ÖNBELLEKLER İÇİN SOFT ERROR MODELİ

Bu bölümde önbellekler için mimari seviyesinde tanımladığımız soft error modelimizi açıklayacağız. Güvenilir sistem tasarımcıları açısından önbellekler gibi soft errorlara karşı daha duyarlı sistem bileşenleri için mimari seviyesinde soft errorların analizine izin veren modeller son derece önemlidir. Bunun iki ana nedeni vardır. Birincisi; bu modelleri kullanarak değişik sistem bileşenleri için soft errorları daha iyi anlayabilirler. İkincisi, bu modeller tasarımcılara, değişik güvenilirlik arttırma tekniklerini değerlendirirken uygun fiyat/güvenilirlik değerlendirmesi yapma, sistem gereksinimleri/hedefleri ve bütçe sınırlamalarına göre en iyi yaklaşımı seçme imkânı sunar.

Yakın zamanda, soft errorların modellenmesini mimari seviyesinde yapan çalışmalar vardı. Mukherjee ve diğ. (2003), mimarisel olarak doğru çalışma (Architecturally Correct Execution – ACE) bitlerine dayanan mimari bozulabilirlik faktörü (Architecturally Vulnerability Factor - AVF) fikrini önermişlerdir. Bu modelde, işlemcinin durum bitleri iki gruba ayrılır: ACE ve ACE-olmayan bitler. ACE-olmayan bitler de iki alt kümeye ayrılır: mikro mimarisel ACE-olmayan bitler ve mimarisel ACE-olmayan bitler. ACE bitler programın doğru çalışması için doğru olması gereken bitlerdir. Yani sadece ACE bitlerini etkileyen hatalar, hatalı çıktı üretilmesine neden olur. Burada önemli olan hangi bitlerin ACE, hangi bitlerin ACE-olmayan olduğuna nasıl karar verileceğidir. Bu amaç için her bir komut için pipeline safhalarının izi sürülür. Örneğin, yanlış speküle edilmiş bir dallanma tahmini sonucu olarak etkilenen işlemci durum bitleri ACE-olmayan bitler olarak kabul edilir. AVF ölçüsü temelde veri yoluna odaklanır ve sonuç program çıktısında görünür hata ile karşılaşılabilen işlemci bileşenlerindeki hatanın olasılığı olarak tanımlanır.

Li ve diğ.(2005), soft errorları mimari seviyesinde modelleyen ve analiz eden SoftArch adında bir tool önermiştir. Hata üretimi ve yayılımının olasılığına dayanır. Mukherjee ve diğ. (2003) çalışmasında detayları açıklandığı gibi bir komuttan o

komutu izleyen komutlara soft error yayılımını modellemek için DFG-benzeri bir yaklaşım kullanılır. Bu amaçla her veri değeri için temel hata kümesi tanımlanır. Temel hata kümesindeki her eleman, soft errorın oluşunu gösteren temel hata olayıdır. Örneğin; her ne zaman önbellek gibi bir depo bölgesinden bir değer okunsa, son okunduğu zamandan o ana kadar geçen zamanda hata oluşma olasılığını gösteren temel hata olayı, temel hata kümesine eklenir.

Sridharan ve diğ (2006), bellekler için bozulabilirlik faktörü (Vulnerability Factor) adını verdiği bir ölçü tanımlamıştır. Bizim modelimize zıt olarak o model bir önbellek bloğundan diğerine hata yayılımını göz önünde bulundurmaz.

Bu temel modelleme çabalarına kıyasla, bizim modelimiz bazı hataların programın sonuç çıktısı üzerinde etkisi olmadığı gözlemlerine dayanarak önbelleklerdeki soft errorları hedef alır. Bunun iki nedeni vardır. İlki; bazı önbellek hatalarının, önbelleğin içinde kaldığı ve diğer sistem bileşenlerine yayılmadığı durumudur ki buna maskelenmiş hatalar (masked error) denir. Örneğin; geçersiz veri tutulan önbellek bloğunda veya değiştirilmemiş bloğun ulaşılmayan kısmında bir hata oluştuğunda diğer sistem bileşenlerine bu hata yayılmaz, böylece izole olmuş olur ve programın sonuç çıktısını etkilemez. İkincisi; önbellek hataları diğer sistem bileşenlerine yayılsa bile bazen programın sonuç çıktısında görünür bir hataya neden olmaz. Bu tip hatalara iyi huylu hatalar (benign error) denir. Örneğin; ölü/yanlış speküle edilmiş bir komuta veri girdisi sağlayan veri önbellek bloğundaki hata sistem güvenilirliğini etkilemez.

Modelimizin iki bileşeni var: hata üretimi ve hata yayılımı. Her önbellek satırını kelimeler kümesi olarak düşünüyoruz. Kelime (word) terimi, veri tutan bir deponun herhangi bir birimi için kullanılır, bu yüzden büyüklüğü tuttuğu veriye bağlıdır. Örneğin; bir kelime C programlama dilinde, temel veri tipinden olan float veya int yada bir dizinin elemanına denk gelebilir. d verisini, l satırında tutan kelime $w_{l,d}$ şeklinde gösterilir. Bu çalışmanın devamında blok ve satır terimleri birbiri yerine kullanılacaktır. Mimari modelimizdeki her kelimenin iki özelliği vardır: Hata Üretimi (Error Generation- EG) ve Hata Yayılım Kümesi (Error Propagation Set -

EPS). *EG*, hata oluşma olayını gösterir ve büyüklüğü |*EG*| ile gösterilir. Her iki özelliği de kısa bir zaman sonra açıklayacağız. *EG* ve *EPS* ye bağlı olarak, biz daha sonra bir önbellek yapısındaki bir hatanın programın son çıktısında görülebilen bir hataya yol açma ihtimalini gösteren AVFC metriğini tanımlayacağız.

2.1. Hata Üretimi

Yüksek enerjili bir parçacık çarptığında, toplanan yük veriyi depolayan devrelerin çıkışlarını değiştirebilecek yeterli bir miktarda olduğu zaman ilgili veride soft error oluşur. Modelimizde SRAM önbelleğindeki ham hataların sabit bir hızda olduğunu (Li ve diğ., 2005) ve rastgele düzgün dağıldığını varsayıyoruz (Mukherjee ve diğ., 2003). Bu nedenle veride hata oluşma olasılığı, veri büyüklüğüyle ve verinin parçacık çarpmalarına maruz kaldığı zaman aralığıyla doğru orantılıdır. Kelime '*w*' için |*EG*| (*l* satırında bulunan ve *d* verisini tutan);

$$\left|EG(w_{l,d})\right| = boyut(w_{l,d}) \times maruz\ kalma\ süresi(w_{l,d}).$$

olarak ifade edilir.

2.2. Hata Yayılımı

Programın çalışması sırasında, değerler önbellekten yazmaçlara (register) okunur ve bu değerler tekrar önbelleğe yazılacak olan yeni değerlerin hesaplanmasında kullanılır. Orijinal değerlerdeki soft errorların yazmaç dosyalarından önbelleğe kolayca yayılabileceğini biliyoruz. Bu hatalar eğer düzeltilmezse, önbellek hiyerarşisinde daha alt seviyelere kirli veri üzerinden yayılabilir. *l* satırında tutulan değiştirilmiş her bir veri *d* için $EPS(w_{l,d})$'yi tanımlanır. $EPS(w_{l,d})$, $w_{l,d}$'deki değerin hesaplanmasında yer alan verilerin hata yayılımına etkilerini ifade eden bir kümedir. Bunu şu ifade ile düşünelim: $v_1 = v_2 + v_3$, v_2 ve v_3 önbellekte aynı *l* satırında tutulan değerleri göstersin. v_2 ve v_3 sırasıyla R_2 ve R_3 yazmaçlarıyla işlemciye alındığını ve toplama işleminin sonucunun R_1 yazmacında tutulduğunu varsayalım. Eğer önbellekteki v_2 hatalıysa R_2 ve dolayısıyla R_1 bozulur. Sonuç olarak R_1, v_1'e karşı gelen yere yazıldığından, v_1 hatalı bir değer tutmuş olacaktır. Diğer bir deyişle v_2 değerindeki hata R_2 ve R_1 yazmaçları vasıtasıyla v_1'e yayılmış olur. Bu örnekte $w_{l,v1}$ kelimesinin *EPS*'si,

$$EPS(w_{1,v1}) = \{EG(w_{l,v2})\} \cup \{EG(w_{l,v3})\} \cup EPS(w_{l,v2}) \cup EPS(w_{l,v3})$$

olarak hesaplanır. Bu, diğer değerlerden *v2/v3*'e yayılan hataların da *v1*'in güvenilirliğini etkilediği anlamına gelir.

Her kirli önbellek satırındaki her bir kelime için bir *EPS* varsayıyoruz. Bu *EPS*, satırın önbelleğe getirilmesiyle başlayan ve önbellekten çıkarılmasıyla sonlanan zaman zarfı içerisinde ilgili kelimeye yayılan hataların etkisini ifade eder. Böylece bir önbellek satırının *EPS*'si o satırda yer alan kelimelerin *EPS* kümelerinin birleşimi olarak düşünülebilir. Programın veri akış grafı, yeni bir değerin hesaplanmasında hangi değerlerin kullanıldığını tespit etmek suretiyle hata yayılımının yönünü belirlemede bize yardımcı olur.

2.3. AVFC

Ne maskelenmiş ne de iyi huylu olan hataları ciddi hatalar olarak tanımlıyoruz. Bunlar programın doğru çalışmasında güvenilirlik problemi oluşturan soft error türleridir. Bu çalışmada ciddi olarak düşünülebilecek iki tip hata üzerine odaklanacağız. İlk tip, değişmiş (kirli) önbellek bloklarındaki hataları içerir. Bu noktada her değişmiş bloğun programın görünür çıktısını değiştirdiğini varsayıyoruz. İkinci tip hatalar, programın çalışmasının kontrol akışını (komut dallanmasının bağlı olduğu değerler) değiştiren değerlerdeki hatalardır. Bu iki tip ciddi hataya dayanarak, önbellekteki değişmiş bloklar için toplam hata üretimi ve yayılımı (*$TEGP_c$*), dallanma komutlarını besleyen veriler için toplam hata yayılımı (Total Error Propagation - *TEP_B*) ve AVFC ölçütü aşağıdaki gibi hesaplanır:

$$TEGP_c = \sum_{\substack{yer\ değiştiren \\ kirli\ l\ satırı}} \sum_{d} \left|EG(w_{l,d})\right| + \sum_{\substack{e \in \\ EPS(w_{l,d})}} |e|$$

$$TEP_B = \sum_{b\ dallanma} \sum_{d} \sum_{\substack{e \in \\ EPS(w_{l,d})}} |e|$$

$$AVFC = \frac{TEGP_C + TEP_B}{\text{Önbellek boyutu} \times \text{Çalışma Devri}}$$

İlk denklem, önbellekteki hataların etkilerini göstermektedir. Bu denklemden de kolayca görülebileceği gibi sadece değişmiş/kirli önbellek bloklarını etkileyen hatalar dikkate alınmaktadır. Değişmiş önbellek bloğundaki hatalar için iki kaynak vardır. İlki hata oluşumu, ikincisi diğer bir bozulmuş önbellek bloğundan hatanın yayılmasıdır. Tüm bunların ilk denklemde içerildiğine dikkat edelim. Değiştirilen kirli önbellek satırı *l*'nin hata bütçesi, barındırdığı her kelimenin hataya katkısı düşünülerek hesaplanır. Burada |*e*|, *l* satırında *d* verisini tutan kelimelerin *EPS*'nin bir elemanı olan *e*'nin byte x devir cinsinden değerini gösterir. İkinci denklem, çalışma akışını değiştirebilen değerlerdeki hataların etkilerini gösterir. Veri yolunda (CPU'daki hesaplamalarda) hata oluşumu olmadığını kabul ettiğimiz için ikinci denklemde dallanmaların hesaplanmasında yer alan değerlerde CPU' ya getirildikten sonra oluşabilecek hataları dikkate almamaktayız. Yani, bu değerlerdeki hatalar sadece önbellekteki değerlerden yayılan hatalardır. İlk iki denkleme dayanarak, *AVFC* üçüncü denklemde gösterilmiştir. *Çalışma devri* programın çalışma zamanını gösterir.

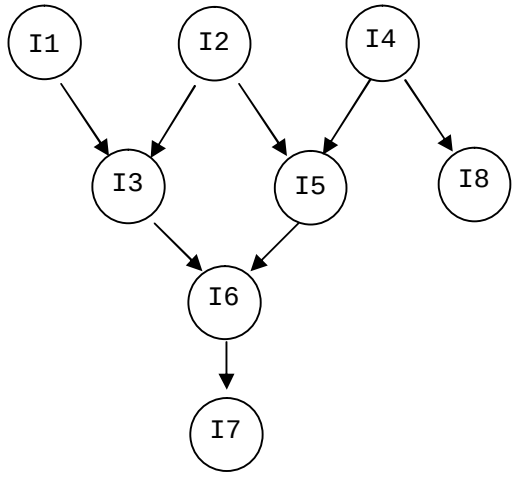
Modelimizde, L2 önbelleğinde hata olmadığını ve L2 önbelleğinden L1 önbelleğine veri akışının her zaman hatasız olduğunu varsayıyoruz (L1 önbelleğine daha önceki bir ziyareti sırasında güncellenmiş olsa bile). L2 den L1'e veri akışının daha önceki L1 önbellek ziyaretlerinden dolayı soft error taşıma olasılığını göz önünde bulunduracak şekilde modelimizi değiştirmek zor değildir.

2.4. Örnek

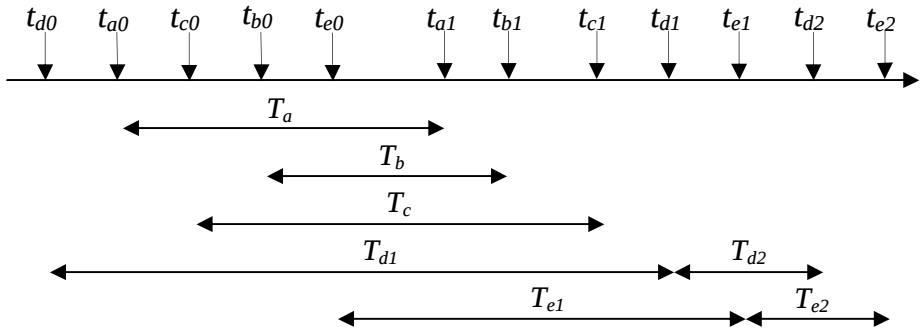
```
d = (a+b) x (b-c)
e = c


load R1, <a>      //I1
load R2, <b>      //I2
add R4, R1, R2    //I3
load R3, <c>      //I4
sub R5, R2, R3    //I5
mult R4, R4, R5   //I6
str R4, <d>       //I7
str R3, <e>       //I8
```

Şekil 1- (Üstte) Kaynak kod parçası ve (altta) karşılık gelen birleştirici dil kodu.



Şekil 2 – Şekil 1'deki kod parçacığının veri akış diagramı (DFG) .



Şekil 3- Şekil 1'deki kod parçasının veri işleyişi için önbelleğe erişimi de gösteren zaman tablosu.

Hata üretimi ve yayılımı modelinin nasıl gerçekleneceğini bir örnek üzerinde görelim. Bu örnekte, Şekil 1'deki kod parçasına ve onun birleştirici dil koduna odaklanacağız. Şekil 2, Şekil 1'deki kodun veri akış diyagramıdır (DFG). L1 önbelleğine yaptıkları ziyaretlere göre koddaki verilerin zamanlama çizelgesi Şekil 3'te verilmektedir. Bir önbellek satırında tutulan her a değişkeninin t_{a0} zamanında önbelleğe getirildiğini ve önbellekten yazmaçlara t_{a1} zamanında yüklendiğini varsayıyoruz. Her değişkenin farklı önbellek bloğunda bulunduğunu varsaydığımızdan, kelimenin gösteriminde ikinci bir alt simge kullanmıyoruz. a verisinin soft errorlara maruz kalma süresi T_a'ya eşit olduğundan $|EG(w_a)|$ $4byte \times T_a$ ya eşit olacaktır, tabi a verisinin boyutunun 4 byte olduğu varsayımıyla. Aynı şekilde

b ve c'ninde sırasıyla T_b ve T_c periyotları boyunca soft errorlara maruz kaldığını düşünüyoruz. Böylece, bunların $|EG|$'leri $|EG(w_b)|=4\ byte \times T_b$ ve $|EG(w_c)|=4\ byte \times T_c$ olur. d değerini bulmak için kullanılan veriler dikkate alındığında, w_d kelimesi için

$$EPS = \{EG(w_a)\} \cup \{EG(w_b)\} \cup \{EG(w_b)\} \cup \{EG(w_c)\}$$

şeklinde hesaplanır. İlk ikisi *R4* yazmacı vasıtasıyla yayılır, oysa son ikisi *R5* yazmacı aracılığıyla yayılır. Burada dikkat edilmesi gereken *R1*,*R2* ve *R3* yazmaçları tarafından taşınan hata kümeleri ayrık iken, *R4* ve *R5* tarafından taşınan hata kümeleri ayrık değildir. Sonuç olarak $EPS(w_d)$ $\{EG(w_a),EG(w_b),EG(w_c)\}$'dir. $EPS(w_d)$'nin hata katkısı $|EPS(w_d)|$, $4\ byte \times (T_a+T_b+T_c)$ şeklinde hesaplanır. Eğer, d'yi tutan satırın önbelleğe t_{d0} zamanında getirildiğini, d'ye önbellekte t_{d1} zamanında yazma yapıldığını ve d'yi tutan satır t_{d2} zamanında yer değiştirmeye maruz kaldığını kabul edersek, w_d haricindeki satırın her kelimesinin $|EG|$'si $4byte \times (T_{d1}+T_{d2})$ olacaktır. Diğer yandan $|EG(w_d)|= 4byte \times T_{d2}$'dir. Yani, T_{d1} periyodu sırasında üretilen soft error, w_d için t_{d1} zamanında yapılan yazma operasyonuyla çakışır, böylece w_d'nin güvenirliğini etkilemez. Önbellek satırının 8 kelime içerdiğini (her bir kelime 4 byte içerir) kabul edersek, satırın toplam $|EG|$'si $7 \times 4byte \times (T_{d1}+T_{d2})+4byte \times T_{d2}$ olacaktır. Satırın *TEGP*'si $|EG|$ ile $|EPS(w_d)|$'nin toplamı olacaktır, bu da $28byte \times (T_{d1}+T_{d2})+4byte \times (T_{d2}+T_a+T_b+T_c)$'ye eşit olur. e'nin de d verisini tutan satırdan farklı bir satırda olduğunu düşünelim. e'yi tutan satır önbelleğe t_{e0} zamanında alınsın, `I8` komutuna karşılık gelen 'store' operasyonu t_{e1} zamanında yapılsın ve ilgili önbellek satırı t_{e2} zamanında önbellekten atılsın. Bu varsayımlar altında veriyi tutan kelimenin $|EPS|$ sinin hesaplanmasında, d, Tc periyodu için tekrar düşünülmez, çünkü d'yi barındıran satır için bu daha önce dikkate alınmıştı. Sonuç olarak, e'yi tutan satırın e dışındaki tüm kelimeleri için maruz kalma zamanı $T_{e1} + T_{e2}$ iken e verisini tutan kelime için maruz kalma zamanı T_{e2}'dir.

BÖLÜM 3

DENEYSEL DÜZENEK ve YAKLAŞIMLARIMIZA GÖRE SONUÇLAR

3.1. Simülasyon Ortamı

Daha sonra açıklayacağımız yöntemlerimizi SimpleScalar 3.0'ı değiştirerek gerçekleştirdik. SimpleScalar, oldukça geniş bir aralıktaki işlemcileri ve sistemleri simüle eden hızlı bir simülasyondur. Bu çalışmada Alpha-benzeri bir çalışma ortamını simule etmek için sim-outorder bileşenini kullandık. Tablo 1'de temel simulasyon parametrelerinin varsayılan değerlerleri gösterilmektedir. SPEC2000 suite'inden herhangi bir uygulamanın simule edilmesi çok uzun zaman aldığından biz önce 500 milyon komutu hızlı geçtik ve sonraki 500 milyon komutu simule ettik. Testlerle ilgili önemli istatistikler Tablo 2'de verilmiştir. Bu tabloda, 2. ve 3. sütunda L1 veri önbelleği erişimleri ve L1 veri önbelleği ıska sayıları, son sütunda ise yürütme devirleri verilmiştir. Bu testlerin ıska oranları parantez içinde üçüncü sütunda verilmiştir.

Tablo 1 – Deneylerimizdeki temel konfigürasyon parametreleri ve varsayılan değerleri

İşlemci Çekirdeği	
Fonksiyonel Birimler	4 Integer ve 4 FP ALU 1 Integer çarpıcı/bölücü 1 FP çarpıcı/bölücü
LSQ boyutu	64
RUU boyutu	64
Fetch/Decode/Issue Commit Width	4 komut/devir
Fetch kuyruk boyutu	4 komut
Önbellek ve Bellek Hiyerarşisi	
L1 Komut Önbelleği	32KB, 2-way, 64 byte blok 1 devir gecikme süresi
L1 Veri Önbelleği	32KB, 4-way, 64 byte blok 1 devir gecikme süresi
L2 Önbelleği	1MB birleştirilmiş, 128 byte blok 2-way, 6 devir gecikme süresi
Veri/Komut TLB	128 kayıt, tam çağrışımlı 30 devir ıska gecikme süresi
Ana bellek	100 devir gecikme süresi

Tablo 2 – Deneylerimizde kullanılan testlerin önemli özellikleri

Test	dl1 Erişimleri	dl1 Iskaları	Çalışma Devirleri
swim	167471711	15143535 (9.02%)	383766165
mgrid	183491655	6459614 (3.52%)	257632645
applu	190957116	11129182 (5.81%)	417994656
galgel	233190000	8224918 (3.52%)	873411254
equake	179382938	31959 (0.02%)	208719213
lucas	87103142	930199 (1.08%)	254742657
gzip	170770857	2912553 (1.73%)	257665291
gcc	282139139	9343024 (3.30%)	344464804
mcf	177339203	127964 (0.07%)	211505810
crafty	192464835	2685017 (1.35%)	296149344
bzip2	182104350	3885913 (2.14%)	244955485
twolf	171800466	10098842 (5.81%)	414627048

Önce, sonuçları yorumlamamıza yardımcı olacak bazı parametrelerden bahsedelim. Önbellek satırının aktif periyodu (AP) ile satırın önbelleğe getirildiği andan en son erişildiği zamana kadar olan aralık kastedilmiştir. Satırın ölü (dead) periyodu (DP), ilgili satırın son erişildiği andan başlar ve yer değiştirmek için kurban seçildiği ana kadar devam eder. Son olarak önbellek satırının hareketsiz (dormant) periyodu (DRP), iki ardışık erişim arasındaki zaman aralığıdır. AP genellikle birden fazla DRP içerirken, DP AP tamamlandıktan sonra başlar.

Bu tanımlara dayanarak, geçici hataları iki gruba ayırabiliriz. İlk grup, AP'ler sırasında gerçekleşir ve bir satırdan diğerine yayılır. Bu hata katkısına aktif periyot hata katkısı (Active Period Error Contribution - APEC) denir. İkincisi, önbellek satırlarının ölü periyotları sırasında önbellek satırlarında oluşan hatalardır. Bu hata katkısına da ölü periyot hata katkısı (Dead Period Error Contribution - DPEC) denir. Bu tanımlara dayanarak AVFC ölçüsü:

$$AVFC = \frac{APEC + DPEC}{\text{Önbellek boyutu} \times \text{Çalışma Devri}}$$

olarak ifade edilir.

3.2. Temel Durum İçin AVFC Değerleri

Tablo 3'te, temel durum için AVFC değerleri verilmiştir. Temel durumdan, soft errorlara karşı herhangi bir mekanizma kullanmadığımız durumu ifade ediyoruz. Tablodan görebildiğimiz gibi her benchmark için L1 veri önbelleğinde oluşan hataların önemli bir kısmının uygulamanın doğru olarak çalışmasında herhangi bir sıkıntıya sebep olmamaktadır. Daha ayrıntılı açıklamak gerekirse, ortalama olarak hataların %83'ü bu uygulamaların çalışmasında maskelenir. Ancak, bu geçici hataların önbellekte bertaraf edilmesinin gereksiz olduğuna işaret etmez. Aksine, önbellek yapısı tipik olarak çok büyük olduğu için hatalara maruz kalma şansı oldukça yüksektir; bu da önbellek güvenilirlik arttırma mekanizmalarının kullanılma gereksinimini zorunlu kılar. Bir sonraki bölümde anlatılacak olan yaklaşımların ortak amacı Tablo 3'te verilen temel AVFC değerlerinin azaltılmasıdır.

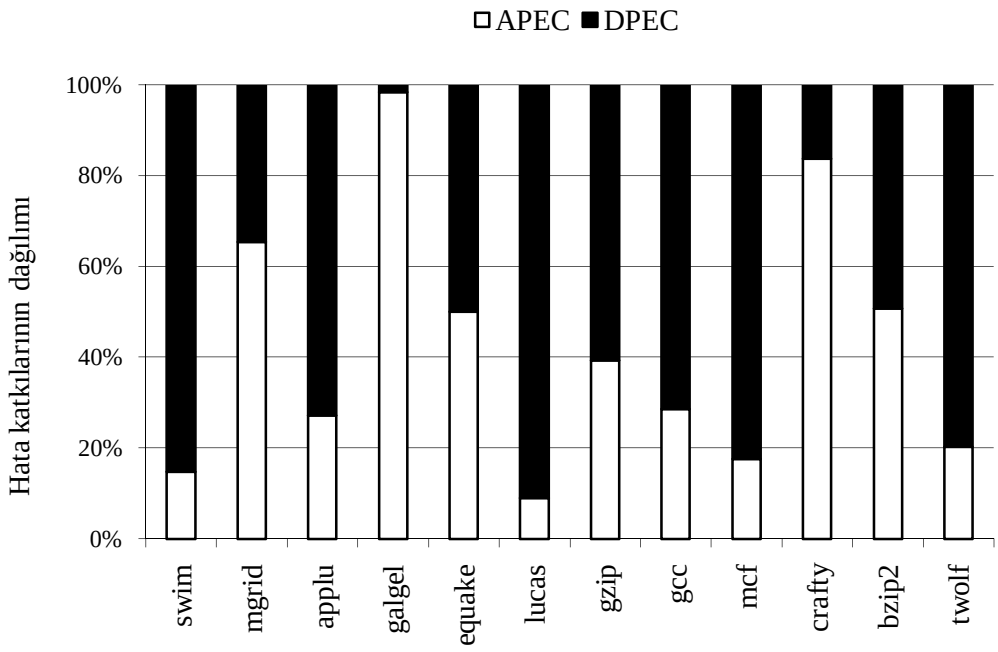
Tablo 3 – Temel durum (soft errorları azaltmaya yönelik bir mekanizma kullanılmadığı durum) için kodların AVFC'leri

Test	swim	mgrid	applu	galgel	equake	lucas	gzip	gcc	mcf	crafty	bzip2	twolf
AVFC	0.11	0.14	0.11	0.19	0.26	0.24	0.18	0.13	0.24	0.17	0.12	0.12

3.3. Yaklaşımlarımıza Göre Sonuçlar

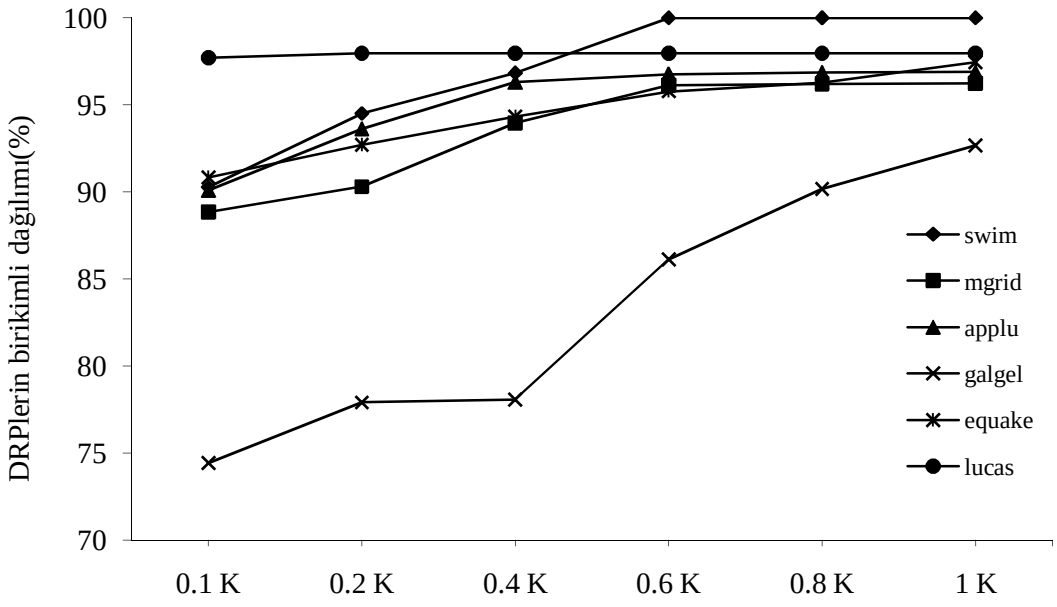
Bu bölümde önerdiğimiz üç yaklaşımları açıklayacak ve deneysel olarak onları değerlendireceğiz. Bütün sonuçlar, Tablo 3'te verilen temel durum AVCF değerlerine göre bağıl (göreceli) olarak verilecektir. Transient errorların hem önbellek satırları arasında hem de zamanda uniform olarak oluştuğunu varsaydığımızdan, bizim yöntemlerimiz tarafından elde edilen AVFC kazançları tespit edilen/düzeltilen transient error sayısında da aynı oranda kazançlara yol açar. Bu nedenle, sadece AVFC değerlerindeki gelişmeyi göstereceğiz. Her test için AVFC'nin APEC ve DPEC kısımlarının katkısı Şekil 4'te gösterilmiştir. Bu şekilden, kolayca görüldüğü gibi bazı testler için DPEC oranı APEC oranında daha büyüktür. Bu tip testler için bunun anlamı, DP sırasında oluşan ve kirli satırlar üzerinden L2 önbelleğine yayılan hataların, AP periyodunda oluşan ve bir satırdan diğerine yayılan hatalara göre güvenilirlik üzerinde daha ciddi sonuçlara neden olmasıdır. Örneğin; lucas'ta APEC ve DPEC değerleri sırasıyla %8,9 ve %90,1'dir.

Bu veri açıkça gösteriyor ki DPEC'in azaltılması pratikte çok önemlidir. Grafikten gördüğümüz üzere tüm testlerdeki APEC ve DPEC'in ortalama değerleri %42,1 ve %57,9'dur.

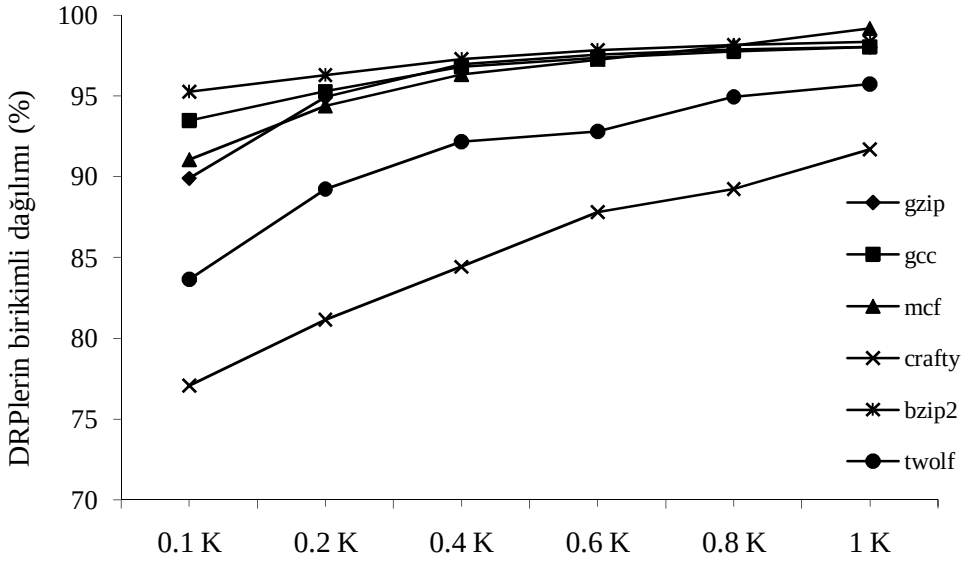


Şekil 4. AVFC'de DPEC ve APEC'in dağılımı.

Yüksek DPEC değerlerinin nedeni, aktif blokların DRP'lerinin uzunluğunun birikimli dağılımı ve kirli blokların DP'lerinin uzunluğunun birikimli dağılımına çalışılarak açıklanabilir. DRP değerlerinin birikimli dağılışı Şekil 5 (a) ve (b)'de, kirli blokların DP'lerinin birikimli dağılışı Şekil 6(a) ve (b)'de verilmiştir. Her iki şekli karşılaştırdığımızda, kirli bloklar, L2 önbelleğine yazılmadan önce (DRP'yle karşılaştırıldığında) önemli bir miktarda zaman harcarlar. Örneğin, applu benchmarkında DRP'lerin yaklaşık %93'ü, 0,2K devirdedir. Diğer taraftan, aynı uygulama için değişmiş blokların sadece %15,5'i L2 önbelleğine gönderilmeden önce 10K devir veya daha az zaman geçirirler. Başka bir örnek olarak, mcf'de, L2 önbelleğine geri yazılmadan önce 50K devir veya daha az zaman harcayan kirli satırların yüzdesi sadece %4,3'tür.

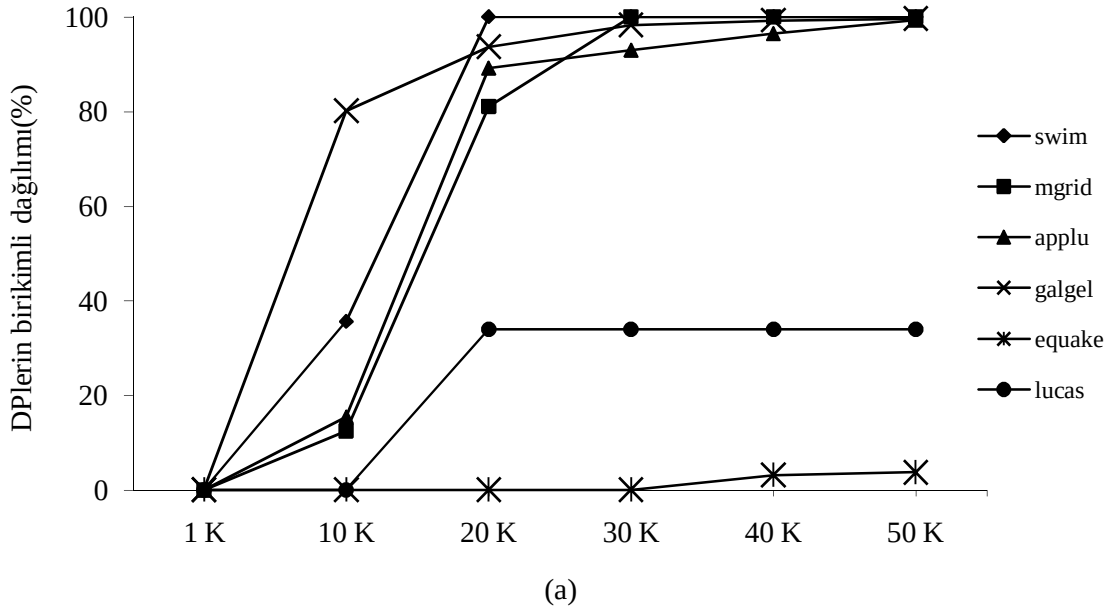


(a)

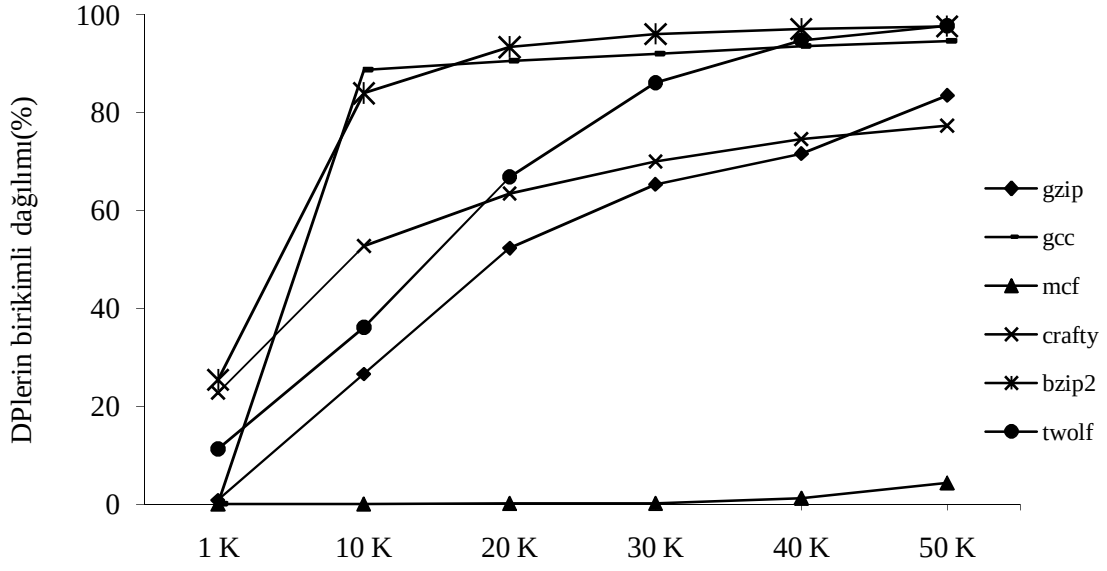


(b)

Şekil 5. Önbellek bloklarının hareketsiz periyot uzunluklarının birikimli dağılımları.



(a)

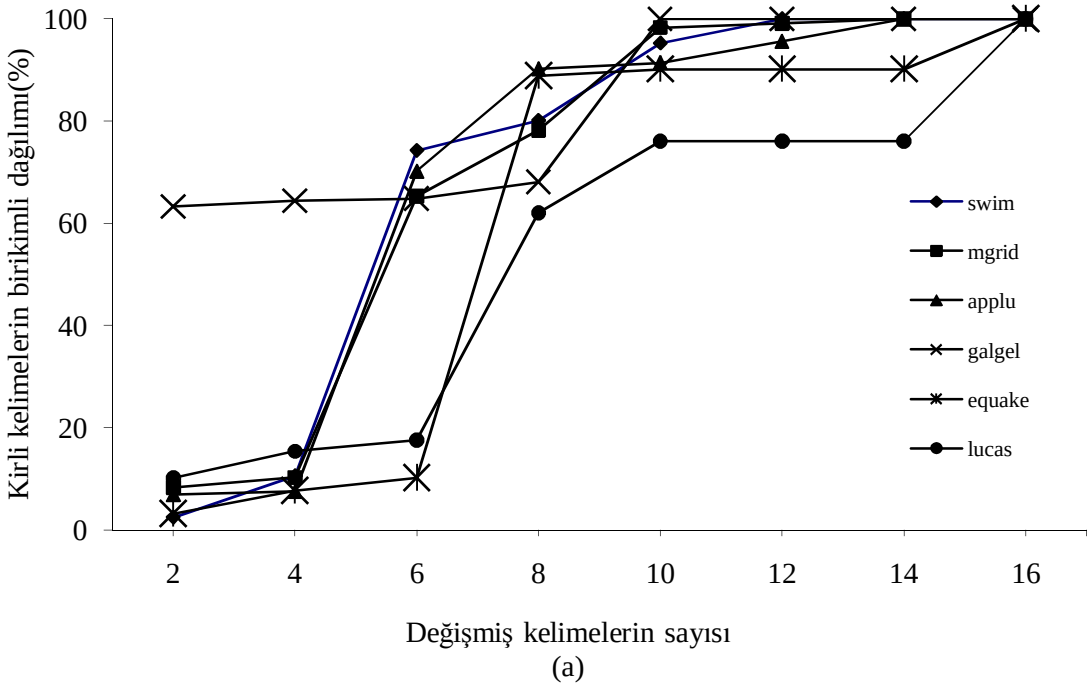


(b)

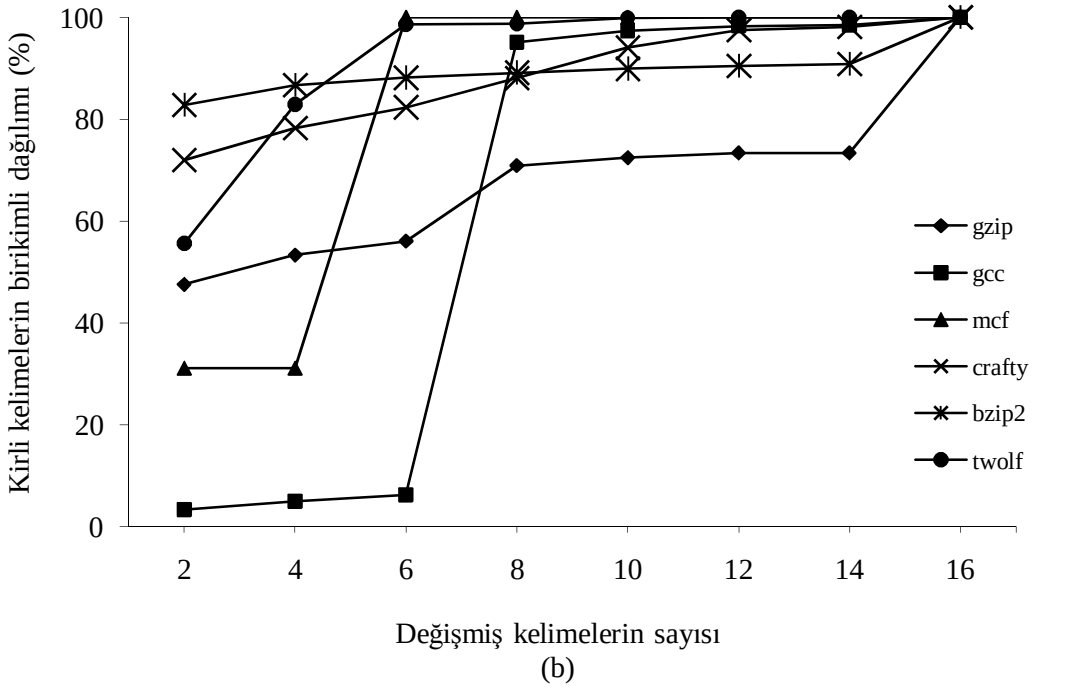
Şekil 6. Kirli önbellek blokların ölü periyotlarının (DP) uzunluğunun birikimli dağılımı.

3.3.1. Sadece Değişmiş Kelimeleri Geri Yazmanın Etkisi

Daha önce de değinildiği gibi bazı testler için DPEC değerleri oldukça yüksektir. Değiştirilmiş önbellek blokları son erişimlerinden sabit bir zaman aralığı sonra L2 önbelleğe gönderen erken geriye yazma politikası Degalahal ve diğ. (2005) tarafından çalışılmıştı. Bizim ilk yöntemimizden farklı olarak bu erken geriye yazma politikası değiştirilmiş bir önbellek bloğuna ait bir kelimedeki hatanın L2 önbelleğe yayılmasına izin verir. Ancak bizim ilk yaklaşımımız, değişmiş önbellek satırlarının değişmemiş kelimelerindeki oluşan hataların önbellek hiyerarşisinde daha alt seviyelere yayılmasını önleyerek DPEC değerlerini azaltmayı hedefler. Eğer değişmiş kelimede hata oluşup oluşmadığına bakılmaksızın bir sonraki seviyedeki önbelleğe tüm satır geri yazılırsa, kirli önbellek satırındaki hatalar, yer değişim operasyonu zamanında daha alt seviyedeki önbelleğe yayılabilir. Kirli önbellek satırının değişmemiş kelimelerinde oluşan hataların sonraki seviyeye yayılmasını, sadece değişmiş kelimeleri geri yazarak önleyebiliriz. Değişmemiş kelimelerin kopyaları bir sonraki seviyede bulunduğu için bunun verinin kullanabilirliği açısından herhangi bir sıkıntı yaratmadığına dikkat edelim. Bu yaklaşımı gerçeklemek için tüm satıra ait ‘dirty bit’ yerine, önbellek satırının her bir kelimesi için kelimenin kirli ya da kirli değil olduğunu gösteren ‘dirty bit’ kullanıyoruz. Yer değiştirme zamanında, geri-yazma kararı kelime bazında alınır ve bunun için kelime başına ekstra bir bit kullanmaktayız. Bir sonraki seviyede kelimelerin kirlilik bitine bakılarak sadece kirli kelimeler güncellenir. Bu ekstra bitler için hata kontrolünde eşlik mekanizması kullanılabilir.

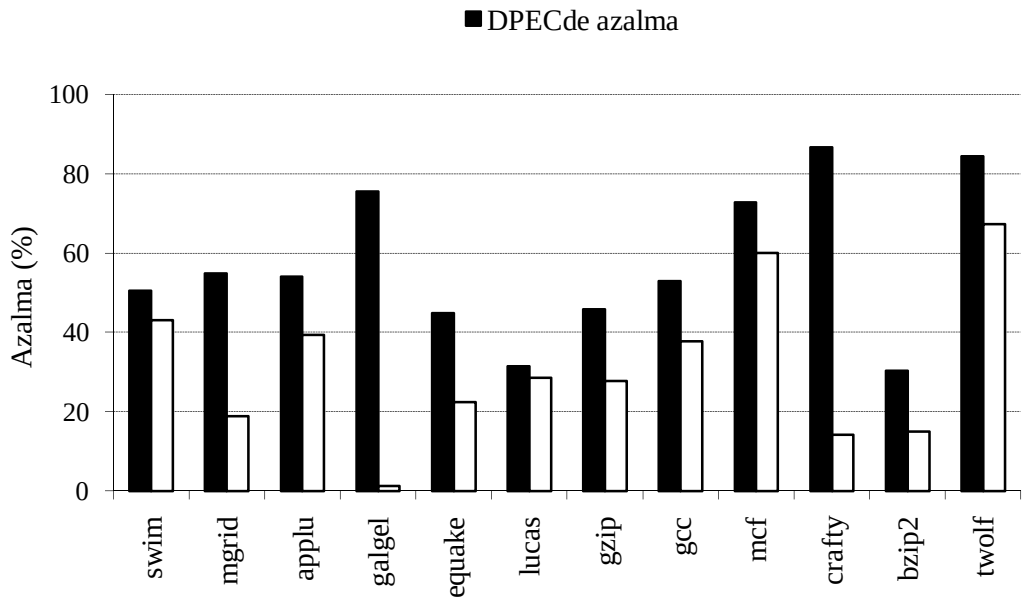


(a)



(b)

Şekil 7. 64 byte blok boylu bir veri önbelleği için kirli bloklardaki değiştirilmiş kelimelerin sayısının birikimli dağılımı. x ekseni değiştirilmiş kelimelerin sayısını temsil etmektedir.



Şekil 8. Birinci yaklaşım tarafından sağlanan DPEC ve DPEC+APEC deki azalma. Yer değiştirme işlemi esnasında L2 önbelleğine temiz (değiştirilmemiş) kelimeleri yazmamak suretiyle veri güvenirliği geliştirilir.

Bu tekniğin temel noktası, çoğu durumda değişmiş önbellek satırındaki kelimelerin yarıdan fazlasının temiz veri tuttuğudur. Bu yüzden bu tekniği kullanarak veri güvenirliği artırımı sonuçlarını göstermeden önce Şekil 7(a) ve (b) de geri-yazma işlemi sırasında değişmiş bloklardaki kirli kelimelerin sayısının birikimli dağılımını gösterelim. Sonuçlar 64-byte blok büyüklüğünde L1 veri önbelleği içindir. Deneylerimizde bir kelimenin büyüklüğünü 4 byte kabul ettiğimizden dolayı bir önbellek satırı 16 kelimeden oluşmaktadır. Şekilden görüldüğü gibi, bazı uygulamalar için geri-yazma zamanında değişmiş kelimelerin sayısı oldukça az olabilmektedir. Örneğin gzip'te L2 önbelleğine geri yazılan blokların %47,6'sı bir ya da iki tane değişmiş kelime içermektedir. İlginç olarak yarım milyar büyüklüklü bir komut penceresi için, test edilen herhangi bir uygulama için tüm kelimeleri kirli olan bir önbellek bloğuna şahit olmadık. Tüm değişmiş önbellek bloğu yerine sadece değişmiş kelimeyi geri yazmak performans, güç tüketimi ve güvenilirlikte bazı etkiler bırakabilir. Özellikle, L1 veri ve L2 önbellekleri arasındaki trafik azaldığı için bu yaklaşımla performans artırımı sağlamak mümkün olabilir. Şekil 8'de bu teknik tarafından sağlanan veri güvenilirliği gelişimi gösterilmektedir. Bu grafikte her test için iki tane sütun vardır. İlk sütun DPEC'deki gelişimi, ikinci sütun ise

DPEC+APEC'deki gelişimi göstermektedir. Tüm testlerdeki ortalama azalma DPEC için %57, DPEC+APEC için %32'dir. Güvenilirlikteki gelişimin büyüklüğünü belirleyen iki etken vardır. Birincisi, kirli satırların DP'lerinin uzunluğu, ikincisi ise değişmiş bloklardaki değişmiş kelimelerin sayısıdır. DP'nin uzunluğunun daha büyük ve değiştirilmiş kelimelerin sayısının daha az olması elde edilen güvenilirlikteki gelişmenin daha büyük olması demektir.

3.3.2. İnaktif (Aktif Olmayan) Satırların Geçersiz Kılınmasının Etkisi

Hata üretim modelimizden de görülebileceği gibi bir veride hata olma olasılığı, verinin hataya maruz kaldığı zaman aralığıyla doğru orantılıdır. Bu nedenle, güvenilirliği arttırmanın bir yolu verinin hataya maruz kalma süresinin azaltılmasıdır. Bunu başarmak için uygun inaktif önbellek satırlarını geçersiz kılarız; yani eğer son dokunulmalarından bu yana yeterince uzun zaman geçmişse geçerlilik bitlerini kapatırız. Burada belirttiği gibi inaktif blok ve ölü blok kavramlarının farklı olduğuna dikkat edelim. İnaktif blok, önbellek bloğunun belirli bir süre erişilmediği anlamına gelir, ancak bu tip bloklar tekrar erişilebilir. Ölü blokta ise bloğun son erişiminden bu yana uzun zaman geçmiştir ve büyük bir ihtimalle tekrar kullanılmaması anlamına gelir.

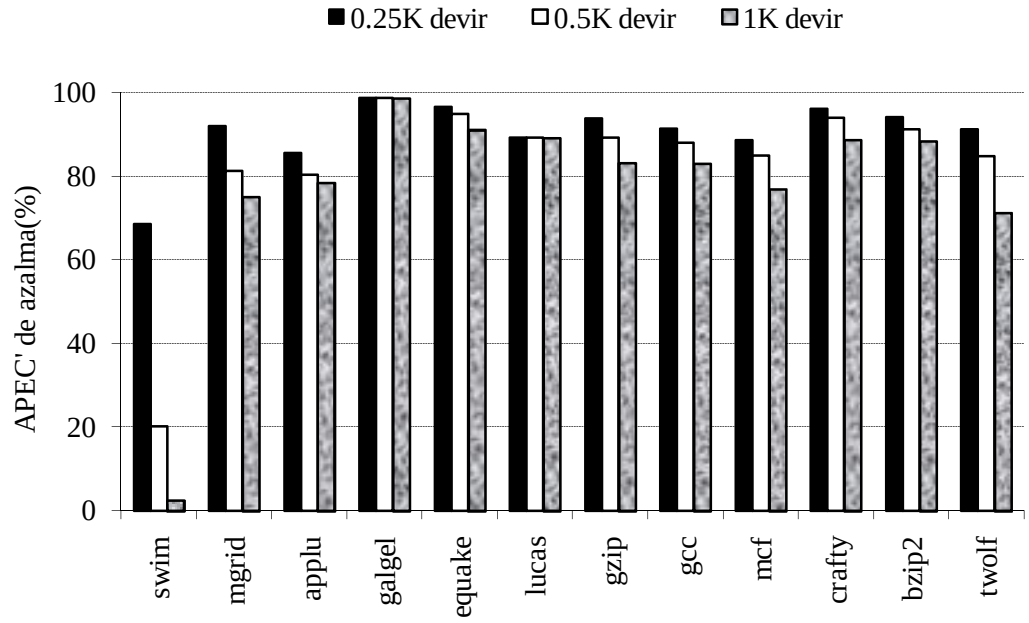
Kaxiras ve diğ. (2001) ölü blok kavramını önbellekler için bir durum yok edici sızıntı enerji azaltımı tekniğinde kullandılar ve bu tekniği önbellek çürümesi (cache decay) olarak adlandırdılar. Bu amaçla her önbellek bloğu 4 durumlu bir sonlu durum makinesi (finite state machine - FSM) ile ilişkilendirildi. Önbellek bloğuna erişilmediği sürece FSM durumlar arası ilerler. Son duruma gelindiğinde önbellek bloğu kapatılır. İkinci tekniğimizde hem kirli ve hem de temiz bloklar için inaktif önbellek bloğunun ne zaman geçersiz kılınması gerektiğine karar vermede aynı mekanizmayı kullanmaktayız.

İkinci tekniğimizde, geçersiz kılınmış bir bloğa istekte bulunulduğunda, onun hatasız taze bir kopyası bir sonraki seviyedeki önbellekten getirilir, böylece veri güvenilirliği arttırılır. Ancak bu yaklaşım bazı performans sorunlarını da beraberinde getirir. Özellikle bu yöntem ıskaları ve bus trafiğini arttırdığından bazı performans

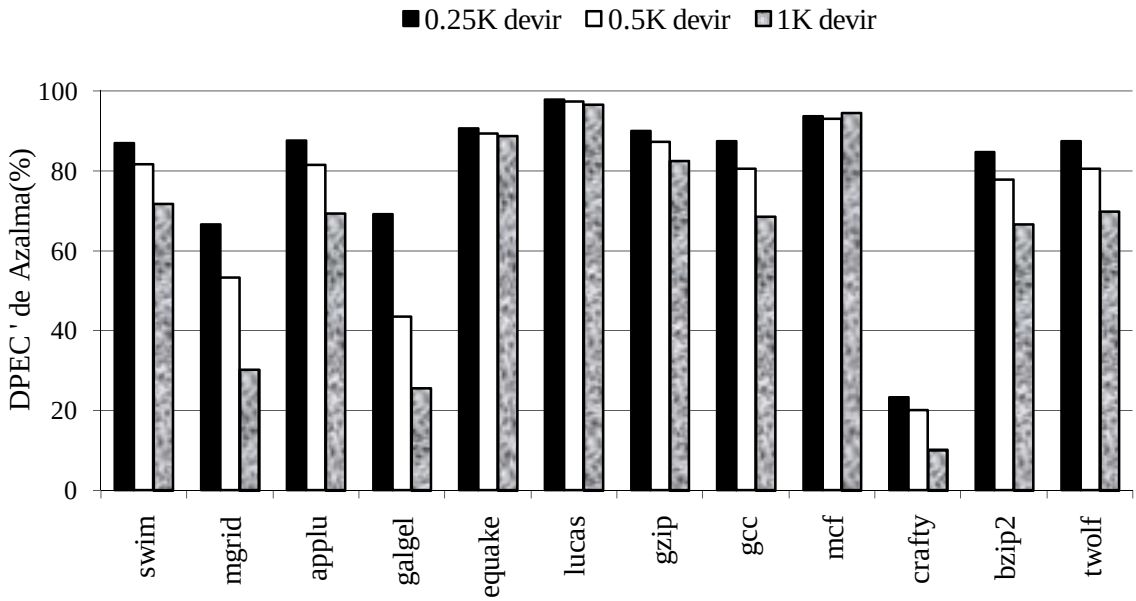
sorunlarına yol açar. Performans kaybını azaltmanın mümkün olup olamayacağını görmek için satırlar LRU durumuna ulaştıklarında onları geçersiz kılan ve Lee ve diğ. (2000) tarafından önerilen kirli blokların geçersiz kılınması için istekli geriye yazma mekanizmasını ileride kullanmayı planlamaktayız. Kabul edilebilir performans kaybıyla veri güvenilirliğini arttırmak için aynı zamanda uygun bir eşik değerine karar vermek oldukça önemlidir. Buna DRP (dormant period) denir. Eğer DRP çok kısa seçilirse, önbellek ıskalarının ve geri yazmaların sayısı oldukça artar, bu da düşük performansa yol açar. Diğer taraftan eğer DRP gereksiz bir biçimde uzun seçilirse, blok büyük olasılıkla ölü periyoda geçer, yani bunun anlamı güvenilirliği arttırma şansını kaçırdığımızdır. Çünkü temiz ölü bloklar güvenilirliği etkilemez.

Bu tekniğin diğer bir iyi özelliği de, sadece çalışma zamanında CPU'ya hatasız veriyi iletmeye yardımcı olmayıp, aynı zamanda blokların L2 önbelleğine kirli bloklar vasıtasıyla hata yayılımını azaltmaya yardımcı olmasıdır. Bu da kirli blokları geriye erken yazmaya zorlayarak, yani onların hataya maruz kalma sürelerini azaltarak yapılır. Şekil 9, 10, 11 değişik DRP uzunlukları için bu yöntemin veri güvenirliği artırımını gösterir. Şekil 9 ve 10 sırasıyla APEC ve DPEC'deki azalımı, Şekil 11 ise APEC+DPEC'deki azalımı gösterir. Bu sonuçları elde etmek için DRP uzunluğunu 0,25K, 0,5K ve 1K olarak değiştirdik. Bu üç şekilden ortak bir gözlem bizim yöntemimizin sağladığı veri güvenilirliğindeki artış yüzdesinin DRP azaldıkça artış yönünde olmasıdır. Örneğin; applu testi için APEC+DPEC deki azalım, 0,25K, 0,5K ve 1K devir için sırasıyla %87,5, %81,2, %71,83'tür. Ayrıca, Şekil 5'e bakarak, uzun DRP'li blokları göz önünde bulundurduğumuzda bu blokların önemli bir yüzdesine ilgili DRP süresince tekrar erişileceğinden, bu tip blokların gereksiz kılınabilme şansına sahip olmayacaklarını düşünebiliriz ve böylece ikinci yöntemin güvenilirliği gerçekleştirmede zayıf kalacağını düşünebiliriz. Tam aksine iki ardışık erişimleri arasındaki sürenin DRP'den büyük olduğu blokların yüzdesinin oldukça küçük olmasına rağmen, bu süre DRP ile karşılaştırıldığında oldukça uzundur ve bu da bu tekniği APEC'in azalmasında etkili hale getirir. swim'de Şekil 5(a)'da gösterildiği gibi önbellek bloklarının %100'ü 1K devir sürede erişildiği için 1K devirlik DRP için APEC azalımı çok küçüktür (%2,5). Bu yaklaşımla ilgili başka bir

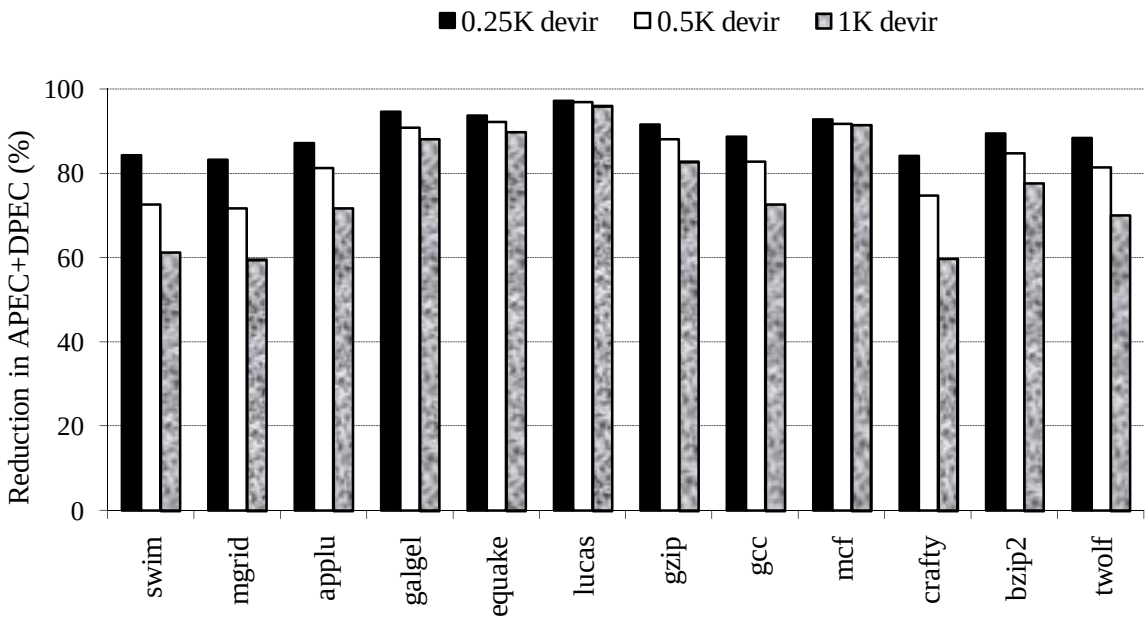
gözlem, deneyde kullandığımız tüm testler için güvenilirliği oldukça arttırmış olmasıdır. Şekil 11'den görebileceğimiz gibi APEC+DPEC'deki ortalama azalma 0,25K, 0,5K ve 1K DRP'ler için sırasıyla %90, %87 ve %77'dir.



Şekil 9. İkinci yaklaşım tarafından 0,25 K, 0,5 K ve 1 K devirlik DRP'lerle sağlanan APEC'deki azalma.

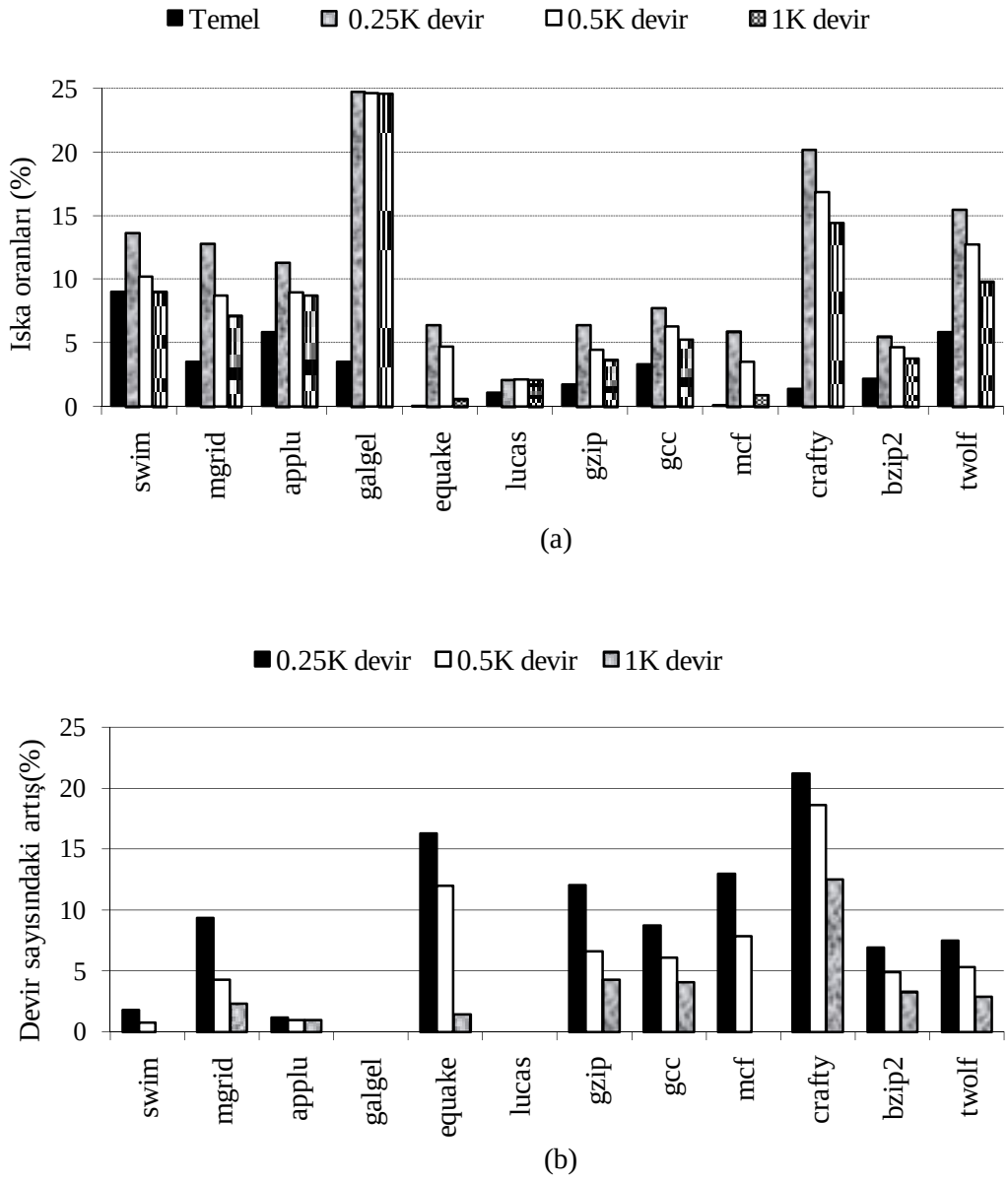


Şekil 10. İkinci yaklaşım tarafından 0,25K, 0,5K ve 1K devirlik DRP için DPEC'deki azalma.



Şekil 11. İkinci yaklaşım tarafından 0,25 K, 0,5 K ve 1 K devirlik DRP'lerde sağlanan APEC+DPEC'deki azalma.

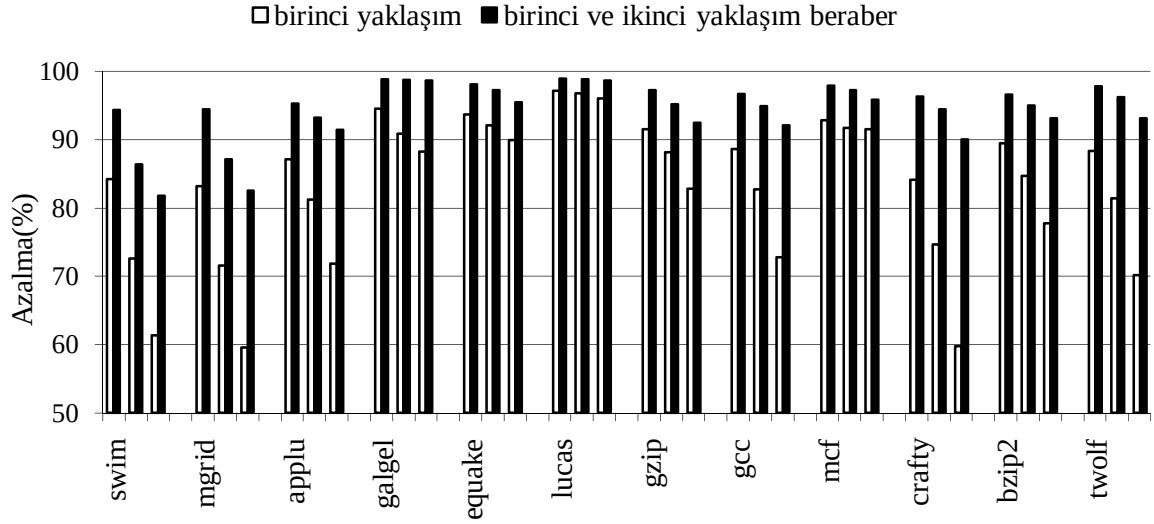
Bu teknik, ıska oranını ve geri yazma sayısını arttırarak performansa olumsuz yönde etki eder. Bu nedenle, Şekil 10'da ima edildiği gibi geçersiz kılma için çok kısa zaman aralıkları kullanamayız. Şekil 12, blokları geçersiz kılmak için değişik DRP'ler kullanıldığında ıska oranı ve devir sayısındaki artışı gösterir. Bu şekillerden gözlenebileceği üzere, genelde DRP uzunluğu azaldığında hem ıska oranı hem de devir sayısı artar. Bu gözleme, 'galgel' ve 'lucas' 2 tane istisnadır. galgel için önbellek satırının geçersiz kılınmasının sonucu olarak ıska oranında dramatik bir artış olmasına rağmen, performans azalımı ihmal edilebilecek kadar küçüktür. Bu durum şöyle izah edilebilir. Bu uygulama oldukça fazla veri bağımlılığına sahiptir. Dolayısıyla Tablo 2'deki ilgili devir sayılarının da ima ettiği gibi oluşan ıskalar performansı kötüleştirmeksizin arka planda tolere edilebilir. Diğer taraftan, lucas için ıska oranında azalma çok küçük olduğundan inaktif satırların geçersiz kılınması herhangi bir hissedilebilir performans kaybına neden olmaz. Şekil 12 (b)'den görebileceğimiz gibi bu teknik 0,25K, 0,5K ve 1K'lık DRP için temel duruma oranla sırasıyla ortalama %8,2, %6,2 ve %5,6'lik performans kaybına neden olur.



Şekil 12. İkinci yaklaşımımız için performans değerleri. Sonuçlar 0,25K, 0,5 K ve 1 K'lık devirler içindir. Sonuçlar temel durumdaki ilgili değerlere göre artış oranı şeklinde verilmiştir.

İlk tekniğimizin herhangi bir performans yüküne neden olmadığını hatırlayalım. Dolayısıyla, fazladan performans kaybına sebep olmaksızın, birinci teknik ikincisiyle birlikte kullanılarak ikinci tekniğin tek başına sağladığı güvenilirlik artışından daha fazla güvenilirlik artışı elde edilebilir. Bu seçeneği değerlendirmek için sonuçları Şekil 13'de gösterilen diğer bir takım deneyler yaptık. Bu grafikte üç

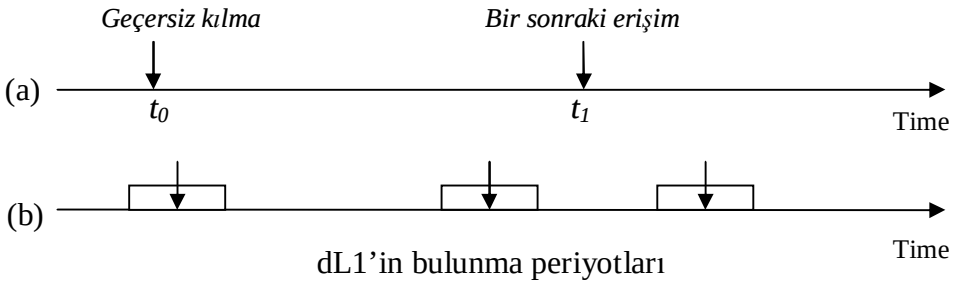
grup sütun var. Soldan sağa doğru, birinci, ikinci ve üçüncü gruplar 0,25K, 0,5K ve 1K devirlik DRP'ler için APEC+DPEC'deki azalmayı göstermektedir. Her gruptaki ilk sütun ikinci yaklaşımımız tarafından tek başına sağlanan gelişmeyi gösterirken, ikinci sütun birinci yaklaşımın ikinci yaklaşımla kullanılması sonucu elde edilen gelişmeleri göstermektedir. Daha önce de söylediğimiz gibi, tüm testler düşünüldüğünde ilk yaklaşımımız ile sağladığımız güvenilirlik artışı (APEC+DPEC) oldukça fazladır, ortalama %32 (Şekil 8). Ancak, ilk yöntemimiz ikinci yöntemle kullanıldığı zaman tek başına sağladığı katkıdan daha az bir katkı sağlar. Bu aşağıdaki, sebepten dolayıdır. Kirli önbellek bloklarının L2 önbelleğine oldukça erken yazılmasıyla ikinci yöntem tarafından önbellek bloklarının geçersiz kılınması değiştirilmiş önbellek bloklarının hatalara maruz kalma periyodunu azaltır. Bunun iki temel sonucu vardır. İlki Şekil 10'da gösterildiği gibi bunun DPEC'de azalmaya sebep olmasıdır ve ikincisi ise bunun ilk yöntemimizin etkinliğini azaltmasıdır. Diğer bir deyişle, ilk yöntemimizin yalnız başına sağladığı güvenilirlik gelişiminin bir kısmını ikinci yöntem de sağlayabilir, bu da DPEC'in gelişimi düşünüldüğünde ilk yönteme daha az fırsat bırakılması anlamına gelir. Buna rağmen özellikle uzun DRP'li bazı uygulamalar için ilk yöntem makul bir miktarda ekstra güvenilirlik artışı sağlar.



Şekil 13. İkinci yaklaşım yalnız başına ve birinci ve ikinci yaklaşım beraber uygulandığında APEC+DPEC'deki azalma. DRP değerleri 0,25K, 0,5K ve 1K devir içindir.

3.3.3. İnaktif Satırın Yenilenmesinin Etkisi

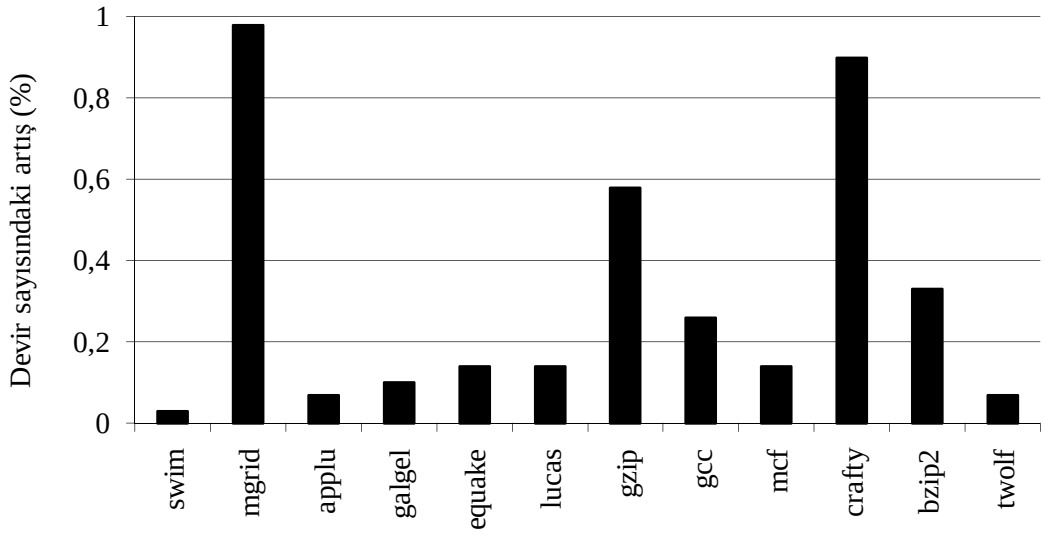
Daha önce söylediğimiz gibi ikinci yaklaşımla ilgili önemli bir problem, getirdiği performans kaybıdır (ıska oranı için Şekil 12(a) ve karşılık gelen devir sayısı için Şekil 12(b)). Gerçekte geçersiz kılmaya bağlı performans kaybı Şekil 12 (a)'da 0,25K devirlik DRP için yaklaşık %8'dir ki bu da oldukça yüksektir. Buna ek olarak, crafty uygulamasında blokları geçersiz kılma performansa %21'e kadar zarar verebilir ki bu da çoğu çalışma ortamında tolere edilemez. Üçüncü yöntemimiz bu performans kaybını azaltmaya çalışır. Bunu başarmak için önbellek satırı geçersiz kılındıktan hemen sonra ileriki erişimler için veriyi önbellekte hazır kılmak için ilgili satırı önceden getirme yöntemi ile önbelleğe getirir. Bu yöntemde sadece temiz satırlar geçersiz kılındığında, önceden getirme sadece temiz veri için uygulanır.



Şekil 14. (a) Önbelleğin geçersiz kılınması ve bloğa bir sonraki erişim. (b) Performans için önemli zaman aralıkları.

Bu tekniğin arkasındaki mantığı açıklamak için Şekil 14 (a)'yı göz önüne alalım. Bu şekilde yatay eksen zamanı gösterir ve bizim yaklaşımımıza göre önbellek satırı t_0 zamanında geçersiz kılınır. Satıra bir sonraki erişimin t_1 zamanında yapıldığını düşünelim. t_0 ve t_1 zamanları arasında satır dL1 önbelleğinde tutulmayarak herhangi bir performans kaybı yaşanmaz, çünkü problemdeki satıra bu arada gerek duyulmayacaktır. Bu yüzden, bizim üçüncü yöntemimiz gerek duyulan satıra sonraki erişim gerçekleşmeden önce onu bellekte hazır bulma umuduyla o satır geçersiz kılındıktan sonra önceden getirme isteğinde bulunur. İdeal durumda, dL1 deki her satır için Şekil 14 (b)'de gösterildiği gibi bir şablon izlenmelidir. Bu şablondan da görüldüğü gibi satırın önbellek içinde sadece erişildiği zaman aralığı bulunmasını

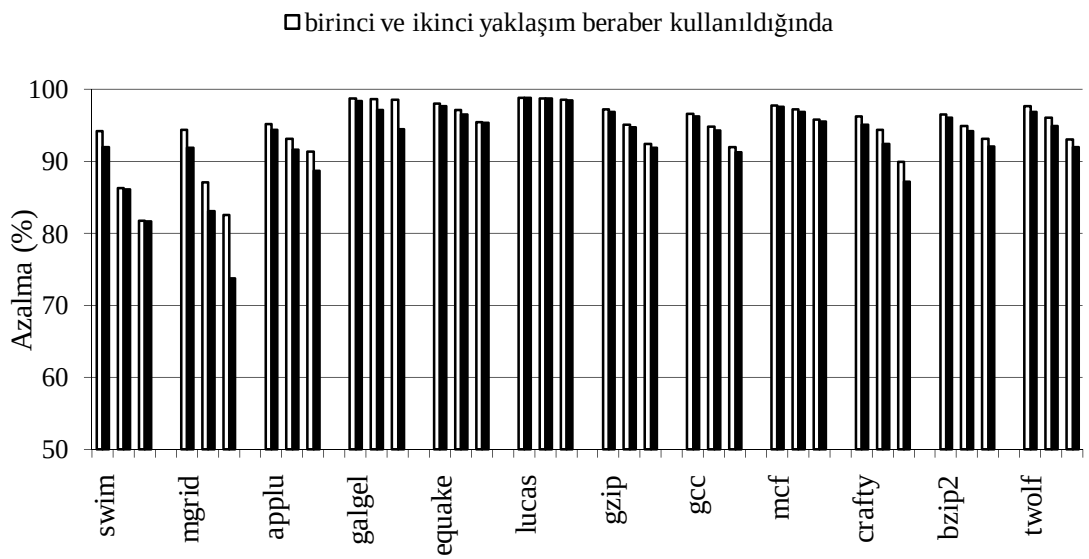
istiyoruz (erişimlerin yapıldığı zaman aralığında küçük bloklar şeklinde gösterilmiştir.) İşletimin geri kalanında, satırın geçici hatalara karşı daha iyi korunabileceği L2'de bulunması tercih edilir (çoğu mimaride L1 eşlenik ile korunurken, L2 ECC ile korunur).



Şekil 15. Temel duruma oranla birinci ve üçüncü yaklaşım 0,25K devirlik DRP için beraber uygulandığında devir sayısında oluşan artış.

Bu yaklaşımı kullanarak elde ettiğimiz sonuçlara bakalım. Şekil 15, 0,25 devirlik DRP için birinci ve üçüncü yaklaşımları bir arada kullandığımızda temel duruma oranla devir artışını göstermektedir. Şekilden açıkça görüldüğü gibi, satırın geçersiz kılınmasıyla oluşan performans kayıplarını azaltmak için önceden getirme oldukça verimli bir yöntemdir. Özellikle, üçüncü yaklaşımımız performans kaybını, tüm test edilen uygulamalar için %1'den aşağı çekmiştir.

Şekil 16, farklı tekniklerin bir arada kullanılmasıyla APEC+DPEC'deki azalmayı göstermektedir. Bu grafikte, her bir test için birinci ve ikinci grup sütunlar 0,25K ve 0,5K devirlik DRP'ye karşılık gelmektedir, hâlbuki son grup sütun 1K devirlik DRP içindir. Her grupta ilk sütun birinci ve ikinci yaklaşımın bir arada kullanılmasında, ikinci sütun birinci ve üçüncü tekniğin beraber kullanılmasıyla APEC+DPEC'de oluşan azalmayı göstermektedir. Şekilden de görüldüğü gibi testlerin çoğu için veri bütünlüğü üzerindeki önceden getirmenin etkisi oldukça azdır. mgrid testi 1K devirlik DRP için yaklaşık %9 ile en fazla güvenilirlik kaybı gösteren testtir.



Şekil 16. 0,25K, 0,5K ve 1K devir DRP'ler için birinci ve ikinci yaklaşım beraber kullanıldığında ve birinci ve üçüncü yaklaşım beraber kullanıldığında oluşan APEC+DPEC'deki azalma.

Tablo 4'te tüm testler için AVFC değerleri ve ortalamaları gösterilmiştir. Tablo 3 ve Tablo 4'teki değerler karşılaştırıldığında bu uygulamaların AVFC değerlerini azaltmada tekniklerimiz oldukça etkilidir. Daha açık söylemek gerekirse, Tablo 3'te temel durum için ortalama AVFC değeri 0.17 iken, birinci ve ikinci yaklaşım bir arada kullanıldığında ve birinci ve üçüncü yaklaşım beraber kullanıldığında ortalama AVFC sırasıyla 0,004 ve 0,006'dır. Bu değerler güvenilirliği arttırmaya yönelik yöntemlerimizin nedenli etkili olduğunun kanıtıdır.

Tablo 4. Yaklaşımlarımız uygulandığında ortaya çıkan AVFC değerleri

Test	İlk Yaklaşım	İkinci Yaklaşım	Birinci ve İkinci Yaklaşım Beraber	Birinci ve Üçüncü Yaklaşım Beraber
swim	0.060	0.016	0.006	0.006
mgrid	0.114	0.023	0.007	0.011
applu	0.066	0.014	0.005	0.006
galgel	0.189	0.012	0.003	0.004
equake	0.203	0.014	0.004	0.006
lucas	0.172	0.007	0.003	0.003
gzip	0.129	0.013	0.004	0.005
gcc	0.078	0.013	0.004	0.005
mcf	0.096	0.015	0.005	0.006
crafty	0.146	0.022	0.005	0.008
bzip2	0.101	0.012	0.004	0.005
twolf	0.037	0.012	0.002	0.003
Ort.	0.116	0.014	0.004	0.006

BÖLÜM 4

SONUÇ VE TARTIŞMA

Bu çalışmanın temel sonucu önbelleklerin güvenilirliği için AVFC olarak adlandırılan yeni bir ölçü ve L1 önbelleğinin güvenilirliğini arttırmak için üç mimarisel yaklaşım önermesidir. Yaklaşımlarımız önbellek güvenilirliğini arttırmak için eşlik ve ECC gibi geleneksel yöntemlerle birlikte kullanılabilirler. Üç yaklaşımımızı simülasyon ortamında değerlendirdik ve temel bulgular da aşağıdaki şekilde özetlenebilir.

L1 veri önbelleğinin kirli blokları, L2 önbelleğine geri yazılmadan önce iki ardışık blok erişimleri arasındaki ortalama zamana kıyasla oldukça uzun bir zaman harcarlar. Bunun yanında çoğu durumda değişmiş önbellek bloklarının kelimelerinin yarıdan fazlası önbellekten atıldıkları sırada temiz veri tutmaktadırlar. Gerçekte DP'ler sırasında hataların oluşması ve kirli bloklar vasıtasıyla L2 önbelleğine yayılması tüm hataların %57,9 civarlarındadır.

İlk yaklaşımımız bir sonraki seviyede önbellekte hata yayılımını önlemek için DPEC'in azaltılmasını hedefler. Bunu sağlamak için de kirli blok önbellekten atıldığında kirli önbellek bloğunun değişmemiş veri kelimeleri L2 önbelleğine yazılmaz. Bu yaklaşımla DPEC'de sağlanan ortalama azalma %57'dir.

İkinci yaklaşımımız, L2 önbelleğine değişmiş verileri erkenden geri yazmaya zorlayarak APEC'in yanında DPEC'de de azalma sağlamaktadır. Bu yaklaşım sonucunda APEC+DPEC'deki ortalama azalma 0,25K, 0,5K ve 1K devir DRP'lerde sırasıyla %90, %84 ve %77'dir. Ancak bu yaklaşım 0,25K uzunluğundaki DRP'lerde %8 oranında performans kaybına neden olur.

Blokların geçersiz kılınmasıyla oluşan performans yükünü hafifletmek için üçüncü yaklaşımımız önceden getirme ile geçersiz kılınmış bloğun yeni bir kopyasını getirir. Önceden getirme geçersiz kılmanın neden olduğu performans kayıplarını azaltmakta oldukça başarılıdır.

İlk yaklaşımımız herhangi bir performans yüküne neden olmadığından güvenilirliği daha da arttırmak için ikinci ve üçüncü yaklaşımlarımızla birlikte kullanılabilir. Birinci ve ikinci yaklaşımlarımız ve birinci ve üçüncü yaklaşımlarımız beraber kullanıldığında 0,25K devir DRP'lerde APEC+DPEC değerlerinde sırasıyla %97 ve %96 ortalama azalma sağlamıştır.

Birinci ve ikinci yaklaşımlarımız beraberce orijinal ortalama AVFC değerini 0.17'den 0,004'e düşürmüştür ve birinci ve üçüncü yaklaşımımız beraber kullanıldığında AVFC'nin ortalama değeri 0,006 olmuştur.

İleriki çalışmalarımızda AVFC metriğini L2/L3 önbelleklerde kullanmayı planlamaktayız. Ayrıca bu çalışmamızı çok çekirdekli (Multicore on Chip) bilgisayar sistemleri için tekrarlamayı planlamaktayız. Bunun için önce hata oluşumu ve yayılımı modelimizi çok çekirdekli sistemler için yeniden ifade edeceğiz. Sonra bu yeni model kullanılarak hataların bir çekirdeğe ait veri önbelleğinden diğer başka bir çekirdeğe ait önbelleğe asgari oranda taşınmasına yönelik çeşitli yöntemlerin veri güvenirliği üzerindeki performanslarını araştıracağız.

KAYNAKLAR

Cannon E. H., Reinhardt D. D. ve Makowenskyj P. S., 2004. SRAM SER in 90, 130and 180nm Bulk and SOI Technologies. *Int. Rel. Phys. Symp.*

Carmichael C., 2001. Triple Modular Redundancy Design Tecniques for Virtex FPGAs. *Xilinx Application Notes 197,v1.0.*

Chen C. L. ve Hsiao M. Y., 1992. *Reliable Computer Systems- Design and Evaluation, Digital Press.* (2nd ed.). Error-correcting Codes for Semiconductor Memory Applications: a State of the Art Review771-786

Degalahal V., Li L., Narayanan V.,Kandemir M. ve Irwin M. J., 2005. Soft Error Issues in Low-Power Caches. *IEEE Trans. On Very Large Scale Integ. Sys.*

Gomaa M., Scarbrough C., Vijaykumar T. N. ve Pomeranz I. 2003. Transient-Fault Recovery for Chip Multiprocessors. *Int. Symp. on Comp. Arch.*

Gomaa M. A., ve Vijaykumar T. N., 2005. Opportunistic Transient Fault Detection. *Int. Symp. On Comp. Arch.*

Hareland S., Maiz J., Alavi M., Mirtry K., Walstra S. Ve Dai C. 2001. Impact of CMOS Scaling and SOI on Soft Error Rates of Logic Processes. *VLSI Technology Digest of Technical Papers.*

Irom F., Farmamesh F. F., Johnson A. H., Swift G. M. ve Millward D. G., 2002. Single-Event Upset in Commercial Silicon-on-Insulator PowerPC Microprocessors. *IEEE Trans. on Nucl. Sci. 49(6).*

Karnik T., Bloechel B., Soumyanath K., De V. ve Borkar S., 2001. Scaling Trends of Cosmic Rays Induced Soft Errors in Static Latches Beyond 0.18u. *Symp. on VLSI Cricuits Digest of Technical Papers.*

Karnik T., Hazucha P. ve Patel J., 2004. Characterization of Soft Errors Caused by Single Event Upset in CMOS Processes. *IEEE Trans. on Dep. and Sec. Comp. 1(2):128-143.*

Kaxiras S., Hu Z. ve Martonosi M. 2001. Cache Decay: Exploiting Generational Behaviour to Reduce Cache Leakage Power. *Int. Symp. On Comp. Arch.*

Kim S. ve Somani A. K., 2002. Soft Error Sensitivity Characterization for Microprocessors Dependability Enhancement Strategy. *Int. Conf. On Dep. Sys. and Net.*

Kumar S. ve Aggarwal A., 2006. Reducing Resource Redundancy for Concurrent Error Detection Tecniques in High Performance Microprocessors. *Int. Symp. On High-Per. Comp. Arch.*

Lee H. H. S., Tyson G. S. ve Farrens M. K., 2000. Eager Writeback- a Techniques for Improving Bandwidth Utilization. *Int. Symp. on Micro.*

Li X., Adve S. V., Bose P. ve Rivers J. A., 2005. SoftArch: an Architecture-Level Tool for Modeling and Analyzing Soft Errors. *Dependable Systems and Networks.*

Mukherjee S. S. , Weaver C., Emer J. ve Reinhardt S. K. ve Austin T., 2003. A Systematic Methodology to Compute the Architectural Vulnerability Factors for a High-Performance Microprocessors. *Int. Symp. on Micro.*

Mukherjee S. S., Emer J. ve Reinhardt S. K., 2005. The Soft Error Problem: an Architectural Perspective. *Int. Symp. on High-Perf. Comp.*

Nguyen H. T. ve Yagil Y., 2003. A Systematic Approach to SER Estimation and Solutions. *IEEE Int. Rel. Phys.*

Pradhan D. K., 2003. Fault Tolerant Computer System Design (2nd ed.). *Computer Science Press.*

Reinhardt S. K. ve Mukherjee S. S., 2000. Transient Fault Detection via Simultaneous Multithreading. *Int. Symp. on Comp. Arch.*

Shivakumar P., Kistler M., Keckler S. W., Burger D. ve Alvisi L., 2002. Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic. *Int. Conf. On Dep. Sys and Net.*

Shridhan V., Asadi H., Tahoori M. B., Kaeli D., 2006. Reducing Data Cache Susceptibility to Soft Errors. *IEEE Trans. on Dep. and Sec. Comp. 3(4):353-364.*

Vijaykumar T., Pomeranz I. ve Cheng K., 2002. Transient-Fault Recovery Using Simultaneous Multithreading. *Int. Symp. on Comp. Arch.*

Wang N. J. ve Patel S., 2003. Characterizing the Effects of Transient Faults on a High Performance Processor Pipeline. *Int. Conf. on Dept. Sys. and Net.*

Wang N. J., Quek J., Rafacz T. M. ve Patel S. J. 2004. Characterizing the Effects of Transient Faults on a High-Performance Processor Pipeline. *Int. Conf. on Dep. Sys. And Net.*

Weaver C., Emer J., Mukherjee S. S. ve Reinhardt S. K., 2004. Techniques to Reduce the Soft Error Rate of a High Performance Microprocessor. *Int. Symp. on Comp. Arch.*

Ziegler J. F., 1996. Terrestrial Cosmic Rays. *IBM Journal of Research and Development, 40(1):19-39.*

TABLOLAR

Sayfa

Tablo 1. Deneylerimizdeki temel konfigürasyon parametreleri ve varsayılan değerleri 13

Tablo 2. Deneylerimizde kullanılan testlerin önemli özellikleri 14

Tablo 3. Temel durum (soft errorları azaltmaya yönelik bir mekanizma kullanılmadığı durum) için kodların AVFC' leri 15

Tablo 4. Yaklaşımlarımız uygulandığında ortaya çıkan AVFC değerleri 31

ŞEKİLLER

Sayfa

Şekil 1. (Üstte)Kaynak kod parçası ve (altta) karşılık gelen birleştirici dil kodu 11

Şekil 2. Şekil 1'deki kod parçacığının veri akış diagramı (DFG) 11

Şekil 3. Şekil 1deki kod parçasının veri işleyişi için önbelleğe erişimi de gösteren zaman tablosu 11

Şekil 4. AVFC'de DPEC ve APEC'in dağılımı 16

Şekil 5. Önbellek bloklarının hareketsiz periyot uzunluklarının birikimli dağılımları 17

Şekil 6. Kirli önbellek blokların ölü periyotlarının (DP) uzunluğunun birikimli dağılımı 18

Şekil 7. 64 byte blok boylu bir veri önbelleği için kirli bloklardaki değiştirilmiş kelimelerin sayısının birikimli dağılımı. x ekseni değiştirilmiş kelimelerin sayısını temsil etmektedir 20

Şekil 8. Birinci yaklaşım tarafından sağlanan DPEC ve DPEC+APEC deki azalma. Yer değiştirme işlemi esnasında L2 önbelleğine temiz (değiştirilmemiş) kelimeleri yazmamak suretiyle veri güvenirliliği geliştirilir 21

Şekil 9. İkinci yaklaşım tarafından 0,25 K, 0,5 K ve 1 K devirlik DRP'lerle sağlanan APEC'deki azalma 24

Şekil 10. İkinci yaklaşım tarafından 0,25 K, 0,5 K ve 1 K devirlik DRP için DPEC' deki azalma 24

Şekil 11. İkinci yaklaşım tarafından 0,25 K, 0,5 K ve 1 K devirlik DRP'lerde sağlanan APEC+DPEC'deki azalma 25

Şekil 12. İkinci yaklaşımımız için performans değerleri. Sonuçlar 0,25K, 0,5 K ve 1 K' lık devirler içindir. Sonuçlar temel durumdaki ilgili değerlere göre artış oranı şeklinde verilmiştir 26

Şekil 13. İkinci yaklaşım yalnız başına ve birinci ve ikinci yaklaşım beraber uygulandığında APEC+DPEC'deki azalma. DRP değerleri 0,25K, 0,5K ve 1K devir içindir 27

Şekil 14. (a) Önbelleğin geçersiz kılınması ve bloğa bir sonraki erişim.(b) Performans için önemli zaman aralıkları 28

Şekil 15. Temel duruma oranla birinci ve üçüncü yaklaşım 0,25K devirlik DRP için beraber uygulandığında devir sayısında oluşan artış 29

Şekil 16. 0,25K, 0,5K ve 1K devir DRP'ler için birinci ve ikinci yaklaşım beraber kullanıldığında ve birinci ve üçüncü yaklaşım beraber kullanıldığında oluşan APEC+DPEC'deki azalma 30

YAŞAM ÖYKÜSÜ

Kişisel Bilgiler

Adı Soyadı :Hande ŞEN

Doğum yeri ve yılı :Çanakkale -1982

Adres : Çanakkale Onsekiz Mart Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü Terzioğlu Yerleşkesi, 17100 ÇANAKKALE

Tel :0 542 223 31 39

e-posta:hande_sen@yahoo.com

Eğitim Durumu

2000-2004 : Ege Üniveristesi-Mühendislik Fakültesi-Bilgisayar Mühendisliği Bölümü

1998-2000 : Çanakkale Özel Lisesi, ÇANAKKALE

1994-1998 : Çanakkale Milli Piyango Anadolu Lisesi, ÇANAKKALE

1993-1994 : Bandırma Anadolu Lisesi, Bandırma/BALIKESİR

1991-1994 : Erdek İstiklal İlkokulu, Erdek/BALIKESİR

1989-1991 : Erdek 18 Eylül İlkokulu, Erdek/BALIKESİR

Stajlar

Temmuz-Ağustos 2003 : Başak Sigorta A.Ş., İSTANBUL

Ocak-Temmuz 2003 : Ege Üniversitesi Dış İlişkiler Ofisi,İZMİR

Mesleki Deneyim

2005-2007 : Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü, Arş.Gör.

Çalışma ve İlgi Alanları

Bilgisayar Mimarisi

Veritabanı yönetimi

Veritabanı yazılımları