iTLB Güvenirliğinden Kaynaklanan Performans Kaybının Azaltılması
Reducing the Performance Overheads Introduced by iTLB Reliability

İsmail KADAYIF

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

Özet

Transistor boyutlarının küçülmesiyle donanım bileşenleri geçici hatalara karşı çok daha duyarlı hale gelebilmektedir. Eşlik kontrolü ve hata düzeltme kodları gibi bütünlük kontrol yöntemleri geçici hatalara karşı devreleri korumada kullanılan en yaygın yöntemler arasındadır. Ancak bu tip yöntemler, Komut Adresi Dönüştürme Tamponu (Instruction Translation Lookaside Buffer –iTLB) gibi çok sıklıkla erişilen donanım bileşenlerinde gecikmelere ve dolayısıyla programın performansının azalmasına sebep olabilmektedir. Bu çalışmada iTLB'lerin geçici hatalara karşı korunmasından kaynaklanan performans kaybının azaltılmasına yönelik bir yöntem sunmaktayız. Bunun için komutlara yönelik son sanal-fiziksel sayfa adres dönüşümünü tutan özel bir yazmaç kullanmaktayız. Aynı sayfada kalındığı sürece adres dönüşümü geçici hatalara karşı korumalı bu özel yazmaçtan temin edilerek iTLB'ye erişim sayısı minimize edilip, hata kontrolü veya düzeltilmesi için gerekli gecikmelerden sakınılır. SPEC2006 uygulamaları üzerinde SimpleScalar simülatörüyle yaptığımız testler, önerdiğimiz bu yöntemle iTLB güvenirliğinden kaynaklanan performans kaybının %2,5'in altına düşürülebileceğini göstermektedir.

Abstract

In concurrent with miniaturizing transistor geometries, hardware components are becoming more sensitive to transient (soft) errors. Integrity checking techniques, such as parity checking and error correcting codes, are among the most commonly used approaches for protecting circuits against transient errors. However, these kinds of approaches introduce delays in frequently accessed hardware components, like iTLBs, so causing the performance degradation of programs. In this study, we present an approach aimed at reducing the performance overheads stemmed from protecting iTLBs against transient errors. To this end, we propose to use a special register storing the last virtual-to-physical page translation for instructions. As long as the execution remains on the same instruction page, address translations are obtained from this special transient-error protected register to minimize the number of iTLB accesses, so avoiding the delays which otherwise would be required for error checking or correcting. Our experimental results carried out by the SimpleScalar tools show that it is possible to reduce the performance overheads in execution time coming from protecting iTLBs against transient errors below 2.5%.

1. Giriş

Gittikçe daha küçük boyutlu transistorların mikroişlemci üretimlerinde kullanılması yeni nesil işlemcileri geçici hatalara (transient/soft errors) [1] karşı daha duyarlı hale getirmektedir. Geçici hataların iki temel kaynağı, ısının etkisiyle paketleme malzemelerinden yayılan alfa parçacıkları ve uzay ışınlarıyla gelen nötron parçacıklarıdır. Yeni nesil işlemcilerde küçülen boyutlara paralel olarak transistor sığasının azalışı, SRAM hücrelerinin çıkışlarında değişiklik için gerekli kritik yükün (Q_{crtq}) azalmasına yol açmaktadır. Böylece bahsi geçen bu yüksek enerjili parçacıkların SRAM hücrelerine çarpması sonucu hücrelerin tuttukları değerlerin kolayca mantıksal 0 değerinden 1 değerine veya 1 değerinden 0 değerine dönüşmesi mümkün olabilmektedir [2].

Günümüzdeki modern işlemcilerde sanal adreslerin fiziksel adreslere dönüştürülmesinden sorumlu bileşenler olarak *Adres Dönüştürme Tamponu* (Translation Lookaside Buffer –TLB) kullanılmaktadır. TLB'ler işlemciye yakın olup, son zamanlarda yapılmış sanal-fiziksel sayfa dönüşümlerini barındırır. Adres dönüştürülmesi sırasında RAM'deki sayfa tablosuna erişim külfetinden işlemciyi kurtardığı için TLB'ler performans üzerinde oldukça önemli rol oynarlar. İşlemciler genellikle iki tip TLB barındırmaktadırlar: komut adresleri dönüşümünden sorumlu komut iTLB ve veri adresleri dönüşümünden sorumlu dTLB.

Diğer işlemci bileşenleri gibi TLB'ler de geçici hatalardan etkilenebilir. Geçici hata sonucu TLB'de tutulan bir dönüşümün bozulması adres dönüşümünün yanlış yapılmasına, dolayısıyla programın yanlış sonuç üretmesine ve hatta programın çökmesine yol açabilir. Bu yüzden TLB'lerin geçici hatalara karşı korunması programların doğru sonuçlar üretmesi açısından son derece önemlidir.

RAM ve SRAM devrelerinde tutulan bilgileri geçici hatalara karşı korumak için geliştirilen ve pratikte çok sık kullanılan iki yöntem *eşlik kontrolü* (parity checking) ve *hata düzeltme kodlarıdır* (error correction codes) [3]. Eşlik kontrolü yönteminde 8 bitlik veriye ekstra 1 bit eklemek suretiyle veride tek sayıdaki bitlerde oluşan hataların tespiti mümkün olmaktadır. Eşlik kontrolü gerçekleştirimi basit ve hızlı olmasına rağmen veride oluşabilecek herhangi bir hatayı düzeltemediğinden fazla yararlı değildir. Önbellek ve TLB

tasarımlarında hata düzeltme yöntemi olarak *tek hata düzeltimi çift hata tespiti* (single error correct double error detect; SECDED) mekanizması yaygın olarak kullanılmaktadır. SECDED yönteminde 64 bitlik bilgi için ekstra 8 bit kullanılarak veride oluşabilecek 1 veya 2 bitlik veri bozulmaları tespit edilebilmekte ve bu bozulmalardan sadece 1 bitlik olanları düzeltilebilmektedir. Yeni nesil işlemcilerde geçici hatalar kümelenmiş bit bozulmalarına (2 veya daha fazla bitte bozulma) yol açabildiğinden SECDED mekanizması tam güvenli bir çözüm sunamamaktadır [4,5]. Ayrıca SECDED veya daha güçlü hata doğrulama mekanizmasıyla korunan SRAM bileşenine erişim çok sıklıkla yapılırsa (örneğin birinci seviye önbellekler veya TLB'lerde olduğu gibi) hata tespiti/düzeltimi için gerekli ekstra zaman erişimleri yavaşlatmakta ve performans üzerinde olumsuz etkilere yol açabilmektedir.

Bu çalışmamızda, iTLB'lerin geçici hatalara karşı korunmasından kaynaklanan performans kaybının azaltılmasına yönelik bir teknik sunmaya çalışmaktayız. Bunun için sistemimizde *aktif çerçeve yazmacı* (current frame register – CFR) diye özel bir yazmacın var olduğunu varsaymaktayız. Bu tekniğe göre sanal sayfanın fiziksel sayfaya dönüşümü bir kez yapılıp, dönüşüm CFR'de saklanır. Program yürütümü mevcut sayfada kaldığı sürece adres dönüşümleri iTLB yerine CFR'den temin edilerek iTLB erişimlerinin sayısı minimize edilir. Böylece iTLB'lerde tutulan adres dönüşümlerine ilişkin verileri SECDED gibi hata tespiti/düzeltimi teknikleri ile performans üzerinde olumsuz etkiye yol açmadan korumak mümkün olabilmektedir. CFR, maliyeti artırmaksızın *kısmi azaltılmış silikon-yalıtkan* (depleted silicon-insulator – SOI) teknolojiyle veya alan ve enerji tüketimini artırmaksızın normal transistorlara göre daha büyük boyutlu transistor tasarımıyla geçici hatalara karşı korunaklı hale getirilebilir. Komut erişimlerinin zamanda ve mekanda yerellik özelliğinden dolayı komut yürütümü mevcut sayfanın dışına genellikle çok nadir olarak çıkmaktadır. Bu husus iTLB erişimlerini minimuma indirgediğinden, iTLB performans maliyetli SECDED gibi mekanizmalarla geçici hatalara karşı korunsa bile performans üzerinde olumsuz etkiye yol açmayacaktır.

2. Önceki Çalışmalar

Kalıcı hatalara göre genelde daha sıklıkla rastlanan geçici hatalardan korunma yöntemlerini genel olarak üç sınıfa ayırabiliriz. Bunlar işlem teknolojisi, devre çözümleri ve mimari düzeyde çözümlerdir [6]. İşlem teknolojisi çözümlerinde transistor tasarımlarında silikon-yalıtkan (SOI) kullanılır. SOI'lu transistorlar daha ince silikon tabakasından dolayı parçacık çarpmalarından daha az yük toplar. Devre düzeyi çözümlerde hücre sığası ve/veya besleme gerilimi gibi transistor parametrelerinde ayarlama yapılarak ışımalara karşı güçlendirilmiş hücreler tasarlanır. Bu iki gruptaki çözümlerin temel problemleri tasarımda önemli ölçüde güç tüketimi ve/veya alan artışına sebep olmalarıdır.

Mimari düzeydeki çözümlerin en yaygın olanları parite kontrolü ve hata düzeltme kodlarıdır. Özellikle SECDED mekanizması SRAM ve DRAM'lerde yaygın olarak kullanılmaktadır. SECDED yöntemi, parite yönteminin aksine 1 bitlik bozulmaları düzeltebilse de hata düzeltme işleminde oldukça yavaştır. Bunun sonucu olarak özellikle sıklıkla erişilen donanım bileşenlerinde kullanımı performans üzerinde olumsuz etkiye sebep olabilmektedir. Bu çalışmada önerdiğimiz bizim çözümümüz de mimari düzeyde olup, hedef alınan iTLB'nin de SECDED veya kümelenmiş hatalara yönelik daha güçlü bir hata düzeltme yöntemiyle korunduğu varsayılmaktadır. Yöntemimizde dönüşümlerin çok büyük bir kısmı CFR yazmacından karşılandığı için iTLB erişimlerinin sayısı çok az olmakta ve dolayısıyla hata düzeltmenin getireceği performans kaybı büyük ölçüde azaltılabilmektedir.

Önceki teknoloji nesillerinde tasarlanan işlemcilerdeki geçici hatalar genellikle izole olmuş ve tek bit bozulması şeklindeki hatalardı. Transistor boyutlarının küçülmesiyle günümüz teknolojisiyle tasarlanan işlemcilerde tek parçacık çarpmasıyla oluşan hatalar kümelenmiş çok bitli hatalar şeklinde olabilmektedir [7,8]. Kümelenmiş çok bitli hataları düzeltebilmek için daha güçlü hata düzeltme yöntemlerine, yani daha güçlü hata düzeltme kodlarına ihtiyaç vardır. Daha güçlü hata düzeltme kodları daha fazla alan gereksinimi ve performans kaybı olarak kendini belli ettirmektedir.

Bu çalışmada bahsi geçen CFR yazmacı daha önceki çalışmalarda veri TLB güç tüketimin azaltılması [9] ve komut getirimi sürecinde işlem değişkenliğinden kaynaklanan performans kayıplarının azaltılmasında [10] kullanılmıştır. Bu çalışmada ise CFR, performans artırımı için kullanılmaktadır. CFR yazmacının temel formatı aşağıdaki gibidir.
[<*Sanal Sayfa Numarası*> <*Fiziksel Çerçeve Numarası*> <*Koruma Bitleri*>]

CFR'nin yukarıdaki formatına bakacak olursak, tipik bir iTLB kaydında bulunması gereken bileşenleri içerdiği görülür. *Sanal sayfa numarası* (physical page number – VPN) işlemcinin ürettiği sanal adrese karşı düşen sanal sayfayı, *fiziksel çerçeve numarası* (physical frame number – PFN) sanal sayfanın fiziksel bellekte yerleştirileceği çerçeveyi, *koruma bitleri* (protection bits – PB) tipik bir sayfa tablosunda bulunan ve ilgili sayfanın yönetimine ilişkin bitleri göstermektedir.

Bu çalışmanın geri kalan kısmı şöyle organize edilmiştir. İzleyen bölümde konuyla ilgili önceki çalışmalardan bahsedilmektedir. 3. Bölümde yöntemimizin ayrıntıları verilmiş, onu izleyen bölümde ise deneysel düzenek tanıtılmaktadır. Deney sonuçları 5. Bölümde sunulmuş olup, 6. Bölüm ise nihai bulguları özetlemektedir.

3. Yöntem

Şimdiye kadarki yazılanlardan anlaşılacağı üzeri, CFR yazmacı sayesinde fiziksel adresleri iTLB'ye erişmeksizin doğrudan üretmeye çalışmaktayız.

3.1. Fiziksel Adreslerin Doğrudan Üretilmesi

CFR yazmacı donanım tarafından yönetilmekte olup son komut sayfasının sanaldan fiziksel adrese dönüşümünü tutmaktadır. Program akışı mevcut sayfada kaldığı sürece, adres dönüşümü CFR yazmacından temin edilir. Böylece iTLB erişiminde geçici hataların kontrolüne yönelik zaman kayıpları bertaraf edilmiş olur. CFR hatalara karşı korunaklı yapıldığından hata kontrolüne gerek duymaz ve tek bir yazmaç özelliği gösterdiğinden erişimi hızlıdır. Eğer program akışı mevcut sayfa sınırları dışına çıkarsa adres dönüşümü

normalde olduğu gibi iTLB'den sağlanır (hata düzeltimi uygulanarak) ve CFR yazmacı bu dönüşümle güncellenir.

Buradaki önemli bir husus, donanımın komut akışının mevcut sayfa dışına çıkıp çıkmadığını nasıl tespit edebileceğidir. Bu konuda çeşitli yaklaşımlar kullanılabilir. Bunlardan en basitini şöyle açıklayabiliriz. Donanım, CFR yazmacının VPN kısmı ile *program sayacının* (*program counter* – PC) VPN kısımlarını karşılaştırır. Eğer uyuşma varsa, CFR yazmacının PFN kısmı fiziksel sayafın numarasını belirler. Aksi halde fiziksel sayfa numarası iTLB erişimi ile belirlenir. Daha sonra PFN bitleri ile PC'nin *sayfa ofset* (page offset) bitleri birleştirilerek ilgili komuta ait fiziksel adres elde edilir.

3.2. Kontrol Akışının Mevcut Sayfa Sınırlarına Çıkıp Çıkmadığının Kontrolü

Yukarıda açıklanan basit yöntemin en önemli dezavantajı her komut erişimi için CFR'nin VPN kısmı ile PC'nin VPN bitlerinin kıyaslanmasıdır. Bu kıyaslama ekstra zaman alıp önbellek erişimini geciktirebilir ve dolayısıyla performansa kötü yönde etki edebilir. Bu, özellikle yüksek performanslı sistemler için sorun teşkil edebilir.

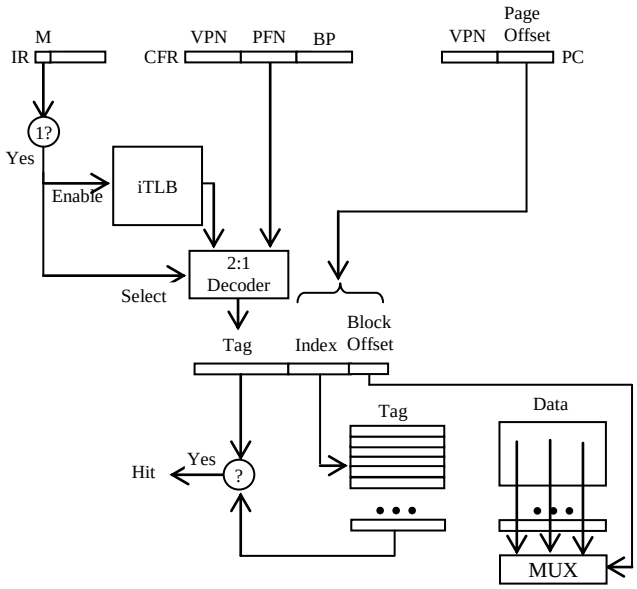
Bu çalışmamızda kontrol akışının mevcut sayfa dışına çıkıp çıkmayacağı kontrolü komutlara eklenen tek bir bit sayesinde yapılmaktadır. Bu bit, *işaretlenmiş bit* (marked bit – M) olarak adlandırılmaktadır. Bu çalışmamızda *komut küme mimarisinde* (instruction set architecture – ISA) komut kodlamalarında kullanılmayan bit pozisyonları olduğunu varsaymaktayız. Birçok işlemci şu an 64-bit mimarisine sahiptir ve bu da bize komutları kodlamak için oldukça fazla sayıda bit pozisyonu sağlamaktadır. Bu bit pozisyonlarından bazıları genellikle boş olup, ileride ihtiyaçlara yöneliktir. Çalışmamızda bu boş bit pozisyonlarından birini M biti olarak kullanarak, sonraki komutun mevcut sayfa sınırları içerisinde kalıp kalmayacağı bilgisi kodlamaktadır.

Bunun için derleyici, kodu inceleyerek koda ait *kontrol akış grafiğini* (kontrol flow graph - CFG) elde eder. Sonraki komutun sayfa dışına çıkıp çıkmadığı bilgisi mevcut komuta ait bahsi geçen boş bit pozisyonuna kodlanır. Örneğin, sonraki komutun sayfa sınırları içerisinde kaldığı bilgisi bu bit 0 ile kodlanarak, sayfa sınırları dışına çıktığı bilgisi 1 ile kodlanarak belirlenir. Çalışma zamanında komut getirimi aşamasında komutun ilgili biti donanım tarafından incelenir. Eğer bu bit 0 ise sonraki komutun mevcut sayfa sınırları içerisinde olduğu tespit edilerek donanımın adres dönüşüm bilgisini CFR yazmacından alması sağlanır. Aksi halde ilgili bit pozisyonunda 1 kodlandığı tespit edilirse donanım adres dönüşüm bilgisinin CFR'de olmadığını ve iTLB'den temin edilmesi gerektiğine karar kılar.

Program akışının bir sayfadan başka bir sayfaya geçmesinin iki yolu vardır. İlki, hedef adresi başka bir sayfaya düşebilen *dallanma komutlarının* (branch) varlığıdır. İkincisi ise sayfa sınırlarına düşen iki komutun var olmasıdır; bir komutun sayfanın son komutu olması ve onu izleyen komutun da sonraki sayfanın ilk komutu olmasıdır. Her iki durumda da derleyici komutun M bitini 1 olarak kodlayarak bu ilgili komutu izleyen komutun adresinin iTLB'den temin edilmesini sağlar. Aksi takdirde M biti 0 olarak kodlanarak izleyen komutun adres dönüşümünün CFR'den elde edilir. Yani komutların bir biti adres dönüşüm kaynağını belirler.

3.3. CFR Varlığında Komut Önbelleğine (iL1) Erişim

Şekil 1, sistemimizde CFR yazmacının var olması durumunda komut önbelleğine erişimin nasıl yapılabileceğini göstermektedir. Şekilden de görülebileceği üzere önbelleğe erişim sürecinde bir kümenin indekslenmesi ve indekslenen kümeye ait önbellek bloklarına ait etiketlerin (tag) adres etiketiyle kıyaslanması yer almaktadır. Pratikte çok sık kullanıldığı için biz bu çalışmada *sanal olarak indekslenmiş, fiziksel etiketlenmiş* (virtually indexed, physically tagged – VI-PT) önbellek erişim yönteminin kullanıldığını varsaymaktayız.



Şekil 1: CFR varlığında iL1 önbellek erişimi. Önbelleğin sanal adreslerle indekslendiği ve etiket kıyaslamasının fiziksel adreslerle yapıldığı (VI-PT) varsayılmaktadır.

VI-PT önbellek erişim mekanizmasında önbelleği indekslemek için sanal adres kullanılır ve indeksleme işlemi esnasında aynı zamanda TLB de paralel olarak erişilir. Bunun ardından fiziksel adresteki etiket, seçili kümenin bütün önbellek bloklarına ait etiketlerle paralel olarak kıyaslanır.

4. Deneysel Düzenek

Araştırmalarımızda SimpleScalar [11] simülatör araçları kullanılmıştır. SimpeScalar günümüzde araştırmalarda yaygın olarak kullanılmakta olup, Linux türevi platformlarda programların performanslarını belirlemeye yönelik yazılım araçları sunmaktadır. Biz bu çalışmada modern işlemcilerde *sıra dışı komut çalıştırılması* (out of order execution) benzeşimini sağlayan *sim-outorder* yazılım aracının kodunu değiştirdik. Deneylerimizde kullandığımız sistem yapılandırma parametreleri Çizelge 1'de verilmektedir. İşlemcimiz 4-yollu Alpha benzeri bir RISC işlemcisidir. Performans ölçümü için kullandığımız programlar ise SPEC2006 [12] takımındandır. SPEC2006 takımından herhangi bir uygulamayı baştan sona kadar simüle etmek çok zaman aldığından, her bir uygulama için uygulamaya özel sayıda komut atlatıldıktan sonra izleyen 2 milyar komut için simülatörle istatistikler topladık.

Deneylerimizde kullandığımız birinci sevideki önbellekler (komutlar için iL1 ve veriler için dL1) 3 aşamalı payplaynlı

(3-stage pipelined) önbellektir ve her bir aşama 1 makine çevrimde (cycle) tamamlanmaktadır. Chishti ve Vijaykumar [13] tarafından önerilen bu önbellek modelinde, önbellek küme adresinin kod çözümü birinci aşamada gerçekleştirilir. Wordline driving, bitline şarjı ve reseptör (sense amplifiers) devrelerden bitline çiftleri arasındaki gerilim farkının gözlemlenmesi ikinci aşamada gerçekleşmektedir. Çıkış çoklayıcıların (multiplexer) ve seçili verinin önbellekten alınması ise sonuncu aşamada gerçekleştirilir. Sonuncu aşamada seçili kümede yer alan önbellek blokların etiketleriyle iTLB'den okunan sayfa dönüşüm bilgisinin karşılaştırma işlemi de yapıldığından, iTLB'de dönüşümün geç gerçekleştirilmesi (hata tespiti/düzeltiminden kaynaklanan gecikmelerden dolayı) bu üçüncü aşamanın uzamasına sebep olacaktır. Bu uzama önbellek erişimin daha geç tamamlanmasına ve dolayısıyla performans kayıplarına sebep olur. Deneylerimizde, iTLB'deki bu gecikmenin iL1 payplaynın üçüncü aşamasını 1 makine çevrimi geciktirerek 2 çevrimde tamamlanmasına yol açtığını ve dolayısıyla iL1 önbellek erişiminin 3 yerine 4 çevrimde bitirildiğini varsaymaktayız.

Adres dönüşüm bilgisi CFR yazmacından temin edildiği durumda ise iTLB'de hata tespiti/düzeltiminden kaynaklanacak gecikmeden sakınılacağı için, normal durumda olduğu gibi payplaynın üçüncü aşaması 1 makine çevriminde tamamlanmakta ve iL1 erişimi 3 çevrimde sonlanmaktadır.

Çizelge 1. İşlemcinin temel yapılandırma parametreleri

İşlemci Çekirdeği	
Fonksiyonel Birimler	8 tamsayı ve 4 FP ALU, 1 tamsayı çarpma/bölme birimi, 1 FP çarpma/bölme birimi
LSQ boyu	64
RUU boyu	128
Fetch/decode/issue/commit width	4 komut/çevrim
Fetch kuyruk boyu	16 komut
Önbellek ve bellek hiyerarşisi	
L1 komut önbelleği	64KB, 4-yollu (LRU), 64 byte blok boyu, 3 çevrim gecikme
L1 veri önbelleği	64KB, 4-yollu, 32 byte blok boyu, 3 çevrim gecikme, write-back
L2 önbelleği	8MB birleşik, 8-yollu (LRU), 128 byte blok boyu, 10 çevrim gecikme
Veri/komut TLB	128 giriş, 30 çevrim ıska cezası, 8 KB sayfa boyu
Memory	160 çevrim gecikme

5. Deneysel Bulgular ve Tartışma

Bu çalışmamızda göz önünde bulundurulan her bir uygulama programı için üç farklı deney gerçekleştirdik. Birinci tip deneyde iTLB'nin hatalara karşı korumasız olduğu ve dolayısıyla iL1 önbellek erişiminin 3 makine çevriminde tamamlandığı varsayılmıştır. Bu durum, geçici hataların oluşmadığı mükemmel duruma karşı geldiğinden bunu *mükemmel* (M) yöntem olarak adlandırmaktayız.

İkinci tip deneyde iTLB'nin hatalara karşı korunduğunu ve her erişimde hata kontrolü (ve varsa düzeltmesi yapıldığı) varsayılmaktadır. iTLB'deki geçici hatalara karşı korumadan dolayı iL1 önbelleğinin son aşaması 2 çevrimde ve dolayısıyla önbellek erişiminin de toplam 4 çevrimde tamamlandığı düşünülmektedir. Bu durum geçici hatlara karşı korumalı bir iTLB'yi yansıttığından bu yöntemi korumalı (K) yöntem olarak adlandırmaktayız.

Üçüncü tip deneyde, önerdiğimiz CFR yazmacının sistemde var olduğunu düşünmekteyiz. Burada da iTLB, K yönteminde olduğu gibi geçici hatalara karşı korunmaktadır. Eğer sanaldan fiziksele dönüşüm bilgisi CFR yazmacından sağlanırsa, iL1 erişimi M'de olduğu gibi 3 makine çevriminde tamamlanmaktadır. Eğer dönüşüm CFR'de mevcut değilse, bu durumda adres dönüşümü iTLB'den sağlanır ve iL1 erişimi K'de olduğu gibi 4 çevrimde tamamlanarak, CFR yazmacı iTLB'den temin edilen dönüşüm bilgisi ile güncellenir. Dolayısıyla daha sonraki adres dönüşümünün CFR'den temin edilmesine çalışılmış olunur. CFR yazmacının kullanılmasından dolayı bu yöntemi CFR yöntemi olarak adlandırmaktayız.

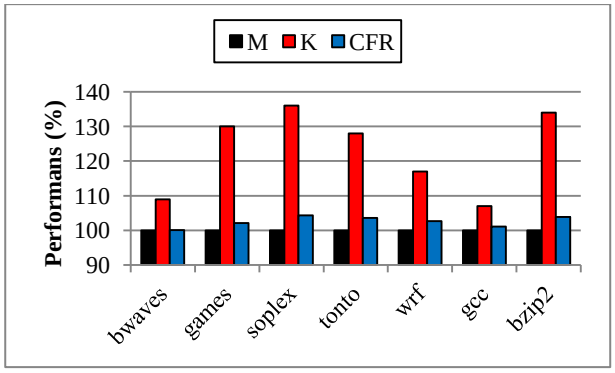
Çizelge 2. Deneylerimizdeki Uygulama Programları

program	iTLB Er. Say.	Çev. Say.	CFR Gün. Oran.
bwaves	1226505191	1267101890	%0,1
gamess	567102063	744310032	%3,4
soplex	729410432	580240912	%5,2
tonto	535960321	794250208	%4,0
wrf	664053109	842898102	%2,9
gcc	705671902	934201673	%1,2
bzip2	607812769	820904519	%1,8

Çizelge 2'de, M yöntemi uygulandığında kullanılan programlara ait elde edilen temel özellikler gösterilmektedir. Çizelgeden de görüleceği üzeri, deneylerimizi SPEC2006 takımından 7 farklı program üzerinde gerçekleştirdik. Daha önce de belirtildiği üzere deneylerde her bir uygulama programı için o programa özel sayıda komut atlanılmış [14] ve sonraki izleyen 2 milyar komut çalıştırılarak simülatörden istatistikler toplanmıştır. Çizelgenin ikinci sütununda her bir uygulama programı için iTLB erişimleri, üçüncü sütununda çevrim sayıları ve sonuncu sütununda ise CFR yazmacının güncelleme sayıları (program akışının kaç kez mevcut sayfa sayfası dışına çıktığı) verilmektedir.

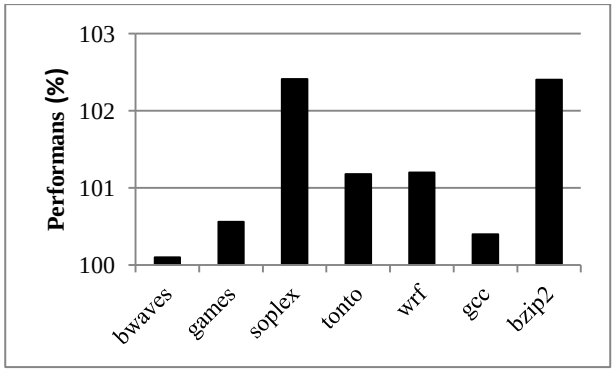
Deney yaptığımız 7 uygulama programından 6'sında program akışı %96 veya daha büyük bir ihtimalle aynı sayfa içerisinde devam etmektedir. Sadece tek bir uygumla programı için program akışı %5'ten biraz daha büyük bir ihtimalle mevcut sayfa dışına çıkmaktadır. Çok yüksek bir ihtimalle program akışının aynı sayfa içerisinde kalmaya devam etmesinin temel sebebi komut yürütümlerindeki yerelliktir. Zaten programların bu özelliği, önbelleklerin performans üzerinde sağladıkları çok büyük katkının temel sebebidir.

Program akışının mevcut sayfanın içinde kalması adres dönüşümlerinin iTLB yerine CFR yazmacından temin edilmesine olanak sağladığından, performans artışları mümkün olabilmektedir.



Şekil 2: M, K ve CFR yöntemlerine yönelik performans değerleri.

Şekil 2'de M, K ve CFR düzenekleri ile yapılan deney sonuçları gösterilmektedir. Grafikten de görüleceği gibi her bir uygulama programı için 3 çubuk gözükmektedir. Bu çubuklardan ilki geçici hataların olmadığı ve dolayısıyla iTLB'nin korumaya ihtiyaç duyulmadığı mükemmel duruma karşı gelen performans değerlerini göstermektedir. İkinci çubuk ise her bir iTLB için hata tespiti/düzeltimi yapılan korumalı duruma karşı gelmektedir. Son çubuk ise iTLB'de hata tespiti/düzeltimi yapıldığı ve aynı zamanda sistemde CFR'nin de bulunduğu duruma karşı gelmektedir. Deneysel sonuçlarımız, K yönteminin ortalama %22,21 civarı performans kaybına yol açtığını göstermektedir. CFR yöntemi ise bu performans kaybının önemli bir kısmını bertaraf ederek kaybın %2,48 ile sınırlandırılmasına yol açmaktadır.



Şekil 3: Sayfa boyunun 16K olması durumunda CFR yöntemine ilişkin performans değerleri.

Şu ana kadar yapılan deneylerde sayfa boyunun 8K olduğu varsayılmıştı. CFR yönteminin sayfa boyuna olan duyarlılığını ölçmek için sayfa uzunluğunu 8K'dan 16K'ya çıkararak yeni deneyler yaptık. Elde edilen performans sonuçları Şekil 3'de gösterilmektedir. Sayfa boyunun değiştirilmesi M ve K yöntemlerinin performans değerlerine pek etki etmediği için burada sadece CFR yöntemine ilişkin değerler verilmektedir. Korumalı iTLB ile birlikte CFR yazmacı kullanıldığında performans kaybı %1,18'e düşmektedir. Bu düşüşün temel sebebi, sayfa boyunun 8K'dan 16K'ya çıkarılması kontrol akışının mevcut sayfa dışına çıkma sıklığını düşürmesi ve dolayısıyla sanal adresten fiziksel adrese dönüşümün CFR yazmacından elde edilme sıklığını artırmasıdır. 16K'dan daha büyük sayfa boyutları pratikte kullanılmadığı için daha büyük boyutlu sayfalar için CFR yöntemini test etmedik.

6. Kararlar

Transistor boyutlarının küçülmesi sonucu yüksek enerjili parçacıkların devrelere çarpması sonucu SRAM hücrelerinin değişmesi daha da kolaylaşmaktadır. Devreleri geçici hatalara karşı korumak için çeşitli hata tespiti/düzeltimi kodları kullanılmaktadır. Ancak bu kodların özellikle iTLB gibi çok sıklıkta erişilen işlemci birimlerinde kullanılması erişimleri oldukça geciktirmekte ve dolayısıyla performans kayıplarına sebep olmaktadır. Bu çalışmamızda CFR yazmacı sayesinde sanal-fiziksel adres dönüşümünü doğrudan üreterek, iTLB'ye erişim sıklığını azaltıp hata kontrollerinin performans üzerinde oluşturduğu etkiyi minimize etmeye çalıştık. Yaptığımız testler sonucu 8K boyutlu sayfaların kullanıldığı bir sistem için performans kaybının ortalama %22,21'den %2,48'e düşürülmesinin mümkün olduğunu gözlemledik.

7. Kaynaklar

[1] 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.

[2] 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.

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

[4] Mukherjee, S. S., Weaver, C. T., Emer, J., Reinhardt, S. K., and Austin, T., "A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor", the IEEE/ACM *Intl. Symp. on Microarchitecture*, 2003, 29-40.

[5] Osada, K., Yamaguchi, K., Saitoh, Y., and Kawahara, T., "SRAM immunity to cosmic-ray-induced multierrors based on analysis of induced parasitic bipolar effect", *IEEE Journal of Solid-State Circuits*,2004, 827-833.

[6] Mukherjee, S. S., Emer, J., and Reinhardt, S. K., "The soft error problem: an architectural perspective", the *Int. Symposium on High Performance Computer Architecture*, 2005, 243-247.

[7] George, N. J., Elks, C. R., Johnson, B. W., and Lach, J., "Bit-slice logic interleaving for spatial multi-bit soft-error tolerance", the *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, 2010, 141-150.

[8] Szafaryn, L. G., Meyer, B. H., and Skadron, K., "Evaluating overheads of multibit soft-error protection in the processor core", *IEEE Micro*, 2013, 33(4):56-65.

[9] Kadayif, I., Nath, P., Kandemir, M., and Sivasubramaniam, A., "Reducing data TLB power via compiler-directed address generation", *IEEE Trans. Comp. Aided Des. Integr. Circ. Syst.*, 2007, 26(2):312-324.

[10] Kadayif, I, Turkcan, M., Kiziltepe, S., and Ozturk, O, "Hardware/software approaches for reducing the process variation effect on instruction fetches", *ACM Trans. on Des. Auto. of Elect. Sys.*, 2013, 18(4):54.

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

[12] SPEC2006 Benchmark. http://www.spec.org

[13] Chishti, Z. and Vijaykumar, T. N., "Wire delay is not a problem for SMT (in the near future) ", the *Int. Symposium on Computer Architecture*, 2004, 137-148.

[14] Hamerly, G., Perelman, E., Lau, J., and Calder, B., "Simpoint 3.0: faster and more flexible program analysis", Journal of Instruction-Level Parallelism, 2005, 7:1-28.