ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
YÜKSEK LİSANS TEZİ

ÖLÜ BLOKLARA DAYALI VERİ ÖNBELLEĞİ GÜVENİRLİĞİNİN ARTTIRILMASI

Davut AKÇİÇEK

Yrd.Doç.Dr. İsmail KADAYIF

Mayıs, 2007
ÇANAKKALE

ÖLÜ BLOKLARA DAYALI VERİ ÖNBELLEĞİ GÜVENİRLİĞİNİN ARTTIRILMASI

Çanakkale Onsekiz Mart Üniversitesi Fen Bilimleri Enstitüsü
Yüksek Lisans Tezi
Bilgisayar Mühendisliği Anabilim Dalı

Davut AKÇİÇEK

Yrd.Doç.Dr. İsmail KADAYIF

Mayıs, 2007
ÇANAKKALE

Çanakkale Onsekiz Mart Üniversitesi Fen Bilimleri Enstitüsü Müdürlüğüne,

Bu araştırma, jürimiz tarafından Bilgisayar Mühendisliği Anabilim Dalı'nda Yüksek Lisans Tezi olarak kabul edilmiştir.

Başkan : Yrd. Doç. Dr. İsmail KADAYIF

Üye : Yrd. Doç. Dr. İbrahim TÜRKYILMAZ

Üye : Yrd. Doç. Dr. Önder AYYILDIZ

Kod No :

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım.

Enstitü Müdürü

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

DAVUT AKÇİÇEK, tarafından **YRD.DOÇ.DR.İSMAİL KADAYIF** yönetiminde hazırlanan **"ÖLÜ BLOKLARA DAYALI VERİ ÖNBELLEĞİ GÜVENİRLİĞİNİN ARTTIRILMASI"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

Yrd.Doç.Dr. İsmail KADAYIF

Yönetici

Yrd.Doç.Dr. İbrahim TÜRKYILMAZ

Jüri Üyesi

Yrd.Doç.Dr.Önder AYYILDIZ

Jüri Üyesi

Prof.Dr.Mehmet Emin ÖZEL

Müdür

Fen Bilimleri Enstitüsü

TEŞEKKÜR

Öncelikle bu çalışmanın ortaya çıkmasında benden bilgi, deneyim ve yardımlarını esirgemeyen danışmanım Yrd.Doç.Dr. İsmail KADAYIF'a saygı ve şükranlarımı sunuyorum.

Maddi ve manevi destekleri ile beni bugünlere getiren aileme sonsuz sevgilerimi iletiyorum.

Tez çalışmamda benden bilgilerini esirgemeyen çalışma arkadaşlarım Arş.Gör.Olcay KABAL ve Arş.Gör.Selçuk KOYUNCU'ya teşekkür ediyorum.

Davut AKÇİÇEK

2007

SİMGELER VE KISALTMALAR

ACE (Architecturally Correct Execution): Mimari doğru çalışabilirlik

AVF (Architectural Vulnerability Factor): Mimari bozulabilirlik faktörü

CPU (Central Processing Unit): Merkezi İşlem Birimi

CVF (Cache Vulnerability Factor): Önbellek bozulabilirlik faktörü

D-Cache (Data Cache): Veri önbelleği

DRAM (Dynamic RAM): Dinamik RAM

DUE (Detected Unrecoverable Error): Sezilebilen düzeltilemeyen hata

ECC (Error Correction Code): Hata doğrulama kodu

FIFO(First In First Out): İlk giren ilk çıkar

FIT (Failure In Time): Hata zamanı

I-Cache (Instruction Cache): Komut Önbelleği

ICR (In Cache Replication): Önbellek içinde kopyalama

L1 (Level 1): Birinci seviye

L2 (Level 2): İkinci seviye

LRU (Least Recently Used): Son zamanlarda en az kullanılan

nm: Nanometre

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

ns: Nanosaniye

PC: Program Counter

RAM (Random Access Memory): Rasgele erişimli bellek

RC (Replication Cache): Kopyalama önbelleği

SECDED: 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

TLB (Translation Lookaside Buffer): Sanal adres-fiziksel adres dönüştürücü tampon bellek

IMPROVING DATA CACHE RELIABILITY BASED ON DEAD BLOCKS

ABSTRACT

Soft errors due to energetic particle strikes are a big concern for systems to run in a reliable manner. This reliability concern have been more serious with technology scaling and aggressive leakage control mechanisms. Since cache memories consumes the largest fraction of on-chip real estate, they are more vulnerable to soft errors, as compared to many other components. This thesis proposes a solution to the problem of designing a reliable data cache without trading reliability for performance and area, which is a typical characteristic of conventional ECC and parity based protection techniques. Although parity is simple and fast, it can detect only odd numbered errors without correcting any of them. On the other hand, ECC techniques are more complex and time-consuming, and have the capability by storing the replica(s) of data items in active use into cache lines which hold data not likely to be reused. The bookkeeping information about replicas is maintained in a small fully associative cache called shadow cache. By exploiting the replicas to correct the soft errors enhances the data reliability. Since we keep the replicas in potentially dead blocks, the performance loss is negligible with a little extra chip area requirement for the shadow cache. Our experimental results indicate that our technique, compared to the previous similar techniques, is more effective for enhancing the L1 data cache reliability in modern Superscalar machines with only negligible degradation in performance.

Keywords : soft error, cache reliability, shadow cache, replica

ÖLÜ BLOKLARA DAYALI VERİ ÖNBELLEĞİ GÜVENİRLİĞİNİN ARTTIRILMASI

Özet

Enerji yüklü parçacık çarpmasına bağlı olan soft errorlar güvenirli sistem için ciddi bir endişe oluşturmaktadır. Bu güvenirlilik endişesi, teknoloji ve sızıntı enerji kontrol mekanizmaları geliştikçe artmaktadır. Önbellekler, günümüzde işlemci çip yüzeyinin en geniş alanını kapladığından, CPU'nun diğer bileşenleri ile kıyaslandığında soft errorlara karşı daha bozulabilirdirler.

Bu çalışma, güvenilir veri önbelleği tasarlama problemine parite ve ECC temelli koruma teknikleri gibi performanstan ve alandan fazla kayıp vermeden çözüm sunmaktadır. Parite yöntemi basit ve hızlı olmasına rağmen tek sayılı hataları sezebilir ama düzeltemez. Diğer bir taraftan, ECC teknikleri karmaşık ve daha fazla zaman harcamaktadır. Ancak bu teknikler hataların bir kısmını düzeltebilmektedir. Bizim tekniğimiz, aktif kullanımda ki verilerin kopyalarını, verisi yakın zamanda kullanılmayacak önbellek satırları içinde depolayarak veri önbelleği güvenirliliğini geliştirmektedir. Kopyalar hakkındaki bilgi gölge önbellek adı verilen küçük ve tam çağrışımlı bir önbellekte tutulur. Kopyaları kullanarak soft errorların düzeltilmesi veri güvenirliliğini artırmaktadır. Kopyaları potansiyel ölü bloklarda sakladığımızdan gölge önbellek için fazladan çip alanı ihtiyacı ile performans kaybı önemsenmeyecek kadar azdır. Deneysel sonuçlarımız gösteriyor ki, bizim tekniğimiz daha önce önerilen benzer teknikler ile kıyaslandığında, modern Superscalar makinelerde performanstan önemsenmeyecek bir kayıpla L1 veri önbelleği güvenirliliğini geliştirmede daha verimlidir.

Anahtar sözcükler: Soft error, önbellek güvenirliliği, gölge önbellek, kopyalama.

İÇERİK **Sayfa**

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

BÖLÜM 1 – GİRİŞ 1

BÖLÜM 2 – ÖN BELLEKLERE GENEL BAKIŞ 2
2.1. Önbellek 2
2.2. Önbellek Mimarisi 4
2.3. Önbelleğe Yazma Yöntemleri 7
2.3.1. Doğrudan Yazma (Write-Through) 7
2.3.2. Geriye Yazma (Write-Back) 8
2.4. Önbellek Çeşitleri 9
2.4.1. Komut Önbelleği 10
2.4.2. Veri Önbelleği 10
2.4.3. TLB Önbelleği 10
2.5. Önbellek Organizasyonları 11
2.5.1. Doğrudan Haritalanmış Önbellek (Direct Mapped Cache) 12
2.5.2. Tam Çağrışımlı Önbellek (Fully Associative Cache) 15
2.5.3. Küme Çağrışımlı Önbellek (Set Associative Cache) 16
2.6. Yer Değiştirme Algoritmaları 18
2.6.1. Rasgele Algoritması (Random) 18
2.6.2. İlk Giren İlk Çıkar Algoritması (First In Firs Out-FIFO) 18
2.6.3. Son Zamanlarda En Az Kullanılan Algoritması (Least Recently Used-LRU) 18

BÖLÜM 3 – SOFT ERRORLAR VE ÖNBELLEKLERE ETKİLERİ 19
3.1. Soft Error Nedir? 19

3.2. Soft Error Sebepleri 24
3.2.1. Pakajlama Bozulmaları (Package Decay)..... 24
3.2.2. Kritik Şarj (Critical Charge) 24
3.2.3. Kozmik Işınlar (Cosmic Rays)..... 24
3.2.3. Diğer Sebepler 25
3.3. Soft Error Türleri..... 25
3.4. Soft Errorlar Nasıl Ölçülür? 26
3.4.1. SDC Oranının Hesaplanması..... 28
3.4.2. Mimari Bozulabilirlik Faktörü (Architectural Vulnerability Factor)..... 29
3.4.3. Önbellek Bozulabilirlik Faktörü (Cache Vulnerability Factor).. 31
3.5. Soft Error Etkileri 32
3.6. Mimari Temelli Güvenirliliği Artırma Teknikleri 32

BÖLÜM 4 – BİZİM YAKLAŞIMIMIZ ve DENEYSEL SONUÇLAR..... 36
4.1. Soft Error Bilinçli Önbellek Mimarisi..... 36
4.2. Düşünülen Yöntemler..... 39
4.3. Deneysel Ayarlamalar (Experimental Setup) 42
4.3.1. Simülasyon Ortamı (Simulation Environment)..... 42
4.3.2. Değerlendirme Ölçümleri..... 44
4.3.3. Değerlendirme Sonuçları..... 45
4.3.4. Performans Sonuçları..... 48
4.3.5. SC Yöntemini Geliştirme 49

BÖLÜM 5 – SONUÇ VE TARTIŞMA 51

KAYNAKLAR 53

Çizelgeler I
Şekiller II
Yaşam Öyküsü.....III

BÖLÜM 1

GİRİŞ

Intel'in kurucularından Gordon Moore'nun Moore Yasası olarak bilinen öngörüsüne göre, bir çip üzerindeki transistör sayısı her iki yılda ikiye katlanmakta, bu da daha fazla özellik, performans artışı ve transistör başına azalan maliyet sonucunu doğurmaktadır. Transistörler küçüldükçe, birim alanda güç artışı ve ısı yayılımı sorunları ortaya çıkmaktadır. Sonuç olarak, yeni özellikleri, teknikleri ve yapıları uygulamak bu gelişmeyi sürdürmekle mümkün olmaktadır. Silikon entegrasyonu hakkındaki bu düşünce, Intel tarafından gerçeğe dönüştürüldü ve dünya çapında teknolojik ilerlemenin simgesi haline geldi (Hiremane, 2005).

Enerjili tanecik darbelerinin sebep olduğu soft errorlar, özellikle gürültü barındıran ortamlarda çalışan hesaplama sistemleri için önemli bir güvenirlilik kaygısı ortaya çıkarmaktadır. Teknolojideki hızlı gelişmeler ve sızıntı denetim mekanizmaları bu geçici hatalardan kaynaklanan problemi çok daha şiddetlendirmektedir. Bundan dolayı, işlemci/bellek tasarımlarında soft errorlardan korunmak için güvenirliliği arttırıcı mekanizmaların kullanılması çok önemlidir.

Soft errorlar, nadir olaylar olduğu için, buna bağlı hata kontrolleri çok hızlı yapılmalıdır. Aksi halde programların çalışma sürelerinde önemli ölçüde artışlar olacaktır. Hata kontrolü için kullanılan yöntemler, ECC ve parite temelli çözümlerdir. Bu çalışma, bu soruna önbellek içinde aktif blokların kopyalarının tutulması mantığına dayanan yeni bir çözüm sunmaktadır. Yöntem; aktif kullanımdaki veri bloklarının yine önbellek içinde yakın zamanda kullanılmayacak olan veri blokları üzerinde kopyalarının oluşturulması temeline dayanır. Deneysel sonuçlarımızdan görüleceği gibi, bu yöntemle önbellekten okunan verilerin büyük çoğunluğunun kopyası mevcuttur. Yöntemin temel felsefesi, hatalı veri barındıran bir bloğa erişildiğinde (ki bu parite veya ECC ile mümkün), bu blok mümkünse kopyası ile yer değiştirilmek suretiyle hatadan arındırılmış olur.

BÖLÜM 2

ÖN BELLEKLERE GENEL BAKIŞ

2.1. Önbellek

Yarı iletken teknolojisindeki gelişmelere rağmen, mikroişlemci teknolojisi bellek teknolojisinden daha hızlı gelişmektedir. Dolayısıyla mikroişlemciler ana bellekten daha hızlı çalışmaktadır. Yani verilerin işlenmesi, CPU'ya getirilmelerine kıyasla daha hızlı olmaktadır. Bu da uygulamaların çalışmaları esnasında bellek erişim darboğazına sebep olmakta ve çalışma süresinin uzamasına yol açmaktadır. Diğer bir ifadeyle, uygulamanın bellekten getirilmesi çok zaman aldığından CPU'nun çalıştırma hızından tam kapasiteyle faydalanılamamaktadır. Bu da performansın düşmesine sebep olmaktadır. Yüksek çalışma hızına sahip CPU ve yavaş erişimli ana bellek arasında ki bu sıkıntıyı ortadan kaldırmak için, CPU ile ana bellek arasına küçük ve yüksek hızlı bir önbellek koyulması fikri ortaya atılmıştır. Bu küçük ve ana belleğe oranla çok hızlı önbellek, ana bellekte tutulan ve yakın gelecekte sıklıkla erişilecek olan verilerin bir kopyasını tutmaktadır.

Uygulamalarda kullanılacak verilerin ana bellekten getirilme süreleri çok uzun olduğundan, bu veriler önbellekte saklanır. Çünkü önbelleğe erişim zamanı ana belleğe erişim zamanından daha kısadır. Veriler ana bellekten önbelleğe getirilirken de yavaş gelmektedir. Ancak, ihtiyaç duyulan verilerin çoğu ön bellekte bulunabildiğinden ana belleğe erişim sayısı azalacak, dolayısıyla performans artacaktır (Sun, 2002).

Yakın zamanda erişilmiş ve tekrar erişilme olasılığı bulunan verileri ve komutları tutan önbellekler genellikle ayrık tasarlanırlar. Veri ana bellekte ilk defa erişildiğinde CPU'ya getirilmeden önce önbelleğe yazılır. Önbellekten CPU'ya getirilir. Veriler CPU'da işlendikten sonra, işlenen ve üretilen veriler önce önbelleğe yazılır. Daha sonra önbellekten ana belleğe yazılır. Veriye CPU tarafından ihtiyaç

duyulduğunda önce önbellekte aranır. Verinin önbellekte olup olmadığını kontrol etmek için gereken ilgili verinin adresinin bir kısmı da ilgili önbellek bloğunda saklanır. Aranılan veri önbellekte ise işlenmek üzere CPU'ya getirilir. Bulunamaz ise aramak için ana belleğe erişilir. Ana bellekte bulunan veri CPU'ya getirilirken önbelleğe de yazılır (Howe, 2006).

Önbellekler CPU'ya ana belleğe oranla daha yakın olduğundan (genelde CPU ile aynı çip içerisinde bulunurlar) ve tazelemeye ihtiyaç duymadıklarından daha hızlıdırlar. CPU'ya en yakın olan önbelleğe birinci seviye önbellek (L1-Level 1) denir. Çoğu tasarımda birinci seviye önbellek CPU ile aynı çip üzerinde bulunmaktadır. Mikroişlemci çipinin üzerinde bulunması performansı daha çok artırır. Mikroişlemci çipinin dışında ve ana belleğe daha yakın olan önbelleğe ise ikinci seviye önbellek (L2-Level 2) denmektedir (Hoewe, 2006).

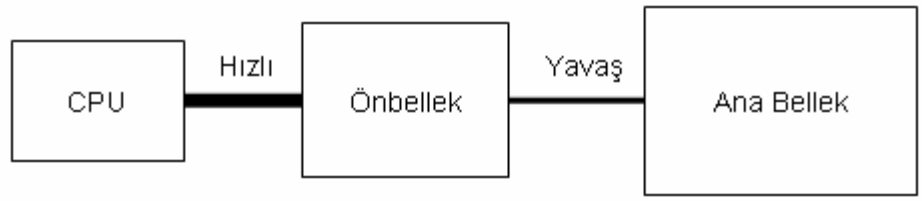
Bellek erişimlerinde önbellekten faydalanılma yüzdesine isabet oranı denmektedir. Bu oran önbelleğin fiziksel tasarımına ve boyutuna bağlıdır. Önbellek boyutunun küçük oluşunun sebebi ise çiplerinin çok pahalı olmasıdır (Hoewe, 2006).

Neden dinamik ana bellek statik bellek ile yer değiştirmiyor?

Dinamik ana bellek ile statik belleğin yer değiştirememesinin temel nedeni statik belleğin fiyatının çok yüksek olmasıdır. Statik bellek dinamik bellekten birkaç kat daha pahalıdır. Hem de statik bellek dinamik belleğe göre daha fazla enerji kullanır ve daha çok yoğundur. Dolayısıyla dinamik bellek ile statik belleğin yerlerini değiştirmek çok pahalıya mal olacaktır (Intel, 2006).

2.2. Önbellek Mimarisi

Şekil 2.1. ‘de örnek temel bir bellek sistemi mimarisi gösterilmektedir (Sun, 2002).



Şekil 2. 1 - Temel bellek sistemi

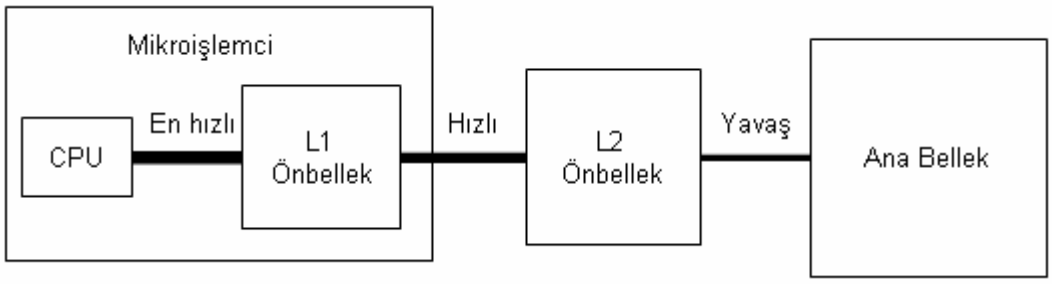
Bu şeklin anlamı, CPU’nun ürettiği verilerin ana belleğe yazılması için öncelikle önbelleğe yazılması gerekliliğidir. Aynı şekilde CPU’nun ihtiyaç duyduğu verinin ana bellekten getirilirken önbelleğe de yazılması gerekmektedir. Arama yapılacağı zaman, ana belleğe erişim maliyeti yüksek olduğu için öncelikle önbellekten aranılması gerekmektedir.

Önbellekte veri arandığında ve bulunamadığında ana belleğe gidilir. Bellek transfer zamanını amorti etmek için, önbelleğe veriler bir blok halinde getirilir. Transfer edilen birime önbellek bloğu (cache block) veya önbellek satırı (cache line) denir. Yani ana bellekten içerisinde aranan veriyi barındıran bir satır önbelleğe getirilir. Bu satır, istenen veriyi içerdiğini garanti eder. Bu tasarım, eğer satırlardaki verilere ardışık olarak erişim varsa çok faydalıdır. Ama verilere düzensiz bir erişim var ise daha az verimli olur (Sun, 2002).

Çip teknolojisindeki gelişmeler, çok seviyeli önbellek tasarımını mümkün kılmıştır. Genelde birinci seviye önbellek mikroişlemci çipinin üzerinde bulunmaktadır (on-chip). Diğer önbellek seviyeleri mikroişlemci çipinin dışında bulunur. Son zamanlarda ki tasarımların bir kısmında L2 önbelleği CPU ile aynı çip içerisine konulmaktadır. Örneğin, AMD işlemcilerinin “AMD Athlon™ 64” ve

"AMD Athlon™ 64 X2 Dual-Core" modellerinde L2 önbelleği işlemci ile aynı çip üzerindedir (AMD, 2006).

En yüksek seviye işlemciye en yakın belleğe en uzak olan önbellektir ve L1 önbelleği diye adlandırılır. Buna Şekil 2.2 bir örnektir (Sun, 2002).



Şekil 2. 2 - Bellek sisteminin çoklu seviyeleri

CPU ile mikroişlemci birbirinden farklı kavramlardır. CPU mikroişlemcinin bir bölümüdür, mikroişlemci, CPU'yu da içinde barındıran çipin tamamına denmektedir. Şekil 2.2'de görüldüğü gibi, soldan sağa doğru gidildiğinde yani CPU'dan uzaklaşıldığında önbellek boyutu artmakta, hız ise azalmaktadır. Başka bir deyişle soldan sağa doğru gidildikçe kapasite artarken, veri transferi daha uzun zaman almaktadır (Sun, 2002). Önbellek ile ilgili bazı kavramlar aşağıda verilmektedir.

İsabet (Hit): İstenilen ve aranılan bilginin önbellekte bulunması (bulunabilmesi) durumuna denir (Intel, 2006).

Iska (Miss): İşlemcinin ihtiyaç duyduğu bellek bölgesini önbellekte bulamayıp aramak için ana belleğe gitmesine denir.

İsabet (Hit) Oranı: Aranılan verilerin yüzde kaçının önbellekte bulunduğunu ifade eder. Önbellek boyutu ve politikaları isabet oranını etkiler (Handy, 1998).

1960 yıllarında IBM firmasının araştırmacıları bazı komutların ardışık ve tekrarlı çalıştığının farkına vardılar ve bunu program lokalitesi olarak adlandırdılar. Program lokalitesi iş akışının iyileştirilmesi için büyük bir adım oldu. Eğer tekrarlı

çalışan herhangi bir program ve program verileri küçük ve yüksek hızlı bir bellekte depolanırsa, daha yavaş ve büyük bir bellekte çalışmasına oranla daha kısa bir sürede sonlanacağı kanısına varıldı. Lokalite iki şekilde olabilir.

Uzaysal Lokalite (Spatial Locality): Program komutlarına/verilerine genelde bellekte ardışık olarak erişilmektedir. Yani, bellekte bir komuta/veriye erişildiğinde, sonraki erişim büyük bir ihtimalle önceki erişilen komutu/veriyi izleyen komuta/veriye olacaktır. Uzaysal lokaliteden dolayı veriler/komutlar önbelleğe bloklar halinde getirilir.

Zamansal Lokalite (Temporal Locality): İşlemci yakın zamanda eriştiği bellek bölgesine kısa bir süre sonra tekrar ihtiyaç duyabilmektedir. Bir önbellek bölgesinin erişildikten sonra belirli bir süre önbellekte saklanması zamansal lokaliteyi açıklar. Zaman geçtikçe aynı bloğun tekrar erişilme olasılığı azalmaktadır. Bu iki lokalitenin kullanılması, ardıl veya tekrarlı bellek adreslerine ihtiyaç duyan programların çalıştırılmasında ana belleğe erişimleri azaltmıştır (Handy, 1998).

Gecikme, Bant Genişliği ve Bellek Alt Sistemi

Gecikme (Latency) ve bant genişliği (Bandwith) önbellek ve ana bellekle ilişkili iki ölçüm değeridir. Gecikme, çoğu zaman işlemci saat periyodu veya nanosaniye ile ölçülür. Bant genişliği ise MB/sn veya GB/sn ile ölçülür.

Bir belleğin gecikmesi, bir üst seviye bellekten bir bellek satırının getirilmesi veya transferi için geçen zamandır. Bu transfer zamanı, transferin yapıldığı belleğin hiyerarşideki pozisyonuna göre değişir. Ama bir kural olarak, Şekil 2.2'de soldan sağa doğru gidildiğinde gecikmenin arttığı söylenebilir.

Bellek bileşenlerinin bazıları (L1 önbelleği), fiziksel olarak mikroişlemci

çipinin üzerinde olabilir. Bu bileşenlerin gecikmesini nanosaniye yerine işlemci saat peryoduyla (cycle) ölçmek daha mantıklı olacaktır.

Bant genişliği bellek bileşeninin başka bir hız ölçüm değeridir. Bu değer geniş boyutlu verileri ne kadar hızlı okuyup yazabileceğini ölçer. Bant genişliği CPU'dan uzağa gittikçe azalır (Sun, 2002).

Önbellek alt sistemi, üç fonksiyonel birimle ifade edilebilir: SRAM, TagRAM ve Önbellek Denetleyici. Fiziksel tasarımda bunlar çoklu çipler veya tekil çip üzerinde olabilir.

SRAM: Veri tutan bellek bloğudur. Önbelleğin boyutuna göre belirlenir.

TagRAM: SRAM'de tutulan verilerin adreslerini tutar.

Önbellek Denetleyicisi: Önbelleğin yazılması, okunması ve güncellenmesi işlerinden sorumludur. Aynı zamanda isabet ve ıska olmasından da sorumludur (Intel, 2006).

2.3. Önbelleğe Yazma Yöntemleri

Önbelleğe veri yazılırken istenen en önemli şey yazılan verilerin tutarlılığı ve hızıdır. Yazma işlemi için iki farklı önbellek tasarımı kullanılmaktadır:

2.3.1. Doğrudan Yazma (Write-Through)

Doğrudan yazma yönteminde veri önbelleğe her yazıldığında ana belleğe de yazılır. Geliştirmesi daha kolay ve ekonomiktir. Her önbellek erişiminde ana belleğe de erişme ihtiyacı duyduğundan performansı oldukça düşüktür. Buna karşın işletimi

basit olduğundan geliştirimi ucuz ve kolaydır. Önbellek içinde yapılan her değişiklikte veya önbelleğe her yazma erişiminde ana belleğe de yazılması, veri tutarlılığını sağlamasına rağmen işlemcinin hızını düşürür (Intel, 2006).

2.3.2. Geriye Yazma (Write-Back)

Bu yöntem sadece, verinin önbellek bloğundan çıkarılacağı zaman ana belleğe yazılmasını sağlar (Howe, 2006). Yani önbellek geçici bir bellek olarak kullanılır. Önbellek bloğunun üzerine yeni bir blok yazılacağı zamana kadar yani önbellekten çıkarılmaya zorlandığı ana kadar işlemci ürettiği verileri önbelleğe yazar, ne zaman ki blok önbellekten çıkarılmaya zorlanırsa, o zaman o blok içindeki veri belleğe yazılır. Bu yöntem, önbellekteki blok üzerine yazılıncaya kadar sadece önbelleğe yazdığından ana belleğe erişim sayısını azaltır. Bu yöntem daha performanslı ancak geliştirimi daha zordur.

Geriye yazmalı önbelleklerin gerçekleştiriminin zor olmasının sebebi, verinin ne zamana kadar önbelleğe yazılacağının takip edilmesi gerekliliğidir. Takip edebilmek için bloğun geçerli olup olmadığını kontrol eden geçerlilik bitleri ve bloğun verisinin değiştirilip değiştirilmediğini kontrol eden kirlenme bitleri kullanılır. Geçerlilik bitinin 0 olması bu bloğun geçersiz olduğunu yani kullanılamayacağını gösterir. Eğer bu bit 1 yapılırsa artık bu blok geçerli hale gelmiş ve kullanılabilir demektir. Kirlenme bitinin 0 olması demek, bu bloğun verisinin temiz (değiştirilmemiş) yani ana bellekten geldiği haliyle aynı olduğu demektir. Eğer bu bit 1 yapılmışsa artık bu blok kirlenmiş demektir. Yani bloğun içindeki veri değiştirildiği anlamına gelmektedir. Önbellek ıska olduğunda aranan blok ana bellekten getirilir. Ana bellekten getirilen blok önbellekte bir blok seçilerek üzerine yazılmak istenir. Üzerine yazılmak istenen bloğun kirlenme biti kontrol edilir. Eğer bloğun kirlenme biti 1 ise yani blok kirli (değiştirilmiş) ise bloğun verisi ana belleğe veri tutarlılığını sağlamak için yazılmak zorundadır. Daha sonra verisi ana belleğe

geri yazılmış blok üzerine ana bellekten ilkin getirilen blok yazılabilir. Bu yönden de biraz karmaşıktır (Intel, 2006).

Doğrudan yazmalı önbellekte her güncelleme işlemi için ana belleğe erişilir. Bu tip tasarımlarda, sonraki seviyeden getirilen veri kirlenmiş bir önbellek bloğuna tekabül ederse, kirlenmiş bloğun sonraki seviyedeki belleğe geri yazılmasına gerek duyulmaz. Ana belleğe gitmeden önbelleğe yazmak ana belleğe erişimi azaltır. Önbelleğe yazma işlemi ana belleğe yazma işleminden daha hızlı gerçekleşmektedir. Örneğin, yığıt gibi döngülü yazmalar gerektiğinde önbelleğe yazmanın hız farkı hissedilecektir. İşlemci ana bellekten çok hızlı çalışmaktadır. Ana belleğe erişim sayısının azalması işlemcinin hızından daha çok faydalanabilmek yani performansı artırmak demektir (Handy, 1998). Aksi halde her yazma isteğinde ana belleğe gidilse (doğrudan yazmalı önbellekte olduğu gibi) işlemci ana bellek hızında çalışacak dolayısıyla işlemcinin yüksek hızından faydalanılmayacaktır. Sonradan yazmalı önbellek bloğunun üzerine yazılıp yazılmayacağının kontrol edilmesi bu yöntemi karmaşıklaştırmaktadır. Doğrudan yazmalı önbellekte ise her adımda ana belleğe zaten yazıldığından bu kontrole gerek yoktur. Önbellek bloğu üzerine yazılacağı zaman ana bellekte zaten temiz kopyası bulunduğundan doğrudan yazılabilir. Doğrudan yazma yönteminin gerçekleştirimi bu yüzden gayet kolaydır. Bellek sistemi hiyerarşisinde trafiği azalttığı için sonradan yazma yöntemi daha verimlidir (Howe, 2006).

2.4. Önbellek Çeşitleri

Modern mikroişlemcilerde amaçlarına göre farklı önbellek çeşitleri bulunur. Sadece boyutu ve fonksiyonelliği değişmez, onun dışında iç organizasyonları da değişebilir. Bu bölüm önbellek çeşitlerini anlatmaktadır (Sun, 2002).

2.4.1. Komut Önbelleği

Komut önbelleği kullanılan komutları depolamak için kullanılır. Komut önbelleği ile gerekli komutun ana bellekten getirilmesi transfer maliyetini azaltır (Sun, 2002). Bazı mimarilerde komut önbelleği dallanma tahminine yardımcı olmak için ekstra bazı bilgiler de içerebilir.

2.4.2. Veri Önbelleği

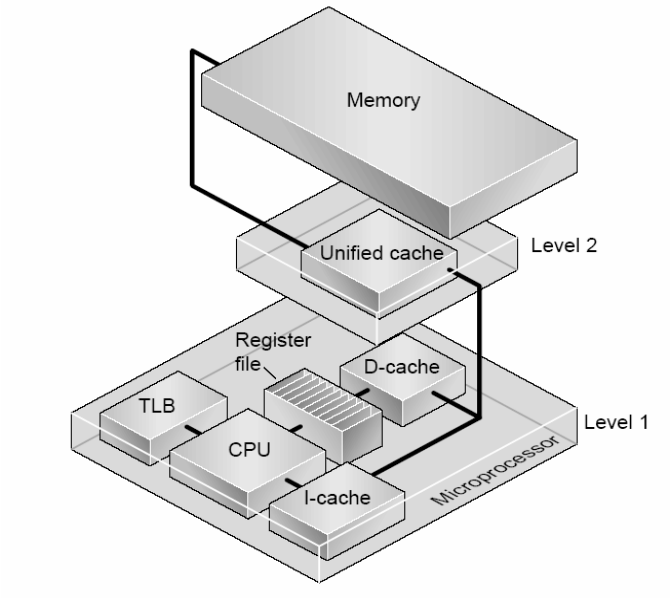
Veri önbelleği, uygulama verilerini içeren hızlı bir tampondur. İşlemci veri üzerinde işlem yapmadan önce veri ana bellekten önbelleğe getirilmek zorundadır. Gerektiğinde önbellekten de yazmaçlara getirilir ve işlenir. İşlemcinin işlediği komutun sonuç verisi yine yazmaca depolanır. Sonra yazmacın içeriği önbelleğe geri yazılır (Sun, 2002).

2.4.3. TLB Önbelleği

Sanal sayfa adresinin geçerli fiziksel adreslere dönüşümü oldukça pahalıdır. TLB bu dönüşümleri depolayan bir önbellektir. TLB içindeki her bir kayıt sanal bellek sayfalarını fiziksel bellek sayfalarına haritalar. CPU ise, sadece adresleri TLB içinde haritalanmış veri ve komutlar üzerinde işlem yapabilir. Eğer bu dönüşüm yok ise sistem onu tekrar oluşturmak zorundadır ki, bu da çok pahalı bir işlemdir.

Genelde veri ve komutlar için farklı iki TLB kullanılmaktadır. Bir tanesi komut içeren sayfalar için i-TLB, diğeri de veri sayfaları için d-TLB.

TLB'ler de göz önünde bulundurulduğunda sistemin bütün bileşenlerin bir araya getirilmiş hali Şekil 2.3'te verilmiştir (Sun, 2002).



Şekil 2. 3 - Genel sistem mimarisi

Şekil 2.3 birleştirilmiş önbelleği ikinci seviyede gösteriyor. Hem komutlar hem de veriler bu önbellekte depolanır. Mikroişlemci dışında gösterildiği için harici önbellek de denir. En alt seviyede ki önbellek daima birleştirilmiş ve mikroişlemci çipinin dışındadır (Sun, 2002).

2.5. Önbellek Organizasyonları

Önbelleğin verimliliğini değerlendirmek için ana belleğe haritalama organizasyonlarına bakılır. Çeşitli önbellek organizasyonları vardır. Bu organizasyonlar önbellek bloklarının nasıl organize edileceğini gösterir. Ana bellek bloğunun önbelleğin neresinde depolanacağının belirlenmesine haritalama denir (Handy, 1998).

2.5.1. Doğrudan Haritalanmış Önbellek (Direct Mapped Cache)

Bu yöntemde ana bellekten getirilen bloğun bellek adresi ile verinin önbellekte nerede depolanacağı belirlenir. Doğrudan haritalamanın gerçekleştirimi kolaydır. Verimli sayılabilecek bir organizasyondur. Önbelleğin hangi bölgesinin kullanılacağı ana bellek adresince belirlendiğinden, kullanımda olan veya ihtiyaç duyulacak bir önbellek bölgesinin yer değiştirilme olasılığı bu yöntemin olumsuz tarafıdır.

Basit bir örnek düşünelim; 4KB'lık önbelleğimiz var. Blok boyutu 32byte olsun. Bu nedenle her veri transferi 32byte veri taşıyacak. Bizim sistemimizde eğer bir kayan nokta tipli değişken 4byte yer tutarsa, her bir satır 8 (32/4=8) değişken içerebilecek. Aşağıdaki döngü (Çizelge 2.1) iki dizi üzerinde toplama işlemi yapmakta ve erişilen verilerin isabet/ıska durumları Çizelge 2.2'de gösterilmektedir (Sun, 2002).

Çizelge 2. 1 - 1024 boyutlu iki dizinin elemanlarının çarpımlarının toplanması

```
float a[1024], b[1024];
for (i=0; i<1024; i++)
Sum += a[i]*b[i];
```

Çizelge 2. 2 - Erişilen verilerin isabet/ıska durumları

```
İlk erişimde a[0] ıska, a[] henüz önbellekte yok.
İlk erişimde b[0] ıska, b[] henüz önbellekte yok.
t=a[0]*b[0]
sum += t
İkinci erişimde a[1] isabet, önceki yükleme ile getirilmişti.
İkinci erişimde b[1] isabet, önceki yükleme ile getirilmişti.
t=a[1]*b[1]
sum += t
```

```
.....
Sekizinci erişimde a[7] isabet, önceki yükleme ile getirilmişti.
Sekizinci erişimde b[7] isabet, önceki yükleme ile getirilmişti.
t=a[7]*b[7]
sum += t
Dokuzuncu erişimde a[8] ıska, henüz önbellekte yok.
Dokuzuncu erişimde b[8] ıska, henüz önbellekte yok.
t=a[8]*b[8]
sum += t
Onuncu erişimde a[9] isabet, önceki yükleme ile getirilmişti.
Onuncu erişimde b[9] isabet, önceki yükleme ile getirilmişti.
t=a[9]*b[9]
sum += t
.....
```

Önbelleğin isabet oranı **7/8**, yani %87.5. Bu en iyi durumdur (Sun, 2002).

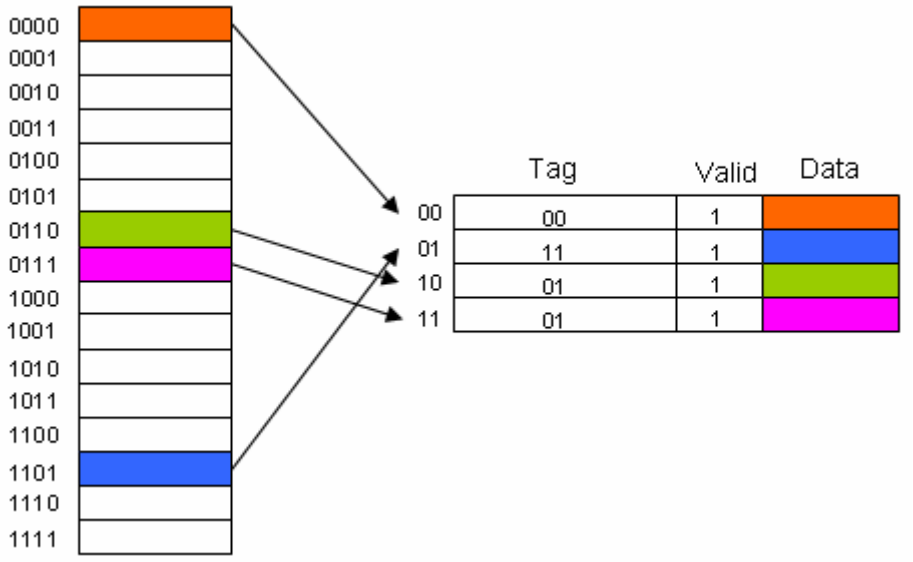
Doğrudan haritalama, önbelleği ana belleğe haritalamanın en basit yoludur. Önbellek ve ana bellek aynı boyutlu satırlara bölünür. Her bir önbellek satırı ona karşılık gelen bir ana bellek satırı tarafından kullanılır. Örneğin önbellek boyutu 16KB, ana bellek boyutu 64MB olsun. Dolayısıyla (64MB/16KB) her bir önbellek satırı 4096 ana bellek adresi tarafından kullanılabilir demektir (Howe, 2006).

Verilen bir adreste ki verinin önbellekte nereye yerleşeceğini belirlemek için orta adres bitlerine bakılır. Eğer bir önbellek bloğu $\mathbf{2^n}$ byte ise sondan **n** tane adres biti blok içi ofsete karşılık gelir. Önbellek $\mathbf{2^m}$ satır barındırıyor ise sonraki **m** tane biti de bloğun gideceği yeri verir. Kalan adres bitleri de "**tag**" kısmını verir (Howe, 2006).

Bu organizasyonda her bloğun gideceği sadece 1 yer olduğundan ıska olma durumunda hangi bloğun çıkarılacağı ile ilgili bir tercih yapmaya gerek yoktur. Bu yöntemin dezavantajı; bir program aynı önbellek bölgesine haritalanmış farklı adreslere erişmeye çalıştığında, her erişimde ıska olmasıdır (Howe, 2006).

Varsayalım ki işlemci 2 adrese ihtiyaç duydu (X,Y). Ama ikisi de aynı önbellek bölgesine haritalanmış. Sırayla önbelleğe çağrıldığını düşünürsek (X,Y,X,Y,X,Y,…) burada bir döngü oluşacak. Önce işlemci X'i bellekten okuyup önbelleğe depolayacak. Sonra Y adresini çağıracak oysa Y'nin haritalandığı yerde X var. Sonra, bellekten Y getirilecek ve ilgili yere depolanacak. Sonra işlemci yine X adresine ihtiyaç duyacak, haritalandığı yerde ise sadece Y var. Bu karmaşıklık defalarca sürer. Bu yüzden bu organizasyon en kötü performans sergiler (Howe, 2006).

Doğrudan haritalama, her ana bellek bölgesinin sadece 1 önbellek bölgesinde depolanacağı yöntemdir. Sadece 1 tane kıyaslayıcı gerektirir. Geliştirilmesi ucuz ve kolaydır (Handy, 1998).



Şekil 2. 4 - Doğrudan Haritalanmış Önbellek (Zilles, 2004)

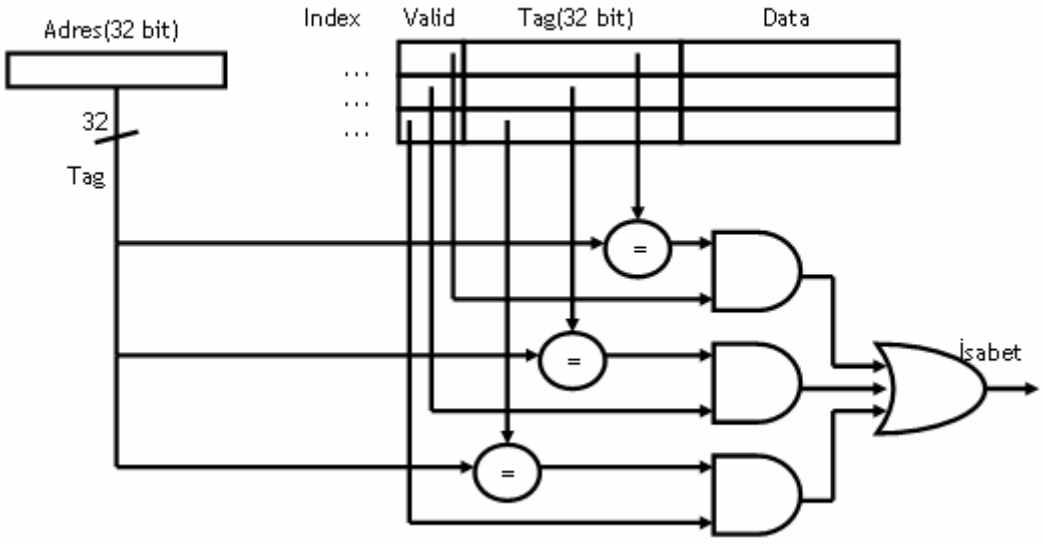
Doğrudan haritalama yönteminde ana bellek satırlarının depolanacağı önbellek satırı belirlidir. Yani her bir ana bellek satırı sadece tek bir önbellek satırında depolanabilir. Doğrudan haritalanmış önbelleğe "1-yollu" küme çağrışımlı önbellek de denilebilir (Şekil 2.4).

Doğrudan haritalama yönteminde ana bellek adresi sadece tek bir önbellek adresi ile kıyaslandığından geliştirimi kolay ve ekonomiktir. Esnek olmadığı için de performansı iyi değildir (Intel, 2006).

2.5.2. Tam Çağrışımlı Önbellek (Fully Associative Cache)

Tam çağrışımlı önbellek organizasyonunda ana bellek ve önbellek eşit boyutlu satırlara bölünür. Herhangi bir ana bellek satırı herhangi bir önbellek satırında depolanabilir (Intel, 2006).

Bu tasarımın dezavantajı, satırların kullanımını izlemek için gerekli ilave mantıksal devrelerin maliyetidir. Önbelleğin boyutu büyüdükçe fiyat arttığından çok geniş veri önbelleklerine uygulamak zordur (Sun, 2002).

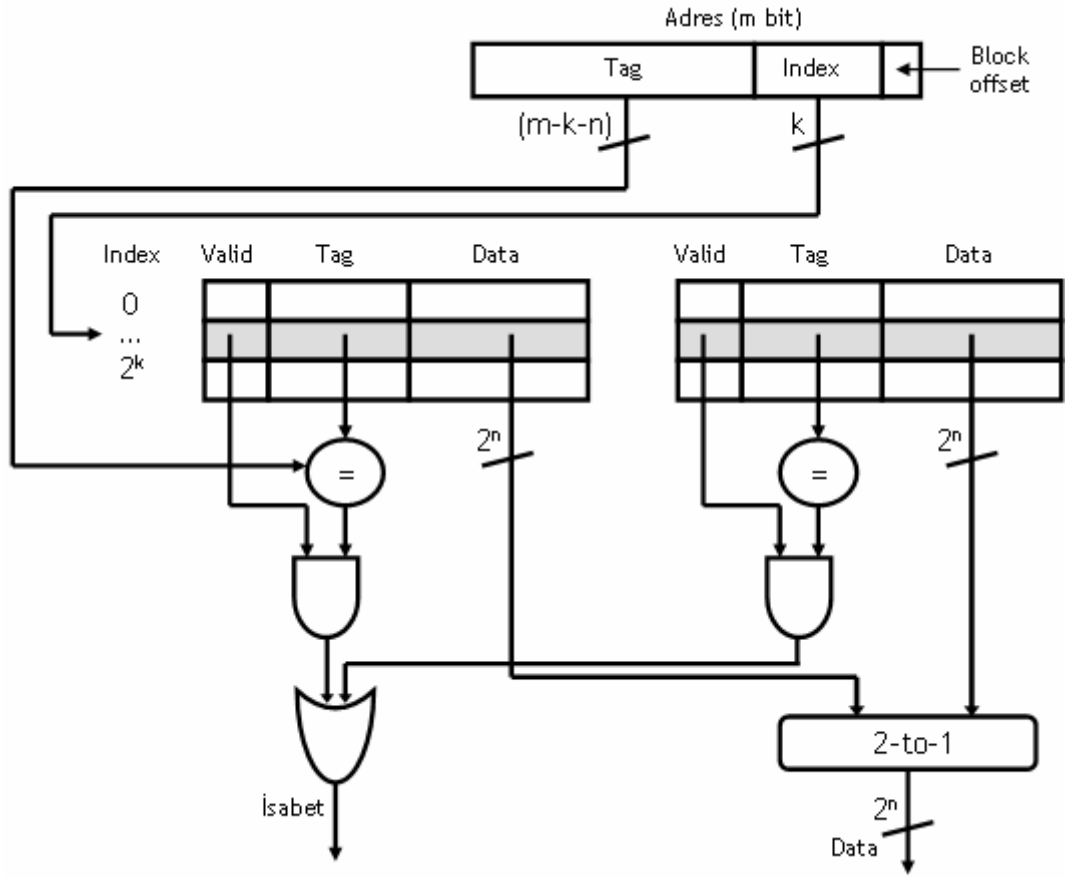


Şekil 2.5 - Tam Çağrışımlı Önbellek (Zilles, 2004)

Her hangi bir ana bellek satırının her hangi bir önbellek satırında depolanabildiği için çok esnektir. Dolayısıyla, en iyi performansa sahiptir. İstenilen verinin önbellekte bulunup bulunmadığına karar verebilmesi amacıyla mevcut adresin bütün önbellek adresleri ile kıyaslanması için çok sayıda karşılaştırıcı gerektirmektedir (Şekil 2.5). Bu da bu organizasyonun gerçekleştirimini karmaşık ve pahalı yapmaktadır. Buna karşın ana belleğin herhangi bir satırının önbelleğin her hangi bir satırında depolanabileceğinden isabet oranı yüksektir. İsabet oranının yüksekliği çok önemli olsa da gerçekleştiriminin karmaşık ve pahalı oluşu pratik açıdan kullanılabilirliğini düşürür. Bu yüzden küçük boyutlu önbelleklerde kullanmak daha mantıklıdır (Intel, 2006). Şekil 2.5'te tam çağrışımlı bir önbellek verilmektedir.

2.5.3. Küme Çağrışımlı Önbellek (Set Associative Cache)

Küme çağrışımlı önbellek tasarımında kümelerin (set) önbelleği oluşturduğu, n tane doğrudan haritalanmış önbellek kullanılır. Adını da içinde kullanılmış doğrudan haritalanmış önbellek sayısına göre alır. Örneğin tasarımında 4 tane doğrudan haritalanmış önbellek kullanıldığında bu önbellek "4-yollu küme çağrışımlı önbellek" diye adlandırılır (Sun, 2002).



Şekil 2. 6 – 2 yollu küme çağrışımlı önbellek (Zilles, 2004)

Küme çağrışımlı önbellek tam çağrışımlı önbellek ile doğrudan haritalanmış önbellek organizasyonlarının birlikte kullanımı ile oluşur. Önbellek, eşit bölümlere bölünmektedir ve bunlara önbellek yolu (cache way) denmektedir.

Küme çağrışımlı önbellek yolu (way) sayısı kadar karşılaştırıcı gerektirdiğinden gerçekleştirimi kolay ve ekonomiktir. Örneğin, 4 yollu tam çağrışımlı önbellek için 4 tane karşılaştırıcı gerekmektedir (Intel, 2006). Şekil 2.6'da 2 yollu küme çağrışımlı bir önbellek verilmektedir.

Bir ana bellek bölgesinin depolanabileceği çağrışım sayısı kadar önbellek bölgesi bulunmaktadır. Çağrışım sayısı kadar kıyaslayıcı gerektirdiği için değişik tasarımlarda gerçekleştirilebilir. En kullanışlı yöntemdir (Handy, 1998).

Tasarımcılar, doğrudan haritalanmış ve küme çağrışımlı önbellek tasarımını daha fazla kullanmaktadırlar. L1 önbellekleri için küme çağrışımlı önbellek, L2 ve diğer önbellek seviyeleri için ise genellikle doğrudan haritalanmış önbellek kullanılmaktadır (Howe,2006).

2.6. Yer Değiştirme Algoritmaları

Önbellek tamamen dolduğunda ve ana bellekten önbelleğe yazmak için yeni bir blok getirildiğinde bu blok, var olan bir önbellek bloğuyla yer değiştirilmeli ve oraya yazılmalıdır. Getirilen bloğun hangi önbellek bloğu ile yer değiştireceği yer değiştirme algoritmaları ile seçilir (Handy, 1998).

2.6.1. Rasgele Algoritması (Random)

Bu algoritma belirli bir kurala göre çalışmaz, yer değiştirme için önbellek içinde rasgele bir blok seçer.

2.6.2. İlk Giren İlk Çıkar Algoritması (First In Firs Out-FIFO)

FIFO algoritmasında her yeni bloğun bir zaman mührü vardır. Yer değiştirme için blok seçileceği zaman, zaman mührü en eski olan yani önbelleğe daha önce getirilmiş blok seçilir. FIFO algoritmasında önbellekte bulunan her bir sözcük için ekstra bitler kullanılır.

2.6.3. Son Zamanlarda En Az Kullanılan Algoritması (Least Recently Used-LRU)

Bazı karşılaştırıcılar önbellek içinde erişimleri gözler. Onları zamana ve erişim sayısına göre sıralar. Son zamanlarda en az erişilmiş olan bloğu üzerine yeni gelen bloğun yazılması için seçer (Handy, 1998).

BÖLÜM 3

SOFT ERRORLAR VE ÖNBELLEKLERE ETKİLERİ

3.1. Soft Error Nedir?

Pakajlama materyallerinden yayılan alfa tanecikleri, kozmik ışınlardan yayılan nötron tanecikleri mantıksal aygıta (transistör) çarparak aygıtın şarjını artırabilir. Aygıt üzerinde biriken şarj işlemcilerin mantıksal yapılarında ve SRAM hücrelerinde tutulan bilgilerin değişmesine sebep olabilir. Bu yapıların kapasitans ve besleme gerilimleri çok düşük ve yüksek frekansta çalıştıklarından, biriken enerji bu devrelerin çıkışlarını değiştirmeye yetebilir. Bunun sonucunda yanlış sonuç üretilmesine, uygulamanın veya sistemin çökmesine sebep olabilinmektedir (Ziegler, 1996). DRAM hücrelerinin kapasitans ve besleme gerilimleri daha yüksek olduğundan, soft errorlar DRAM için her hangi bir güvenirlilik tehdidi oluşturmamaktadır. Sızıntı (leakage) enerjisini azaltmaya yönelik mekanizmalar soft error problemini daha ciddi boyutlara sokmuştur (Degalahal ve diğ., 2003).

Modern mikroişlemciler, işlemciyle bellek arasına aradaki hız farkını dengelemek için köprü vazifesinde bir önbellek koymaktadır. Önbellekler, çip üzerinde en geniş yer kapladıklarından özellikle soft errorlara karşı hassastır. Dış parçacıkların çarpmasıyla oluşan enerji SRAM hücrelerinin depoladığı bitleri ters çevirebilir. Bitlerin değerlerinin ters çevrilmesiyle oluşan bozukluk işlemciye veya diğer seviye belleklere kolayca yayılabilmektedir. Bunun sonucunda hatalı hesaplamaya ve sistemin çökmesine sebep olabilmektedir. Sonuç olarak, önbellekler yapılan işlemlerin doğruluğunu sağlamak için soft errorlara karşı korunmalıdır (Zhang, 2005).

Harici ışınımlardan kaynaklanan geçici hatalar mikroişlemci tasarımında önemle dikkate alınması gerekir hale gelmiştir. Önceki çalışmalar doğrulanamayan hataların, önbellek hata oranını yükselttiğini göstermektedir (Bauman, 2002). Boyutların küçülmesi, devrelerin besleme gerilimlerinin azalması, frekansının

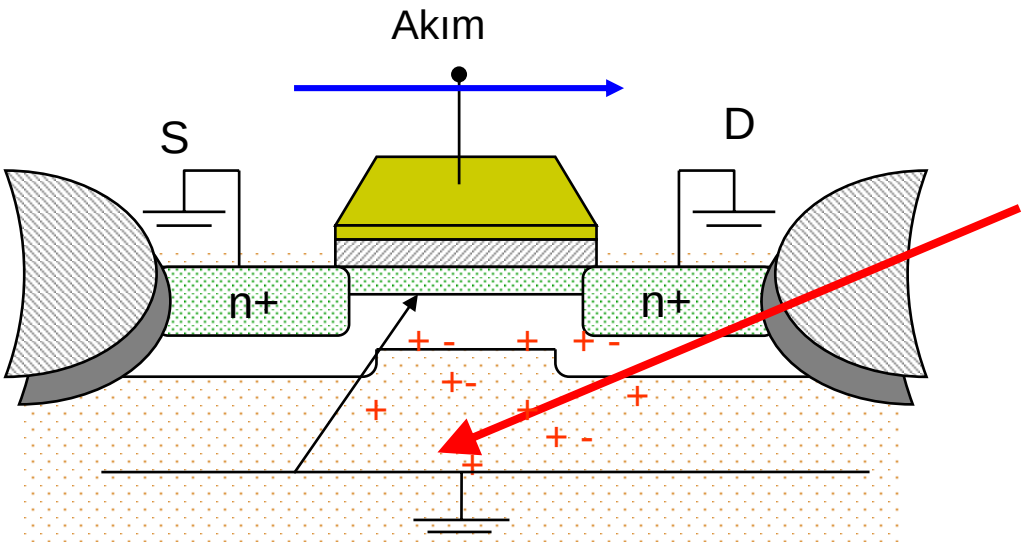
yükselmesi gelecek mikroişlemcilerin soft errora karşı bozulabilirliğini artırmaktadır (Zhang, 2005). Sonuç olarak, soft error bilinçli sistem tasarımı, güvenirlilik için gerekli hale gelmiştir (Zhang ve diğ., 2003).

Teknoloji geliştikçe devrelerin enerji tüketimini azaltmak ve CMOS aygıtlarının güvenirliliğini geliştirmek ciddi bir olay haline gelmiştir. Diğer taraftan sağlanan enerjinin düşürülmesi ve kapasitansın azaltılması, soft error oranını artırmaktadır (Degalahal, 2005).

Bütün donanım devreleri soft errorlardan etkilenmesine rağmen özellikle önbellekler daha duyarlıdır. Bugünün mikroişlemci çiplerinin %60'ını önbellekler oluşturmakta bu da önbellekleri soft errorlara karşı daha bozulabilir yapmaktadır. Önbelleklerdeki soft errorlardan kaynaklanan bir hata çalışma sırasında yanlış bir sonuç üretilmesine sebep olur. Önbellek işlemciye çok yakın olduğundan önbellek içinde oluşan bir hata işlemciye ve yazmaçlara kolayca yayılabilmektedir (Zhang ve diğ., 2003).

Günümüzde kullanılmakta olan popüler hata kontrol yöntemleri; 8 bit veri için 1 bit ekleyen eşlik biti kontrolü (parite) ve SEC-DED (Single Error Correct Double Error Detect). Geçici hataları sezmek ve bu hataları düzeltmek için bu yöntemler kullanıldığında temel üç sorun ile karşılaşılmaktadır. Birincisi, fazladan alan işgali. Eşlik biti kontrol yöntemi 8 bit veri için 1 bit veri eklediğinden alandan %12,5 kayıp verir. Aynı şekilde SEC-DED ise 64 bit veri için 8 bit ilave eder. İkincisi ve daha önemlisi önbellek gecikmesi üçüncüsü ise veri bütünlüğünü doğrulamak için fazladan enerji tüketilmesidir. SEC-DED mekanizması eşlik biti ile kıyaslandığında daha fazla enerji tüketir (Zhang ve diğ., 2003).

Geçici hatalar nadiren olduklarından, hata oluşmadığı genel durumlarda performans ve enerji maliyetini artırmamak çok önemlidir. Hata oluştuğunda eşlik biti kullanılarak tek bitlik hatalar sezilebilmekte ve hatalı bloğun kopyası kullanılarak da hatanın düzeltilmesi mümkün olmaktadır (Zhang ve diğ., 2003).



Şekil 3. 1 - Aygıta çarpan tanecik aygıt boyunca bir elektron boşluğu oluşturur

Enerji yüklü parçacık mantıksal bir aygıta (SRAM, kapı,…) çarptığında, parçacığın kinetik enerjisi aygıt boyunca bir elektron boşluğu oluşturur. Bu olay bellek elemanlarında bit değişimine sebep olur. Hesaplama devrelerinde ise çıkışın geçici olarak değiştirilmesine sebep olmaktadır (Degalahal, 2005). Şekil 3.1'de aygıta çarpan taneciğin aygıt boyunca bir elektron boşluğu oluşturması verilmektedir.

Modern mikroişlemci dizaynında öteden beri performans konusuna dikkat edilmesi zorunludur. Moore Kanunu'na göre çip başına düşen transistör sayısında üssel bir artış olduğuna, yarı iletken aygıtların performans ve fonksiyonelliklerinde ki önemli artışa şahit oluyoruz. Ama şu anda aralarında, soft errorlara sebep olan radyoaktif ışınımların da bulunduğu pek çok dış etken bu gelişime olumsuz yönde etki etmektedir. Radyoaktif ışınımlar bunların en başında gelmektedir.

Mikroişlemci, üreticisinden donanımsal olarak hatalı veya noksan çıkmasa bile çevresel etkenler donanım üzerinde geçici veya kalıcı bozukluklara sebep olabilirler. Geçici hatalar veya tek bitlik bozulmalar (SEU-Single Event Upset) da denilen ve radyoaktif ışınımlar sebebiyle oluşan bu hatalar soft errorlar diye adlandırılırlar. Ana nedenleri kozmik ışınlardan gelen nötron tanecikleridir (Karnik ve diğ., 2001). Diğer sebeplerden en önemlisi ise pakajlama materyallerinde ki radyoaktif kaynaklardan

meydana gelen alfa tanecikleridir. Bu enerji yüklü tanecikler yarı iletken boyunca, transistörün enerji biriktirmesine sebep olan elektron boşluğu oluşturmaktadır. Yeterli miktarda biriken enerji SRAM hücresi veya mandal gibi mantıksal aygıtın durumunu 0'dan 1'e veya 1'den 0'a değiştirebilir. Bu nedenle veride geçici, tekrar etmeyen, donanım bileşenleriyle ve üretim hataları ile alakası olmayan bozukluklara sebep olurlar. Bu tip bozulmalara geçici hata denir. Aygıtta kalıcı bir hataya veya bozukluğa sebep olmaz.

Soft error oranını etkileyen farklı devre parametreleri vardır. Bunlar, depolanan enerjinin ve bozulabilir alanların miktarıdır. Devre boyutları küçüldükçe, her bir aygıt başına düşen enerji azalmakta ve parçacık çarpmasıyla soft errora maruz kalabilmektedir. Diğer taraftan, aygıt alanının küçülmesi ile parçacık çarpabilme olasılığı düşmektedir. Bu iki husus ters etkilere sebep olmakta ve birbirini dengelemektedir. Dolayısıyla her bir transistör için bireysel soft error oranının yakın gelecekte çok az bir azalma veya artmayla birlikte sabit kalması beklenmektedir (Karnik ve diğ., 2001; Seifert ve diğ., 2002). Her bir çip başına düşen soft error oranı ise artan transistör sayısından dolayı, iyi bir hata sezme/düzeltme mekanizması kullanılmadıkça transistör sayısı ile orantılı olarak sürekli artmaktadır. Bu nedenle, Moore Kanunu'nun öngördüğü transistör sayısında ve performansında ki üssel artış korumasız çipler için hata oranının ve bunun maliyetinin üssel olarak artacağını göstermektedir.

Soft errorların bir kısmı tespit edilemeyen olarak sınıflandırılabilirler. Geliştirilmiş teknikler kullanılarak, uzay endüstrisi ve nükleer sistemler gibi güvenirlilikleri hayati önem taşıyan sistemlerin soft errorlara karşı korunması çok önemlidir. Benzer teknikleri barındırmak ticari işlemciler için pratik ve uygun olmasa da, maliyet ve performans feda edilemediğinden, soft errorlar ticari işlemci üreticisi tarafından dikkate alınır hale gelmiştir.

CMOS teknolojisinin tırmanışı, yarı iletken aygıtların performansını muazzam bir şekilde geliştirmiştir. 100nm'lere çıkıldığında bu kazançlar güvenirlilik tehdidiyle karşı karşıya gelmektedir. Özellikle, soft errorlar işlemci tasarımında yeni bir tehlike

olarak karşımıza çıkmaktadır. Tanecik çarpması, bir depolama hücresinde barınan veriyi veya mantıksal bir devrenin hesapladığı değeri değiştirebilir (Li ve diğ., 2005).

Gelişen teknolojiye paralel olarak soft error oranlarının nasıl değişeceği konusunda çeşitli öngörü çalışmaları yapılmıştır (Karnik ve diğ., 2004; Nguyen ve Yagil, 2003; Shivakumar ve diğ., 2002).

Son zamanlara kadar, çalışmaların büyük çoğunluğu aygıt ve devre seviyesine odaklandı. Daha sonraları, mimari seviyeli çalışmalar yapıldı (Kim ve Somani, 2002; Mukherjee ve diğ., 2003; Czeck ve Siewiorek, 1990; Wang ve diğ., 2004; Weaver ve diğ., 2004).

Donanım temelli güvenirliliği geliştirme teknikleri üç genel kategoriye ayrılabilir: Süreç (process) teknoloji çözümleri, devre temelli çözümler ve mimari seviyeli çözümler (Mukherjee ve diğ., 2005). Süreç teknolojisinde, aygıtı soft errora karşı korumak için silikon kaplayıcı (SOI-Silicon Insulator) kullanılır. IBM raporlarında SOI teknolojisi kullanarak SRAM aygıtlarının SER oranlarından 5 kat azaltılmasının mümkün olduğunu belirtmiştir (Cannon ve diğ., 2004). Devre temelli çözümler aygıtın kapasitans, eşik besleme enerjisi gibi parametrelerini değiştirerek ışınımlara dayanıklı yapılar oluşturmayı önermektedir (Calin ve diğ., 1996). Mimari seviyeli çözümler ise, parite, ECC, PI bit, NMR, RC gibi bütünlük kontrolleri ile güvenirliliği geliştirir.

Aygıt seviyesinde oluşan hataların çoğu mimari seviyesinde maskelenebileceğinden, son zamanlarda mimari seviyesinde soft errorların bertaraf edilmesi yönünde oldukça çok çalışmalar var. Bizim önerdiğimiz yöntemde mimari seviyesindedir.

Örneğin (Whang ve diğ., 2004) yaptıkları çalışmada, ham hataların %85'inin mikromimari seviyede maskelenebildiğini gösterdiler. Bu kadar yüksek maskeleme oranının sebebi, modern işlemcilerde düşük kaynak kullanımı ile ilgilidir. Kaynağın çoğalması sadece performansı etkiler, ama gerçek program çıkışını etkilemez.

3.2. Soft Error Sebepleri

3.2.1- Pakajlama Bozulmaları (Package Decay)

Soft error kavramı 1970'lerde dinamik RAM kavramı ile duyulmaya başladı. Önceleri çiplerin pakajlama materyalleri küçük miktarda radyoaktif element içeriyordu. Pakajlama materyallerinde ki çok küçük bozulmalar bile çipi az da olsa soft errorlara maruz bırakıyordu. Çip üreticileri arada sırada karşılaşılan bu probleme katlanıyorlardı. Pakaj radyoaktif bozulmaları genellikle alfa parçacıklarının emiliminden kaynaklanan soft errorlara sebep olmaktadır. Pozitif yüklü alfa parçacıkları yarı iletken boyunca hareket eder ve oradaki elektron düzenini bozmak suretiyle soft errorlara sebep olabilmektedir.

3.2.2- Kritik Şarj (Critical Charge)

Devrelerde soft error olup olmayacağı gelen parçacığın enerjisine, etkinin geometrisine ve mantık devresinin tasarımına bağlıdır. Devrelerin kapasitesi ve eşik enerjileri ne kadar yüksek olursa soft errorlara o denli daha az maruz kalırlar. Bu kapasite ve enerji kombinasyonları kritik şarj (critical charge) parametresi ile tanımlanmaktadır. Qcrit, mantık devresinin durumunu değiştirebilecek gerekli minimum enerjiyi tarif eder. Qcrit değerinin yüksek olması soft error olasılığının çok düşmesi demektir. Maalesef yüksek değerli Qcrit demek devrenin yavaş çalışması anlamına gelmektedir. Besleme gerilimlerinin düşürülmesi dolayısıyla Qcrit değerinin düşmesi bir çok açıdan istenilen bir şeydir (Devrenin düşük enerji tüketimi açısından).

3.2.3- Kozmik Işınlar (Cosmic Rays)

Ziegler (1996) IBM'de tamamladığı bir çalışmasında kozmik ışınların soft errora neden olduğunu gösterdi. Gerçekten, modern cihazlarda kozmik ışınlar soft errorlara neden olan en baskın sebeptir. Kozmik ışınların içinde çok farklı

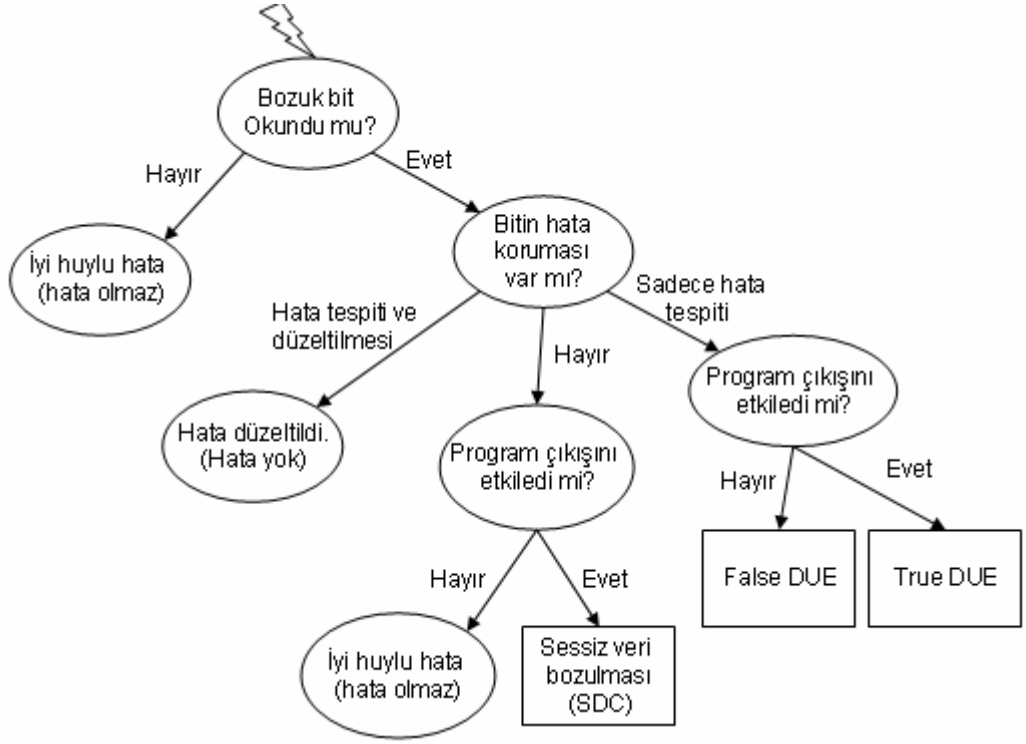
parçacıklar bulunsa da, soft erorların ana sebebi nötron parçacıklarıdır. Ama çip içindeki atom çekirdeği tarafından yakalanan bir nötron taneciği, bozulmalarla meydana gelen, soft errora sebep olan ve kararsız bir izotop olan alfa parçacıklarını üretir. Kozmik ışınlar uzaydan gelmektedir. Deniz seviyesine gelene kadar gökyüzü tabakasında pek çok kozmik ışın süzülür. Yani deniz seviyesinde kozmik ışınlardan kaynaklanan soft errorlara maruz kalma olasılığı düşüktür. Yüksek dağlarda veya uçakta işletilen bir program ile deniz seviyesinde işletilen aynı program aynı sonucu vermeyebilir. Bu da yükseklerde çiplerin kozmik ışınlara maruz kalma olasılıklarının daha yüksek olmasından kaynaklanmaktadır.

3.2.4- Diğer sebepler

Soft errorlar çevredeki rasgele gürültülerden de meydana gelebilmektedir.

3.3. Soft Error Türleri

Şekil 3.2'de işlemcide hataya maruz kalan bir bitin yol açabileceği tüm sonuçlar özetlenmiştir (Mukherjee ve diğ., 2005). Şekilden de görüldüğü gibi, bazı bozulmalar program çıkışında kullanıcı gözüyle görülmeyen hatalar üretebilmektedir. Hata koruması olmayan ve program çıkışını etkileyen bir bitin içindeki hata sessiz veri bozulmasına (SDC-Silent Data Corruption) sebep olur. Veri bozulmasına sebep olan ancak sezilemeyen bu hatalar en zararlı hata şeklidir. En azından bu hataları sezebilmek için parite gibi hata sezme mekanizmaları bulundurulabilir. Parite ile sezilebilen ancak düzeltilemeyen bozulmalar ise sezilebilen ama düzeltilemeyen hatalara (Detected Unrecoverable Error-DUE) sebep olur. Böyle durumlarda herhangi bir hata veya bozukluk fark edildiğinde bu bozukluğun program çıkışına etkisini önlemek için çalışan uygulama durdurulur. DUE hataları doğru (true) DUE ve yanlış (false) DUE olarak ikiye ayrılabilir. Doğru DUE hataları (sezilebilen ama düzeltilemeyen) program çıkışını etkileyen, yanlış DUE hataları ise program çıkışını etkilemeyen iyi huylu hatalardır.



Şekil 3. 2 - Mikroişlemcide ki bozulan bir bitin muhtemel sonuçlarının tasnifi (Mukherjee ve diğ., 2005)

3.4. Soft Errorlar Nasıl Ölçülür?

Soft Errorlar nadir olaylar olmasına rağmen, Soft Error Oranı (SER-Soft Error Rate) performans, yapı malzemesi, düşük güç ve düşük maliyetten sonra bilgisayar tasarımında en önemli kriter haline gelmiştir. SER'i değerlendirmek için en genel ölçüm, FIT (Hata Zamanı-Failure in Time)'tir. O da bir milyar saatte ki hata sayısına karşılık gelmektedir (1 FIT = 1 milyar saatte 1 hata demektir.).

Ham SER, SRAM'de ki transistör sayısı ile doğru orantılıdır (Hareland ve diğ., 2001; Karnik ve diğ., 2001). Daha önceki bazı çalışmalar (Karnik ve diğ., 2004; Nguyen ve diğ., 2003) SRAM ve kapı gibi farklı tip devrelerin SER oranları ile ilgilidir.

İşlemci üreticileri genelde kendi ürünlerinin SER hedefini koymaktadır. Mesela, IBM Power4 işlemcisinde SDC (Sessiz Veri Bozulması) için 114 FIT

hedeflemektedir. DUE (Sezilebilen Ama Düzeltilemeyen Hatalar) için 4566 sistem çökmesi (system kill) FIT, yine DUE için 11415 uygulama çökmesi (process kill) FIT hedeflemektedir (Mukherjee ve diğ., 2005). Üreticiler kendi SER hedeflerini karşılaması için performans ve maliyet açısından en uygun güvenirlilik mekanizmalarını seçmektedirler.

AVF, sonradan bahsedileceği gibi işlemci içinde ki bir yapıda oluşan bir bozulmanın, kendisini program çıktısında yüzde kaç ihtimalle hata olarak göstereceğini ifade eder.

MTTF (Mean Time to Failure-Hataya Olan Ortalama Zaman) sistem güvenirliliğini ölçmek için kullanılır. O da şu şekilde açıklanır:

$$MTTF = \frac{1}{(HamHata\ Orani) * (AVF)}$$

Ama, bu ölçüm performans ve güvenirlilik arasındaki getiri götürü ilişkilerini yakalama da başarısızdır. Örneğin, iki sistemden birincisi ikincisinden iki kat daha hızlı ama ikincisinin yarısı kadar güvenirliğe sahip, o zaman her iki sistemde de bir uygulamanın çalışması sırasında bozulma olma olasılığı aynıdır. Bu şekilde değerlendirildiğinde bu iki sistemde eşit güvenirliğe sahip görünmektedir. Ama, doğru olmamasına rağmen MTTF değeri, yavaş olan ikinci sistemin daha güvenilir olduğuna işaret etmektedir. Bunu açıklamak için, (Weaver ve diğ., 2004), MITF (Mean Instruction to Failure) denilen bir ölçüm değeri tanımladılar. Bu ölçüm performans ile güvenirlilik arasındaki getiri götürüyü verimli bir şekilde yakalamaktadır.

Adından da anlaşılacağı gibi, MITF iki hata arasında tamamlanan ortalama komut sayısını verir. (Weaver ve diğ., 2004), MITF değerini şöyle tanımlıyorlar:

$$MITF = \frac{Tamamlanan\ Komut\ Sayisi}{Karsilasilan\ Hata\ Sayisi}$$

$$= \frac{Tamamlanan\ Komut\ Sayisi}{\frac{Toplam\ Calisma\ Zamani\ *\ Frekans}{Frekans * MTTF}}$$

$$= IPC * Frekans * MTTF$$

$$= \frac{(IPC * Frekans)}{(Ham\ Hata\ Orani * AVF)}$$

$$= (\frac{Frekans}{Ham\ Hata\ Orani}) * (\frac{IPC}{AVF})$$

Varsayalım ki, ham hata oranı ve frekans sabit, MITF değeri (IPC/AVF) değeri ile doğru orantılıdır. Bu nedenle, hem AVF değerini azaltan hem de IPC değerini artıran herhangi bir yöntem MITF değerini geliştirebilir.

MTTF ölçüm değeri sistem güvenirliğini ölçmek için kullanılır. MTBF (Mean time between failures-Hatalar arası ortalama zaman) ölçümü hatalar arasında ki ortalama zamanı ölçer. Bir bileşenin MTBF ölçüm değeri, onun hatalarının arasındaki ortalama zamanı olarak düşünülür. FIT ise MTBF ile ters orantılıdır. Bir FIT bir milyar saatte bir hatayı ifade eder. Sıfır hata oranı ise sonsuz MTBF ve sıfır FIT değerini ifade eder. Çipin bütün FIT oranı, çip üzerinde ki yapıların FIT oranlarının toplamı olarak hesaplanabilir. Anahtar ve SRAM hücrelerinin FIT oranı sayıları deniz seviyesinde 0.001 FIT/bit ile 0.01 FIT/bit arasında değişiklik göstermektedir ve birkaç nesil teknoloji sonrasında da aynı kalması planlanmaktadır (Seifert ve diğ., 2002; Mukherjee ve diğ., 2003; Karnik ve diğ., 2001; Normand, 1996).

3.4.1- SDC Oranının Hesaplanması

Üreticiler genellikle işlemcileri için bir hata oranı hedefi belirlerler. Mesela, IBM SDC hataları için 1000 yıllık MTBF hedeflemektedir. Yani, işlemcisinde olabilecek iki SDC hatasının arasındaki ortalama zamanın 1000 yıl olmasını hedeflemektedir. Ham hata oranları da SDC gibi FIT ile açıklanır. Çok bitlik

bozulmalar yapı içinde çoklu bitleri etkileyen tekil veya çoğul parçacık çarpmasıyla oluşur.

İşlemcinin SDC oranı bütün aygıtlarının SDC oranına katkılarının toplamıdır. Bir aygıtın SDC oranı ise, onun ham hata oranının ve SDC AVF değerine bağlıdır. Aygıtın sahip olduğu herhangi bir hata sezme ve düzeltme tekniği sayesinde SDC AVF değeri ve dolayısıyla SDC oranı sıfır olacaktır. Korunmayan aygıtların SDC AVF değeri onların yapılarına göre değişir. Mesela, program sayacı %100 AVF değerine sahiptir. Çünkü program sayacındaki herhangi bir bozulma kesinlikle yanlış komutun çalıştırılmasına sebep olacak dolayısıyla programın çalışma yolunu değiştirecektir. Oysa, bir dallanma tahmincisi düşünürseniz, tahmin yapılarında bir bozulma olsa bile, sadece dallanma tahmini yanlış olacaktır. Programın çıkışı etkilenmeyecek ve bu nedenle, dallanma tahmin yapısının AVF değeri %0 olacaktır. Komut kuyruğu ve yazmaç gibi çip üzerindeki diğer yapıların AVF değeri doğrudan sezgisel değildir. Bu karmaşık yapıların AVF değeri %0 ile %100 arasında değişir. Daha önce bahsedildiği gibi, bir yapının bozulma oranı onun ham hata oranı ve AVF değerine bağlıdır. Ve bütün işlemcinin bozulma oranı her bir aygıtın bu orana katkıları toplamına eşittir.

3.4.2- Mimari Bozulabilirlik Faktörü (Architectural Vulnerability Factor)

Mikromimari yapılarda ki bütün hatalar programın son çıkışını etkilemeyebilir. Bu nedenle, ham bozulma oranlarına dayanarak etken hata oranlarını karşılaştıramayız. Böyle, ham hata oranlarına dayanarak yapılan kötümser tahmin tasarımda işlemcinin bozulma yönetimi özelliklerini etkileyebilir. Her bir bitinin etkin hata oranı, hem aygıtın ham hata oranına hem de aygıtın bozulabilirlik faktörüne bağlıdır.

Bir yapının AVF'si (mimari bozulabilirlik faktörü) veya soft error duyarlılığı işlemci içindeki bir yapıda ki bozulmanın program çıktısında hangi oranda hataya yol açtığını ifade eder (Mukherjee ve diğ., 2003). Bir yapının AVF değerini tahmin

etmek için, yapı içindeki hangi bitlerin programın çıkışına etki edip hangilerinin etki etmediğini belirlememiz gerekmektedir.

Bu amaçla, Mukherjee ve diğ. (2003) bir yapı içindeki bütün bitleri iki sınıfta grupladılar: Bu bitler de ACE bitleri ve un-ACE bitleridir. Yapı içinde ki ACE bitleri mimari doğru çalışabilirlik bitleridir (*Architecturally Correct Execution*). Yani programın doğru çalışabilmesi için bu bitlerin mutlaka doğru olması gerekmektedir. Diğer taraftan, un-ACE bitler ise doğru çalışabilirlik için gerekli olmayan, yani programın doğru çalışması için doğru olması gerekmeyen bitlerdir. Eğer bu bitlerde bir değişme olursa, program çıkışını kesinlikle etkilemeyecektir. Örneğin, tahmine dayalı olarak yüklenen yanlış yol komutlarındaki bitler ise un-ACE olarak sınıflandırılabilir. Yani bunlarda ki bir bozulma program çıkışına herhangi bir etkide bulunmayacaktır. Yine önbellekte erişilmeyen bloklara ait bitlerde oluşabilecek hatalar da program çıktısına etki etmezler. Bir yapının AVF değeri herhangi bir zamanda şöyle açıklanabilir:

$$AVF = \frac{(Yapi\ Icindeki\ ACE\ Bitlerin\ Sayisi)}{Yapi\ Icindeki\ Toplam\ Bitlerin\ Sayisi}$$

Bellek elemanlarının ham soft error oranı şu eşitlikle açıklanabilir (Hazucha ve Svensson, 2000):

$$SER \propto N_{flux} * CS * e^{(\frac{Qcrit}{Q_s})}$$

Nflux: Nötron akış yoğunluğu
CS: Parçacık akışının etki ettiği alan
Qs: Aygıtın verimli çalışabileceği enerji
Qcritical: Aygıtın depoladığı verinin değişmesi için kritik enerji

Soft error oranı aygıtın alanı ile de orantılıdır. Daha küçük aygıtların daha az alanı olacağından hatalarla bozulma olasılıkları azalmaktadır. Ama yeni nesil teknolojilerde kritik enerjinin düşmesi ve yoğunluğun artması soft error oranını

artırmaktadır (Hazucha ve Svensson, 2000; Sefiert ve diğ., 2001; Shivakumar ve diğ., 2002).

Aygıtın tekil durum değişimlerinden kaynaklanan hata oranı hem karşılaştığı tanecik akış yoğunluğuna hem de devre özelliklerine bağlıdır. Tanecik yoğunluğu çevreye bağlıdır. Örneğin, 1,5 km yükseklikte (Denver'ın yüksekliği) kozmik ışınlardan kaynaklanan nötron akış yoğunluğu deniz seviyesindekinden 3 ila 5 kat daha yüksektir. Hata oranını etkileyen aygıt devre parametreleri arasında depolanan şarj miktarı, bozulabilir kesit alanı ve toplanan şarj verimliliği yer alır (Shivakumar ve diğ., 2002).

3.4.3- Önbellek Bozulabilirlik Faktörü (Cache Vulnerability Factor)

Önbellekler soft errorlara karşı hassas olmasına rağmen bütün önbellek soft errorları görülebilir sistem bozukluklarına sebep olmayabilir. Sonuç olarak, diğer sistem bileşenlerini etkileyen ve hatalı sonuçlar veren önbelleklerin soft errorlara yakalanma olasılıklarını (hesaplamak) ölçmek gerekmektedir. Son zamanlardaki bir çalışmada programın çıktısında görünür hata oluşturan işlemci yapılarındaki bozulmaları ifade eden mimari bozulabilirlik faktörü (AVF-Architectural Vulnerability Factor) önerildi (Mukherjee ve diğ., 2003). AVF çeşitli donanım bileşenleri için soft error oranının doğru tahmin edilebilmesini sağlamaktadır. Aslında, AVF'nin başlıca ilgilendiği konu veri yoludur. Komutlar ile ilişkili olan mimari ve mikromimari un-ACE bitlerin belirlenmesiyle hesaplanmaktadır (Mukherjee ve diğ., 2003). L1 veri önbelleği için bir hata oluşumu ve yayılımı modeli Kadayıf ve Kandemir tarafından önerilmiştir (Kadayıf ve Kandemir, 2007). Bu çalışmaya göre, bir veride oluşacak soft error olasılığı o verinin boyu ve önbellekte kaldığı süre ile doğru orantılıdır. Aynı çalışmada, soft errorların önbellekteki etkisini azaltmak için çeşitli mekanizmalardan bahsedilmiş ve bu mekanizmaların verimlilikleri önerilen model doğrultusunda incelenmiştir.

3.5. Soft Error Etkileri

Kozmik ışınlardan ve alfa taneciklerinden kaynaklanan soft errorlar endüstriyi de etkilemektedir. Beşinci nesil SPARC64 işlemcilerinde, Fujitsu parite kontrollü hata sezme ile toplam 200k boyutlu anahtarın kozmik ışın çarpmalarına %80 oranında karşı koymayı planlıyor. Çarpma ve bölme birimleri kalan diğer kontrol ve parite tahmin devreleri ile planlanıyor (Ando ve diğ., 2003).

Normand (1996) birçok büyük bilgisayar sisteminin hata kayıtlarını inceleyerek çok sayıda kozmik ışın darbelerinden kaynaklanan olaylar raporlamıştır.

Sun Microsystems 2000 yılında, UltraSPARC II işlemcileri üzerinde ki korumasız L2 önbelleğine çarpan kozmik ışınlar, aralarında American Online, e-Bay, Verisign ve diğer bir çok şirketin de bulunduğu bazı müşterilerin kurumsal sunucularının aniden çökmesine sebep oldu (Baumann, 2002). Bundan ötürü Sun firması çok sayıda müşterisini IBM firmasına kaptırdığını belirtmiştir. Sun UltraSPARC T1 işlemcilerinin tamsayı ve kayan noktalı sayı yazmaçları ECC ile korunmaktadır, gelişmiş seviye korumaya sadece büyük boy sistem işlemcilerde karşılaşılmaktadır (Bryg ve Alabado, 2005).

3.6. Mimari Temelli Güvenirliliği Artırma Teknikleri

Soft errorlara karşı önbellek güvenirliliğini artırmak için pek çok yaklaşım mevcuttur. Önerilen teknikler genellikle hem performansı olumsuz etkilemekte hem de alan, enerji ve tasarım zamanını artırmaktadır. Bu yüzden daha az maliyetli teknikler geliştirilmeli veya uygun, verimli ve ekonomik mekanizmalar seçilmelidir. Bu amacı gerçekleştirmek için öncelikle önbelleklerin bozulabilirliği araştırılmalı ve doğru bir şekilde ölçülmelidir (Zhang, 2005).

Daha önce bahsedildiği gibi soft errorlara karşı korumaya yönelik süreç ve devre temelli çözümler mevcut olsa da, biz mimarisel çözümler üzerinde duracağız.

Mimarisel çözümler güvenirliliği geliştirmeyi amaçlar. Eşlik (parite) kontrolü, hata doğrulama kodları (ECC) (Pradhan, 2003), PI bit (Weaver ve diğ., 2004), N mod fazlalık (NMR) gibi kopyalanmış mimariler, kopyalanmış önbellek (Zhang ve diğ., 2003) gibi bütünlük kontrol tekniklerini içerir.

Eşlik biti kontrol mekanizması, her 8 bit veri için 1 bit ekler. Genelde önbelleklerde hata kontrolü için kullanılır. Basit ve enerji açısından verimli olsa da, alandan %12.5 oranında kayıp verir. Buna ilave olarak, sadece tek sayılı hataları sezebilir ama hiçbirini düzeltemez. Dolayısıyla, özellikle yüksek veri bütünlüğü ve yüksek gürültülü ortamlarda çalışma gerektiren (yüksek bölgelerde) sistemler için tek başına iyi bir çözüm sunmamaktadır.

Popüler bir ECC yöntemi olan SECDED (Single Error Detection Double Error Correction-Çoklu Hata Sezme Tekli Hata Düzeltme) daha karmaşıktır. Daha fazla zaman gerektirir. İki bitlik hataları sezebilse de tek bitlik hataları düzeltebilir. Yüksek saat frekansında çalışan işlemciler için L1 veri önbelleğine erişim süresini artırır. L1 önbelleği SECDED ile korunduğunda erişim zamanı uzayabilir ve erişim için gereken süre 1 cycle'dan daha fazla olabilir. Bu nedenle, ECC temelli koruma teknikleri daha az sıklıkla erişilen önbellek hiyerarşisi bileşenleri (L2 ve L3 önbelleği) için daha uygundur.

NMR temelli teknikler çoklu kopyalama (replica) ile veri güvenirliliğini önemli miktarda artırabilseler de, önemli ölçüde ekstra alana ihtiyaç duyabilirler. Örneğin, 3'lü kopyalama kullanıldığında alandan kayıp %200'dür (Carmichael, 2001). Bu da kısıtlı sistemler için çok fazla maliyet demektir.

PI bit, (false DUE) çıkışı etkilemeyen DUE hatalarını yok etmek için kullanılan bir hata yayılım mekanizmasıdır. Parite kontrolü ile bir hata fark edildiği an, makine exception üretmek yerine etkilenen komutun PI biti ayarlanır. İş hattının tamamlanma aşamasında eğer bir hatalı komutun yanlış komut yolunda olduğuna karar verilirse makine hata olmamış gibi işlemlerine devam eder.

Mimari seviyede önbellek güvenirliliğini artırmak için diğer bir yol, önbellek içinde verinin kopyasını tutmaktır. Tekniğe bağlı olarak, kopyalama ya önbelleğin kendisinde veya ayrı bir önbellekte yapılabilir. Her ikisinin de alandan ve performanstan kayıpları vardır. Son zamanlarda (Zhang ve diğ., 2003) L1 veri önbelleği içinde güvenirliliği artırmak için ICR (önbellek içinde kopyalama mekanizması) önerdiler. ICR, aktif kullanımdaki veya henüz yeni erişilmiş veri bloklarının kopyalarını aynı önbellek alanı içinde saklamayı sağlar. En son erişildikten sonra belli bir süre erişilmeyen blok ölü diye tabir edilir. Bu bloklar veri bloklarının kopyalarının depolanması için kullanılır. Bir blok soft errordan etkilendiğinde (örneğin parite kontrolü ile belirlenebilir) bloğun doğrulanması için onun kopyası (replica) kullanılır.

Kopyaları depolamanın başka bir yolu da ayrı bir önbellek kullanmaktır. Örneğin, (Zhang, 2004) soft errorlardan etkilenen veri bloklarının doğrulanabilmesi için kopyaların depolanabileceği, kopyalama önbelleği diye de adlandırılan küçük tam çağrışımlı bir önbellek önerdi.

Bu çalışmada, en son erişilmiş olan verilerin kopyalarını aynı mevcut önbellek içinde tutmaya dayanan veri önbelleği güvenirliliğini geliştirme tekniği öneriyoruz. Bunu gerçekleştirmek için, aktif kullanımdaki blokların kopyalarını depolamak için ölü blokları kullanıyoruz. Ölü blok, kurban olarak seçilmezden önce veri barındıran bir bloktur. Bizim tekniğimiz aktif kullanımdaki blokların kopyalarını depolamak için, ölü blokları kullanması bakımından ICR'a benzemektedir. Ama L1 veri önbelleği içinde kopyaların nerede oluşturulacağını belirlemek için farklı kriterler kullanmaları bakımından bizim metot ICR'dan farklılık arz etmektedir. Ayrıca bizim tekniğimiz, önbellek içinde ölü blok bulunduğu sürece kopya oluşturabilmesine rağmen, ICR sadece belirli bölgelerde ölü blok varsa aktif kullanımdaki blokların kopyasını oluşturabilmektedir. Önceki çalışmalarla sonuçlarımızı tutarlı bir şekilde karşılaştırmak için biz bu çalışmamızda, önerdiğimiz yöntemi ve önceden önerilen iki yöntemi çeşitli kriterler kullanarak karşılaştıracağız. Bu üç kriterden ikisi önceki çalışmalarda kullanılmıştır, üçüncü kriter ise bizim bu çalışmamızda önerilmektedir. Önceki iki kritere dayalı olarak yapılan karşılaştırmaların sağlıklı olmayacağını

düşünüyoruz. Bunun temel sebebi, bu üç yöntem sadece kirli blokların kopyasını tuttuğu için, sadece kirli blokları göz önüne alan bizim önerdiğimiz kriterin kullanılmasının daha uygun olmasıdır.

BÖLÜM 4

BİZİM YAKLAŞIMIMIZ ve DENEYSEL SONUÇLAR

4.1. Soft Error Bilinçli Önbellek Mimarisi

Soft error bilinçli önbellekler, önbellekler işlemci çipinin önemli bir yüzdesini kapladığından ve korunmadığında soft error kaynaklı oluşan hatanın diğer sistem bileşenlerine kolayca yayılabileceğinden güvenirli sistem tasarımları için kaçınılmazdırlar. Oluşan bu hatalar sistemin yanlış çalışmasına veya çökmesine sebep olabilir.

Hem kirli (değiştirilmiş) hem de temiz (değiştirilmemiş) verilerin hatadan etkilenme olasılıkları aynı olsa da, biz bu çalışmamızda sadece L1 veri önbelleğindeki kirli (değişmiş) verilerdeki hatalar üzerinde duracağız. Bunun temel sebebi, temiz verinin (L1 veri önbelleğinde ki değiştirilmemiş veriler veya komut önbelleğindeki komutlar) sağlam kopyası zaten L2 önbelleğinde mevcut. Yani, temiz verilerde soft error kaynaklı bir hata oluşsa bile bu hata L2 önbelleğinde ki temiz kopyası ile doğrulanabilir. Dolayısıyla bizim amacımız L1 veri önbelleğindeki değiştirilmiş veri güvenirliliğini artırmaktır.

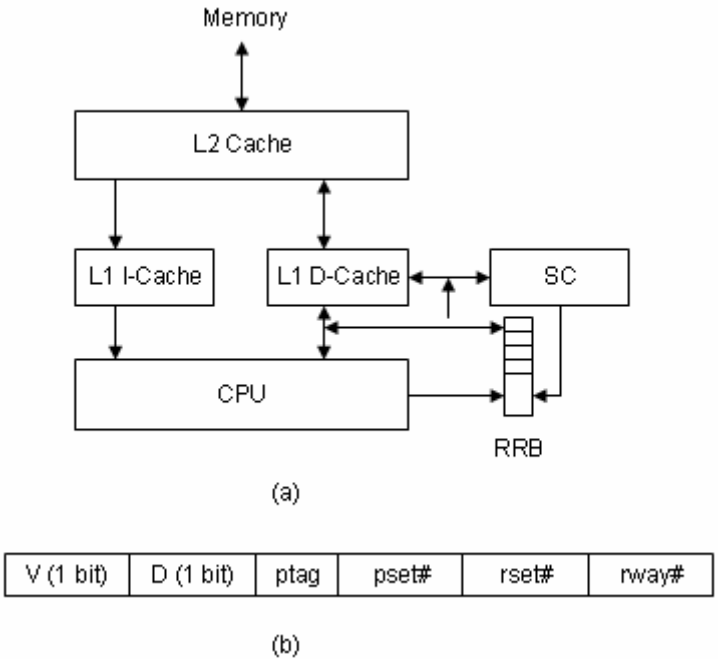
Doğrudan yazmalı önbellekler sistemdeki veri güvenirliliğini geliştirmek için kullanılabilir. L1 önbelleğine her yazma işlemini aynı bloğun L2'ye yazılması takip ettiği için, L1'deki her hangi bir veride hata bulunursa hatayı düzeltmek için L2'deki kopyası kullanılabilir. Doğrudan yazmalı önbellekleri kullanmanın bazı dezavantajları da vardır. L1 ve L2 arasında bir yazma tamponu bulunmasına rağmen doğrudan L2'ye yazmalar L2 önbellek trafiğini artırmak suretiyle performans kaybına sebep olur. Başka bir dezavantaj ise, L2'ye erişim başına harcanan enerji L1'e erişim başına gereken enerjiden oldukça fazla olmasından dolayı, bellek hiyerarşisinde dinamik enerji harcanımını oldukça artırmasıdır. Dolayısıyla, özellikle yüksek performans ve kısıtlı pil ömründe çalışması gereken sistemlerde soft errorları yönetmek için doğrudan yazma iyi bir yöntem değildir.

Yukarıda belirtilen sebeplerden ötürü, bizim yaklaşımımız, L1 sonradan yazmalı önbelleklerin güvenirliliğini artırmayı hedeflemektedir. Bu amaç için kullanılan blokların kopyalarını yine aynı önbellekte tutmaktayız. Bu konuda daha önce yapılmış bir çalışma olan ICR'da ölü blokların işgal ettiği yer aktif kullanımdaki blokların kopyalarını tutmak için kullanılmıştır (Zhang ve diğ., 2003). Ölü bloklar performans temelli önceden getirme ve sızıntı iyileştirmeleri için de kullanılabilir (Hu ve diğ., 2002; Kaxiras ve diğ., 2001; Lai ve diğ., 2001). Önceden getirme ile sızıntı iyileştirme için önbellek bloklarının enerjilerinin kesilmesi arasındaki ilişki daha önceki bir çalışmada incelendi (Kadayif ve diğ., 2006). Çeşitli ölü blok tahmin mekanizmaları vardır. Biz bu çalışmamızda Kaxiras ve diğ. (2001) tarafından önerilen, belirli bir süre boyunca kullanılmayan önbellek bloğunun ölü olarak düşünülmesine dayanan tahmin mekanizmasını kullanıyoruz. Daha açık olarak, her bir önbellek bloğu, 4-durumlu FSM (Sonlu Durum Makinesi) ile ilişkilendirilir. FSM önbellek bloğu erişilmediği sürece ilerler (adım atlar). Son duruma gelindiğinde artık o önbellek bloğu ölü olarak işaretlenir.

Bizim yöntemimiz önbellek bloklarının kopyalarını depolamak için ölü blokları kullanması bakımından ICR'a benzese de, kopyalar oluşturdukları yer bakımından ICR'dan oldukça farklıdır. ICR, kopya blokları depolamak için kullanacağı ölü blokları bulmak için *distance-k* ($k=2^i$) metodunu kullanır. Örneğin, varsayalım ki *m* kümesinde (set) ki bir aktif bloğun kopyası oluşturulacak, *distance-k* metodu $((m+2^i) \bmod N)$ kümesinde bir ölü blok bulmaya çalışılır (*N* önbellekteki toplam küme sayısı). Eğer bu deneme başarısız olursa ya *distance-*2^{i-1} *veya distance-*2^{i+1} kullanılır. Bu nedenle, önbellekte ölü blok bulunsa bile *distance-k* metodu kullanacağı ölü bloğu bazı durumlarda bulamayabileceği için ICR'da kopyalama yapamama ihtimali vardır. Buna karşın, aşağıda açıklanacağı gibi bizim yöntemimiz önbelleğin herhangi bir yerinde ölü blok bulunduğu sürece kopyalama yapmaya izin verir. Bu açıdan bakıldığında, bizim yöntemimizin kopyalama yapabilme yeteneği ICR'dan daha fazladır.

Bizim yaklaşımımızda, ölü blok bilgilerinin tutulması için gölge önbellek

(SC-shadow cache) denilen küçük ve tam çağrışımlı bir önbellek kullanıyoruz. SC önbellek mimarisi Şekil 4.1(a)'da gösterildiği gibi iki seviyeli önbellekle birlikte çalışır. SC önbelleği, veri blokları bilgilerini ve her bir bloğa karşılık gelen kopya bilgilerini tutar. SC'de ki bir kayıt 6 parçadan oluşur: V (1 bit), D (1 bit), ptag, pset#, rset# ve rway# (Şekil 4.1(b)). V ve D bitleri geçerlilik (valid) ve ölülük (dead) bitleridir. ptag ve pset# ise veri bloğunun adresinin sırasıyla tag ve set bitleridir. Eğer V biti 1 olarak ayarlanmışsa bunun anlamı, ptag ve pset# ile belirtilen birincil veri bloğunun önbellek içinde kopyasının bulunduğudur. D biti ise ölülük bitidir ve bu değerin 1 olması demek önbellek içinde rset# ve rway# ile belirlenen adresteki bloğun ölü periyoda girmesi yani ölü olarak işaretlenmesi demektir. Ve artık başka bir bloğun kopyasının depolanabilmesi için kullanılabileceği demektir. ptag ve pset# bitleri, ikisinin birleştirilmesiyle oluşturulan adresteki bloğun kopyasının olup olmadığını kontrol etmek için kullanılır. Bu bitler sadece V biti 1 olduğunda anlam ifade etmektedir. Bir bloğun birden fazla kopyası bulunabileceği için SC içinde farklı kayıtların tag kısmı aynı olabilir. rset# ve rway# bitleri, önbellek içindeki ptag ve pset# bitlerinin birleştirilmesiyle oluşturulan adresteki bloğun kopyasına konumlanmak için kullanılır. rset# bitleri L1 önbelleğini indeksleyip kümeyi (set) bulmak için kullanılır. rway# bitleri ise kümenin yolları arasında kopyanın bulunduğu bloğu seçmek için kullanılır.



Şekil 4. 1 - (a) İki seviye önbellekli bellek sistemine SC'nin entegrasyonu. (b) SC bloğu.

Şimdi önbellekte kopyaların nasıl oluşturulduğu ve bunlarla ilgili bilgilerin SC'de nasıl tutulduğuna bakalım. En son erişimin ardından yeteri kadar zaman geçerse, blok ölü periyoda girer. Eğer mümkünse D=0 ve V=0 olan bir SC kaydı belirlenir. SC tam çağrışımlı bir önbellek olduğundan arama paralel yapılabilir. D=0 ve V=0 olan bir kayıt bulunabildiyse, SC'nin ilgili kaydının rset# ve rway# bitleri ölü bloğu gösterecek şekilde güncellenir. Aynı zamanda, D biti 1 yapılır. Bir kirli bloğun kopyasını depolamaya ihtiyaç duyulduğunda SC'nin içinde D biti 1 olan kayıt aranır. Daha sonra önbellek içinde ki bu ölü bloğu bulmak için rset# ve rway# bitleri kullanılır. Bulunan ölü bloğa kopya oluşturulduktan sonra V biti 1, D biti 0 yapılır. Sonra da SC'nin ilgili kaydının ptag ve pset# bitleri güncellenir.

Çalışma anında önbellekteki bir veride soft error oluştuğunda eşlik kontrolü gibi hata sezme mekanizmaları ile bu hata tespit edilebilir. Verinin adresinin ptag ve pset# kısmı ile SC içinde arama yapılır. V biti 1 olan, ptag ve pset# kısmı verinin adresinin ptag ve pset# kısmıyla uyuşan kayıt belirlenir. Böyle bir kayıt yoksa, bu demektir ki bu bloğun kopyası tutulmuyor, bu nedenle oluşan hata doğrulanamaz. Eğer SC içinde böyle bir kayıt bulunabilirse onun rset# ve rway# bitleri birincil bloğun kopyasına konumlanmak için kullanılır. Bozulan birincil blok kopyasından doğrulanarak güncellenir. Nadiren de olsa, kopyanın kendisi de bozulabilir, bu durumda bizim mekanizmamız birincil bloğun birden fazla kopyasını tutabildiğinden başka bir bozuk olmayan kopya kullanılarak birincil blok ve onun bozulmuş kopyası düzeltilebilir. Her bir önbellek bloğu için ona karşılık gelen bir SC bloğunun bulunduğunu varsayıyoruz. Daha sonra da tartışacağımız gibi 32bit adres uzayı için ve bizim deneysel önbellek sistemi için SC bloğunun boyutu kabaca 6byte'dır.

4.2. Düşünülen Yöntemler

Bu bölümde, deneysel sonuçları daha sonra verilecek olan yöntemleri açıklayacağız. Burada üç yöntemi düşünüyoruz. İki tanesi önceki çalışmalara dayanmaktadır. Birincisi Zhang ve diğ. (2003) tarafından önerilen ICR yaklaşımı, ikincisi Zhang (2004)'ın önerdiği RC yaklaşımı ve üçüncüsü ise bizim SC

yaklaşımımız. Bunların performanslarını daha sonra karşılaştıracağız. Temel yöntem ve diğer 3 yöntem şunlardır:

Base (Temel): Normal L1 veri önbelleğinin bulunduğu durumdur. Her hangi bir kopyalama mekanizması içermez. Bütün önbellek satırları eşlik kontrolü ile korunur. Hata düzeltme yeteneği yoktur. Okuma/yazma işlemlerinin tamamlanması 1 makine cycle (devir) almaktadır.

ICR: Ölü blok tahmin tekniği ile kopyalama mekanizması içerir. Aktif kullanımdaki veri bloklarının kopyası mevcut önbellekte tutulur. Kopyalama depolama amacıyla ölü blokları bulmak için *distance-k* metodu kullanılır.

RC: Kopyalama önbelleği diye adlandırılan L1 önbelleğinden ayrı küçük ve tam çağrışımlı bir önbellektir. Hatalardan etkilenen veri bloklarını doğrulamak için depolanan bu veri bloklarının kopyaları kullanılır.

SC: Bu bizim önerdiğimiz yöntemdir. Ölü blok ve kopya bilgilerini tutmak amacıyla gölge önbellek mimarisi içerir. SC hangi blok ölü, hangi blok kopya gibi bilgileri tutan önbellektir. Ancak ICR yöntemi gibi kopyalar L1 veri önbelleğinin ölü bloklarında depolanır. ICR den farklı olarak L1 veri önbelleğinde ölü blok olduğu müddetçe kopya oluşturabilir.

Çizelge 4. 1 - Temel yapılandırmamızda kullanılan yapılandırma parametreleri ve onların değerleri.

İşlemci çekirdeği	
Fonksiyonel Birimler	8 Integer ALUs, 4 Integer çarpıcı/bölücü 8 FP toplayıcı, 4 FP çarpıcı 4 FP bölücü/karekök
RUU boyutu	256 komut
LSQ boyutu	64 komut
Fetch/Decode/Issue/Commit Genişliği	8 komut/cycle
Fetch Kuyruk Boyutu	8 komut
Önbellek ve ana bellek hiyerarşisi	
L1 Komut Önbelleği	64KB, 4-way (LRU), 64 byte blok, 1 çevrim gecikme süresi
L1 Veri Önbelleği	64KB, 4-way (LRU), 32 byte blok, 1 çevrim gecikme süresi
L2 Önbelleği	1MB birleştirilmiş, 8-way (LRU), 128 byte blocks, 12 çevrim gecikme süresi
Veri/Komut TLB	128 kayıt, tam çağrışımlı 30 çevrim ıska gecikme süresi
Ana bellek	160 çevrim gecikme süresi

Son 3 yöntemin genel özelliklerini şöyle belirtebiliriz: Bir önbelleğin kopyası ona yazma yapıldığı anda oluşturulur. Yazma işlemi sırasında, eğer kopyası yoksa ve mümkünse kopyası oluşturulur. Aksi halde, yani kopyası varsa, orijinal veri ve onun kopyası birlikte güncellenir. Hem kopya hem de kopya olmayan satırlardaki hatalar eşlik kontrolü kullanılarak sezilebilir. Hata oluşma durumunda, eğer bloğun herhangi bir kopyası yok ise blok kirli olmadığı sürece L2'den tekrar yüklenir. Blok kirlenmiş yani değiştirilmiş ise hata düzeltilemez. Hata sadece değiştirilmiş veri bloklarında oluşmuşsa kopyalar kullanılır. Eşlik kontrolü olduğundan yükleme/depolamanın temel yöntemde tamamlanması 1 cycle almaktadır. Hata olma durumunda yükleme kopyaya eriştiği için fazladan 1 cycle gerektirir.Yani tamamlanması 2 cycle zaman almaktadır.

Bizim önerdiğimiz SC yaklaşımında, kopyaları güncellemenin sebep olduğu veri önbelleği trafiğinin azaltılması için bazı mekanizmalar kullandık. Şekil 4.1(a)'da gösterildiği gibi kopya istek tamponu (RRB-Replica Request Buffer) denilen tampon kullanıldı. Kopya güncelleme istekleri veri önbelleğine sunulmadan önce geçici olarak RRB'de depolanır. RRB'nin her kaydı 4 kısımdan oluşur. İlk kısım blok içi offseti tutar (blok içinde hangi sözcüğün güncelleneceğini belirler). İkinci kısım depolama verisini tutar (belleğe yazılacak olan değer). Diğer taraftan, üçüncü ve dördüncü parçalar önbellek satırının set# ve way# bilgisini tutar. Bu da SC kaydının rset# ve rway# kısımlarına karşılık gelir. Belleğe yazma komutu işletildiğinde önbellek ve SC iş hattının geriye yazma (write-back) aşamasında aynı anda adreslenir. Dolayısıyla aynı cycle içinde önbellekteki veri güncellenir ve aynı zamanda kopyanın rset# ve rway# bitleri SC den RRB'ye okunur. Bu arada, blok içinde hangi sözcüğün güncelleneceği bilgisi ve yazma komutunun ürettiği veri tampona yazılır. Bu tampon FIFO felsefesiyle çalışır. Her bir cycle'da veri önbelleğinin yazma portunun meşgul olup olmadığı kontrol edilir. Eğer port meşgul değilse ve RRB boş değilse, bir kayıt silinir ve önbellekteki kopya set# ve way# bilgisine göre güncellenir.

4.3. Deneysel Ayarlamalar (Experimental Setup)

4.3.1. Simülasyon Ortamı (Simulation Environment)

Bütün kopyalama mekanizmalarını SimpleScalar3.0 kaynak kodunu değiştirerek gerçekleştirdik (Burger ve Austin, 1997). SimpleScalar uygulama programlarını modern işlemciler ve sistem mimarileri üzerinde simule etmeye yarayan bir araç setidir. Biz bu çalışmada Alpha işlemcilerini simule etmek için *sim-outorder* bileşenini kullandık. Çizelge 4.1 deneylerimizde kullandığımız simülasyon parametrelerini listeler. Varsayılan parametrelere ve 32-bit adres uzayına dayanarak SC önbelleğinin boyutunu hesaplayabiliriz. Veri önbelleğinin 64 kümesi bulunduğundan kümeleri indekslemek için 6 bite ihtiyaç vardır. 6 bit de blok içi byte adreslemede kullanılacağı için, tag boyutu da 32-6-6=20 olacaktır. Dolayısıyla SC bloğunun boyutu 1 (geçerlilik biti) + 1 (ölülük biti) + 20 (ptag) + 6 (pset#) + 6 (rset#) + 2 (rway#) = 36 bit olacaktır. Bir takım işler için de 4 bit daha ayırırsak, SC bloğunun boyutu 5 byte olacaktır. 256 veri önbelleği bloğu bulunduğundan SC önbelleğinin boyutu 5 X 256 = 1,25KB olacaktır. 16KB veri önbelleği ile kıyaslandığında, 1,25KB boyutuyla SC önbelleğinin önemsenmeyecek bir alan kaybı vardır.

Çizelge 4. 2 - Deneylerimizde kullanılan ölçüm programları ve onların önemli özellikleri

Deney	Veri önbelleğine erişim sayısı	Veri önbelleği ıska sayısı	Çalışma zamanları
swim	166778437	14788797	363220740
mgrid	175392837	6436205	266146426
applu	179941283	10733530	275422794
mesa	168375287	653507	138861036
galgel	233345894	7495187	188351663
art	190149631	63800455	598234421
equake	149925677	30980	117606472
ammp	137911313	79596212	538700294
lucas	87103099	930189	92747045
fma3d	94929197	1181870	124076973
sixtrack	161610129	2025392	140963374
apsi	187501267	555615	123599200
gzip	190371188	1873379	193343905
vpr	224027817	8476709	345366815
gcc	297898488	9067117	343412521
mcf	143424959	111326	119781207
crafty	216162312	1393562	177141485
parser	201594937	3816458	274134356
eon	214188456	55393	167130382
perlbmk	264351429	2175162	314717286
gap	183936775	491337	167541053
bzip2	182574040	3352422	193032537
twolf	190086153	9364916	442022420

Çizelge 4.2'de deneylerimizde kullanılan ölçüm programları ve onların veri önbelleğine erişim sayıları, veri önbelleği ıska sayıları ve çalışma zamanları verilmiştir. Tablonun birinci sütununda ölçüm programlarının isimleri verilmiştir. İkinci sütununda, ölçüm programlarının simülasyonda çalıştırılması sırasındaki veri önbelleğine erişim sayıları, üçüncü sütununda, veri önbelleği ıska sayıları ve dördüncü sütununda ise ölçüm programlarının simülasyonda çalıştırılma zamanları verilmiştir.

4.3.2. Değerlendirme Ölçümleri

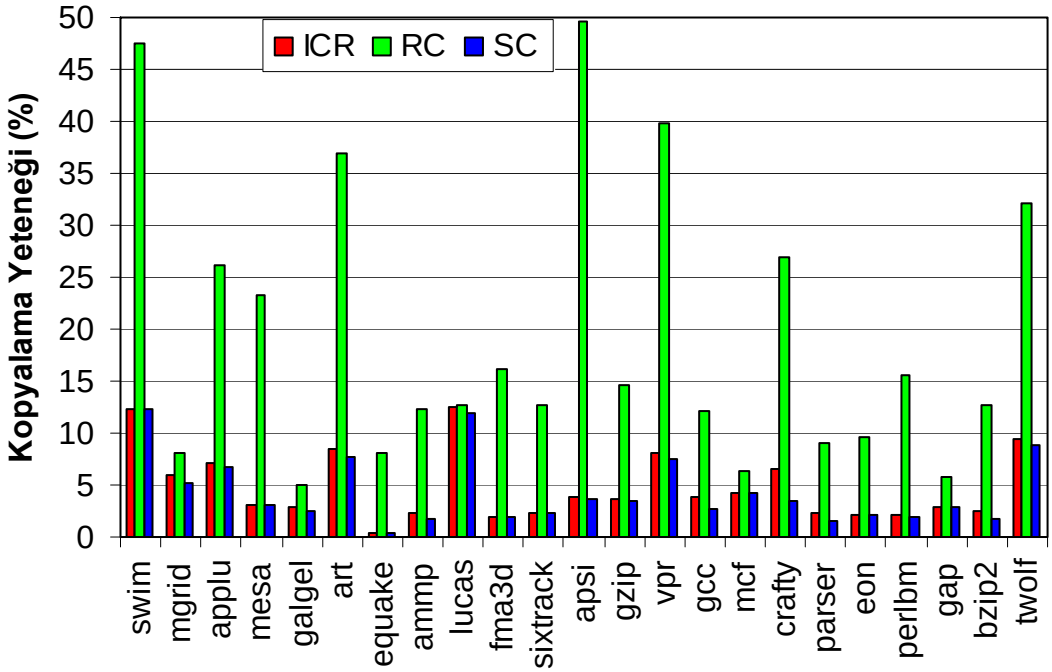
Üç yöntemin performans ve güvenirlilik iyileştirmelerini kıyaslamak için farklı değerlendirme ölçümleri düşünüyoruz. Bu ölçümler:

Çalışma Zamanı (Execution Cycle): 500 milyon komutun çalışması için geçen süre.

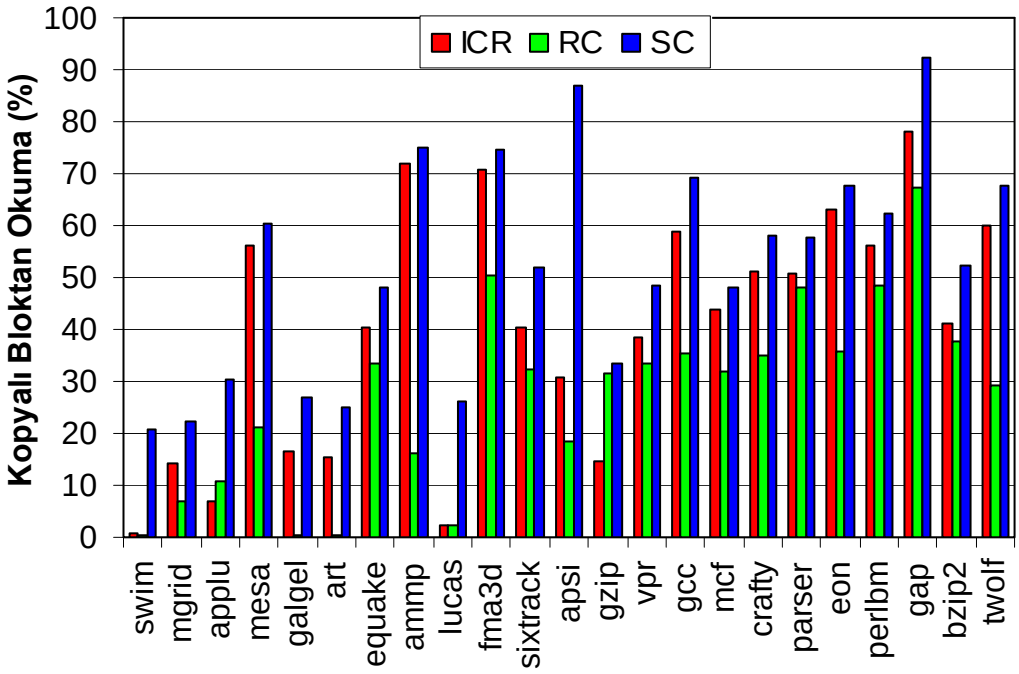
Kopyalama Yeteneği (Replication Ability): Yazmaların yüzde kaçında önbellek bloğunun kopyasının yapılabildiğidir.

Kopyalı Bloktan Okuma (Loads with Replica): Okumaların yüzde kaçının kopyaya sahip olan bloklara yapıldığını gösterir.

Kopyalı Kirli Bloktan Okuma (Loads-to-dirty with replica): Kirli bloklara yapılan okumaların yüzde kaçının kopyaya sahip olan bloklara yapıldığını ifade eder.



Şekil 4. 2 - Bu üç yöntemin kopyalama yeteneği



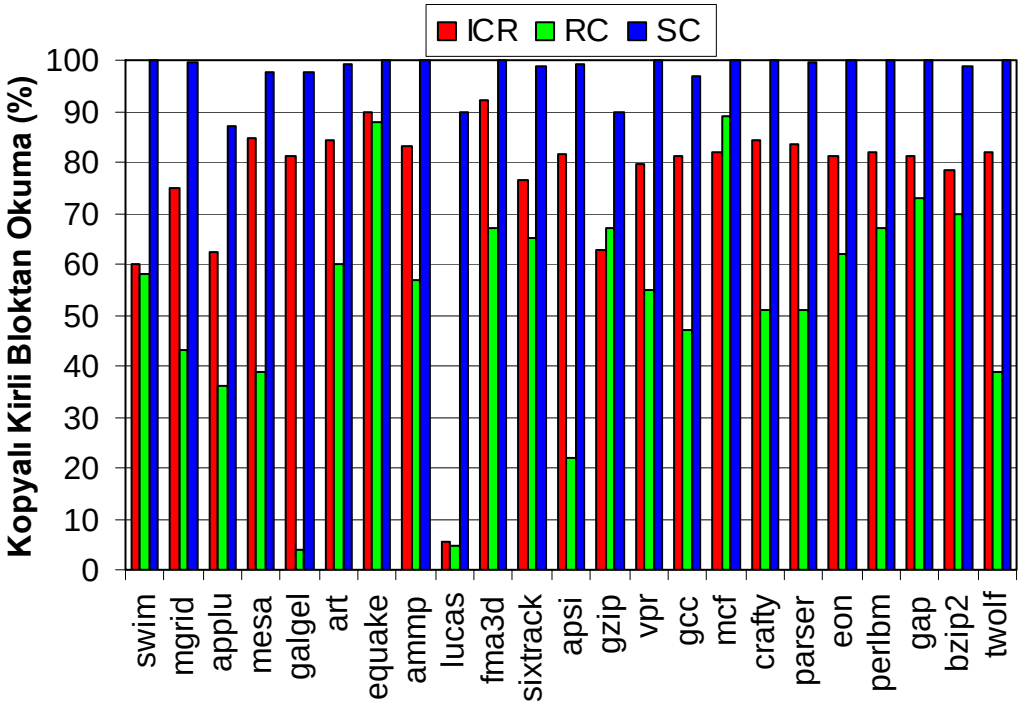
Şekil 4. 3 - Bu üç yöntemin kopyalı bloktan okuma yeteneği

İkinci ve üçüncü ölçümler Zhang ve diğ. (2003) tarafından tanıtılmasına rağmen sonuncu olandan ilk defa bu çalışmada bahsedeceğiz. Kopyalı kirli bloktan okuma ölçümün, kopyalama yeteneği ölçümü ve kopyalı bloktan okuma ölçümü ile güvenirliliği geliştirme açısından kıyaslandığında daha iyi bir ölçüm olduğuna inanıyoruz. Bunun sebebi, önerilen yöntemlerin sadece kirli önbellek bloklarının kopyasını oluşturmaya çalışmalarıdır. Dolayısıyla sadece kirli blokları düşünmek, bu yöntemlerin performanslarını yargılamada daha adil olunacağı anlamına gelir. Daha önceki çalışmalarla tutarlı olması açısından, bizim yöntemimizi diğer iki yöntemle kopyalı bloktan okuma ve kopyalı kirli bloktan okuma ölçümlerini kullanmak suretiyle de kıyaslayacağız.

4.3.3. Değerlendirme Sonuçları

Bu bölümde yukarıda bahsedilen değerlendirme ölçümlerine dayanarak üç yöntemin sonuçlarını vereceğiz. Şekil 4.2 bizim deneylerimizde üç yöntemin

kopyalama yeteneklerini göstermektedir. Şekilde gösterildiği gibi, her bir uygulama programı için üç sütun var. Soldan sağa doğru, birinci sütun ICR'ın kopyalama yeteneğini, ikinci sütun RC yönteminin kopyalama yeteneğini ve üçüncü sütun SC yönteminin kopyalama yeteneğini göstermektedir. Şekilden de açıkça görülebileceği gibi, bütün uygulamalar için RC en yüksek kopya yapabilme yeteneğine sahip, SC de en düşük kopya yapabilme yeteneğine sahiptir. Aşağıda bahsedilecek iki sebepten ötürü, daha küçük orandaki kopyalama yeteneği daha yüksek güvenirlilik sağlama anlamına gelir. Birincisi, fazla kopyalama yeteneği, yeterli sayıda kopya saklayamamaktan kaynaklanmakta ve bu da yüksek bir oranda kopya yapma ihtiyacına yol açmaktadır. İkincisi, düşük kopyalama yeteneği daha geniş bir çalışma verisinin önbellekte tutulmasından kaynaklanmakta ve bu da daha az bir oranda kopya oluşturma isteğine yol açmaktadır. Bu husus özellikle SC yöntemi için geçerlidir.



Şekil 4. 4 – 3 yöntemin kopyalı kirli bloktan okuma yeteneklerini göstermektedir.

Üç yöntemin kopyalı bloktan okuma yüzdeleri şekil 4.3'te gösterilmiştir. Şekilden de görüldüğü gibi her bir deneyde, SC en yüksek kopyalı bloktan okumaya, RC ise en düşük kopyalı bloktan okuma oranlarına sahiptir. ICR, RC ve SC'nin kopyalı bloklardan okuma ortalamaları sırasıyla %35,9, %28,9 ve %48,8'dir.

Soft errorlar tarafından bozulan bir bloğun kopyasının bulunma şansını artıracağından, kopyalı bloktan okumanın yüksek olması daha iyi veri güvenirliliğini sağlama anlamına gelir. Bazı uygulamalar için ICR ve RC yöntemlerine ait kopyalı bloktan okuma değerleri oldukça düşüktür. Örneğin, lucas uygulaması için ICR ve RC'nin kopyalı bloktan okuma oranları %0,9'dur. Bu değerin oldukça düşük olması sözü edilen bu iki yöntem için soft errorlardan etkilenen verilerin düzeltilebilmesinin çok düşük bir şansla mümkün olabileceği anlamına gelir. Ancak SC yöntemi kullanıldığında aynı uygulama için kopyalı bloktan okuma oranı %25,6'dır, bu değer de bu yöntemle hatalı blokların göreceli olarak oldukça yüksek olasılıkla düzeltilebileceğini gösterir.

Şimdi de bu üç yöntemi kopyalı kirli bloklardan okuma oranlarına göre kıyaslayacağız. Kopyalı kirli bloktan okuma kriteri sadece kopyaları tutabilen kirli bloklara odaklandığından bu üç farklı ölçüm kriteri arasında güvenirlilik artışını ölçmede kullanılabilecek en uygun kriter olduğunu düşünüyoruz. Şekil 4.4 bizim uygulama programlarımız için üç yönteme ait kopyalı kirli bloklardan yükleme oranlarını göstermektedir. Bu şekle bakarak aşağıda belirtilen temel iki gözlem yapmak mümkündür. İlki, Şekil 4.3 ve Şekil 4.4'e baktığımızda kopyalı kirli bloklardan yükleme meyili ile kopyalı bloklardan yükleme meyili üç yöntem için de paralellik gösterir. Genel olarak, her bir test programı için RC en küçük kopyalı kirli bloktan okuma yüzdesine sahip, SC yöntemi ise en büyük kopyalı kirli bloktan okuma yüzdesine sahiptir. Bu husus SC'nin veriyi soft errorlara karşı en iyi koruyan mekanizma olduğunu pekiştirir. Ayrıca kirli önbellek bloklarının kopyalarını oluşturmada ICR yöntemi RC yönteminden daha etkilidir. Veri güvenirliliğinin geliştirilmesi açısından SC'nin ICR'dan daha iyi bir yöntem olmasının temel sebebi, ICR'ın aksine SC önbellekte bir ölü blok olduğu sürece kopya oluşturma yeteneğine

sahip olmasıdır. Diğer yandan, RC önbellek blok kopyalarını tutmak için küçük büyüklükte bir tam çağrışımlı önbellek kullandığından dolayı veri güvenirliliğini geliştirmede oldukça başarısızdır. Ortalama olarak ICR, RC ve SC yöntemlerinin kopyalı kirli bloklardan yükleme oranları sırasıyla, %67,3, %51,8 ve %96,4'tür. İkinci temel gözlem, bazı test programları için hem ICR hem de RC'nin veri güvenirliliğini artırmada ki performansları oldukça düşüktür. Örneğin lucas uygulaması için ICR ve RC'nin kopyalı kirli bloklardan okuma oranları sadece %3,8 ve %1,1'dir. Diğer yandan, SC yöntemi her bir uygulama programı için veri güvenirliliğini artırmada oldukça başarılıdır. Örneğin, lucas için, SC yöntemi ile kirli veri bloklarında okuma esnasında kirli bloğun kopyasının %90 oranında kopyasının önbellekte bulunabilme ihtimali vardır.

4.3.4. Performans Sonuçları

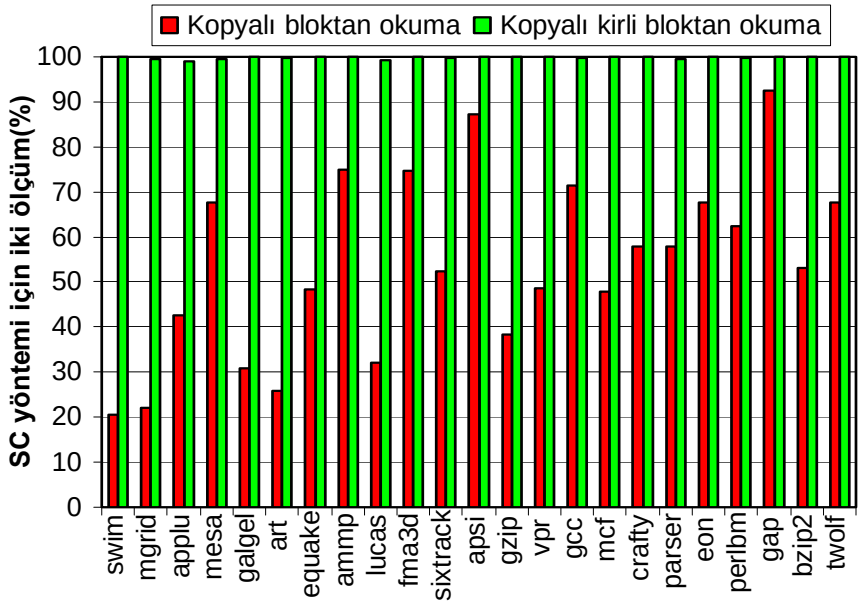
Hatırlarsak ICR ve SC aktif kullanımdaki verilerin kopyalarını ölü bloklarda saklıyordu. Bizim deneylerimizde bloğun en son erişildiğinden sonra ona belirli bir zaman periyodu kadar süre içerisinde dokunulmuyor ise ölü periyoda girdiğini varsayıyoruz (tahmin ediyoruz). Bu belirli zaman dilimine blok çürüme periyodu denir. Bu durumda bu tahmin doğru değilse, yani blok ölü periyoda girmemişse, verisine daha sonra ihtiyaç duyulacak olan bu bloğun üzerine kopya oluşturulmuş olur. Bu da fazladan önbellek ıskalamasına dolayısıyla performans kaybına yol açar. Blok çürüme periyodu artırıldıkça tahminin doğruluğu artar. Ancak önbellekte kopyaya ihtiyaç duyulduğu an kopya oluşturabilmek için ölü periyoda girmiş yeterli sayıda bloklara ihtiyaç duyacağımızdan ölü blok periyodunu gelişigüzel uzun tutamayız. Bir takım deneylerden sonra 1000 cycle'lık ölü periyot süresinin çalışma süresini pek artırmaksızın, blokların ölü blok olarak işaretlendirilmelerinin uygun olacağı kanısına varıldı. 1000 cycle'lık ölü periyot dikkate alındığında, SC mekanizması %0,9'luk bir performans kaybına yol açar. SC, ICR ile karşılaştırıldığında ortalama olarak %0,4'lük daha kötü bir performansa sahiptir. Diğer yandan, RC kopyaları tutmak için ayrı bir önbellek kullanıldığından, herhangi bir performans kaybına yol açmamaktadır.

Yukarıda bahsedildiği gibi, veri bütünlüğünü geliştirmede RC en az etkili yöntemdir. RC'nin performansını artırmanın bir yolu kopyalama önbelleğinin depolama alanını artırmaktır. Bununla beraber, alan kısıtlı sistemlerde bu uygun değildir. Veri bütünlüğünü geliştirmenin başka bir yolu da, sonradan yazma mekanizması içeren kopyalama önbelleği kullanmaktadır. Kopyalama önbelleğinde kapasite ıskası olursa, yer değiştirilen blok L2 önbelleğine geri yazılır. Bu da, L1 ve L2 arasında ki trafiğin artmasına bağlı olarak önemli performans kaybına sebep olur. Kopya oluşturulurken beklemeleri azaltmak için, L2 ile kopyalama önbelleği arasına yazma tamponu koyulabilir. Bu performans kaybının muhasebesi için 8 bloklu, yazma tamponlu, sonradan yazmalı kopyalama önbelleği içeren RC yöntemi için bazı deneyler yapıldı (Skadron ve Clark, 1997).

Bizim deneysel sonuçlarımız bazı testler için performans kaybının oldukça yüksek olduğunu gösterir. Örneğin, bzip2, swim ve applu için performans kayıpları %10,3, %8,83 ve %16,48'dir. Bu da azımsanmayacak bir miktardır. Bu sonuçlara göre, RC kopyalama önbelleği ile L2 arasına bir yazma tamponu bulundursa bile veri güvenirliliğini geliştirmek için performans açısından iyi bir yöntem değildir.

4.3.5. SC Yöntemini Geliştirme

SC yaklaşımında veri bloğu L2 önbelleğinden L1 önbelleğine getirilirken kopya barındıran blokla yer değiştirebilir. Bu da veri güvenirliliğini ters yönde etkiler. Bunu çözmek için 2 yol var. Birincisi, veri bloğuyla yer değiştirilmeden önce kopya L2 önbelleğine yazılabilir.



Şekil 4. 5 – SC yönteminin kopyalı bloktan okuma ve kopyalı kirli bloktan okuma değerleri.

Bu da, L1 önbelleğinde ki soft errorlardan etkilenen birincil bloğun düzeltilmesi için kullanılacak olan kopyayı L2'de saklamaya müsaade eder. Birincil blok daha sonra değiştirildiğinde, eğer mümkünse onun veri önbelleğinde yeni bir kopyasının oluşturulmasına çalışılır. Bu mekanizma L1 ve L2 arasında ki veri yolu trafiğini artırdığından çalışmamızda bunu düşünmedik. İkincisi, kopyayı L2 önbelleğine geri yazmak yerine, L1 önbelleği içinde farklı bir ölü bloğa taşımaktır. Bu mekanizma, SC yönteminin sağladığı veri güvenirliliğini (L1 ve L2 arasında ki veri yolu trafiğini artırmadan) artırır. Şekil 4.5 SC'nin bizim geliştirdiğimiz yöntem ile bütünleştirildiğinde ki kopyalı bloktan okuma ve kopyalı kirli bloktan okuma değerlerini gösterir. Deneylerimizde, kopyanın başka bir bloğa hareketine bağlı olarak verinin önbellekten CPU'ya getirilmesi ekstradan 4 cycle zaman dilimine ihtiyaç duyduğunu kabul ettik (L1 ıska gecikmesi ve veri yolu gecikmesine ilave olarak). Bu yolla, kopyalı bloktan okuma ve kopyalı kirli bloktan okuma ortalama %48,8'den %53,2'ye ve %96,4'ten hemen hemen %100'e çıktığını gözlemledik. Bunun için oluşan ekstra performans kaybı ise sadece %0,3'tür.

BÖLÜM 5

SONUÇ

İşlemcideki ve önbelleklerdeki soft errorlar güvenirli çalışması gereken sistemlerde ciddi bir tehdit oluşturmaktadır. Modern işlemci teknolojisindeki sızıntı enerji iyileştirmesi, transistör yoğunluğu, düşük çalışma gücü ve yüksek saat frekansında çalışma gibi gelişmeler problemi daha ciddi hale getirmektedir. Soft errorlara karşı bazı süreç (process) ve devre çözümleri vardır. Bu çözümler, genelde özellikle performans, enerji ve alan kısıtlı sistemlerde performans, enerji ve alandan kayıp gibi büyük problemlere yol açmaktadır.

Mimari çözüm, önbelleklerin veri bütünlüğünü geliştirme de başka bir seçenektir. Önbelleklerin hata koruması için kullanılan genel yöntem eşlik baytı sınaması (byte-parite) mekanizmasıdır. Hızlı ve enerji tüketimi açısından verimli olmasına rağmen, sadece tek bitlik hataları sezebilir. Ama hiçbirini düzeltemez. Bu nedenle, yüksek güvenirlilikte çalışan sistemler için yeterli değildir. ECC'ler alternatif bir mimari çözümdür. SEC-DED popüler bir ECC yöntemidir. Çift bitli hataları sezebilir ve tek bitlik hataları düzeltebilir. Eşlik sınamasının aksine daha karmaşıktır ve tamamlanması daha fazla zaman ister. Çoğu kez önbellek erişiminin önemli ölçüde uzamasına ve 1 cycle'da tamamlanamamasına yol açabilir. Dolayısıyla, SEC-DED L1 önbellekleri için uygun değildir.

Soft errorlar nadir olaylar olduğundan, her hangi bir hata olmadığı durumda hata kontrolleri uygun bir anda yapılmalı ki, erişim zamanı artmasın. Bunu başarmak için bir seçenek hata kontrolü için eşlik sınama mekanizması kullanmaktır ve aktif kullanımdaki verilerin kopyasını oluşturmaktır. Bu yöntemle eşlik baytı sınaması ile bir hata sezildiğinde, bozulan veri kopyası kullanılarak doğrulanır. Bu düşünceye dayanan önceki iki çalışma, Zhang ve diğ. (2003)'nin önerdiği L1 önbellek bloklarının mevcut önbellekte kopyalarını oluşturmak ve Zhang (2004)'ın önerdiği kopyalar ayrı küçük bir tam çağrışımlı önbellekte tutmaktır. Bizim yaklaşımımız sık erişilen verilerin kopyalarını depolamak için ölü blokları kullanmaktadır. Bu yaklaşımımıza göre önbellekte ölü blok bulunduğu sürece

kopyalama yapılabilir. Bunun Zhang (2004)'den farkı küçük çağrışımlı önbellek kopyaları depolamak için değil de kopya bilgilerini tutmak için kullanılır. Deneysel sonuçlar gösteriyor ki, bizim yöntemimiz ile L1 önbelleğinde kirli bloğunun okunması sırasında kopya bulma oranı %96,4. Bu değerler ICR ve RC için sırasıyla %67,3 ve %51,8'dir. Dolayısıyla bizim yöntemimiz L1 veri önbelleği bütünlüğünü geliştirmede önceki iki yaklaşımdan daha verimlidir.

KAYNAKLAR

AMD. 2006. *AMD Athlon™ 64 X2 Dual-Core Processor Key Architectural Features*

http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_13041%5E13043,00.html

AMD. 2006. *AMD Athlon™ 64 ProcessorKey Architectural Features*

http://www.amd.com/us-en/Processors/ProductInformation/0,,30_118_9485_9487%5e9493,00.html

Ando, H., Yoshida, Y., Inoue, A., Sugiyama, I., Asakawa, T., Morita, K., Muta, T., Motokurumada, T., Okada, S., Yamashita, H., Satsukawa, Y., Konmoto, A., Yamashita, R., Sugiyama, H. 2003. *A 1.3 GHz fifth generation SPARC64 microprocessor.* Solid-State Circuits Conference.

Austin, T.M. 1999. *DIVA: a reliable substrate for deep submicron microarchitecture design, Microarchitecture.* Proceedings. 32nd Annual International Symposium. MICRO-32. Page(s):196 – 207

Baumann, R.C. 2002. *Soft errors in commercial semiconductor technology: Overview and scaling trends*. Reliability Physics Tutorial Notes, Reliability Fundamentals, IEEE.

Baumann, R.C. 2002. *The impact of technology scaling on soft error rate performance and limits to the efficacy of error correction*. In Digest. International Electron Devices Meeting.

Bryg, W., Alabado, J. 2005. *The UltraSPARC T1 Processor-Reliability,Availability, and Serviceability*. UltraSPARC Processors Documentation, Sun Microsystems Inc.

Burger D.C., Austin, T.M. 1997. The Simplesclar Toolset, version 2.0, Tech. Rep. http://www.simplescalar.com/

Butts, J.A. ve Sohi, G.S. 2002. *Dynamic Dead-Instruction Detection and Elimination.* 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X).

Calin, T., Nicolaidis, M., Velazco, R. 1996. *Upset hardened memory design for submicron CMOS technology*. IEEE Transactions on Nuclear Science, vol. 43, no. 6.

Cannon, E. H., Reinhardt, D. D., Makowenskyj, P. S. 2004. *SRAMSER in 90, 130 and 180nm Bulk and SOI Technologies*. International Reliability Physics Symposium.

Carmichael, C. 2001. *Triple module redundancy design techniques for virtex FPGAs*. Xilinx Aplication Notes 197, v1.0.

Chen, C. L., and M. Y. Hsiao. 1992. *Error-correcting codes for semiconductor memory applications: a state of the art review*. Reliable Computer Systems-Design and Evaluation, pages 771-786, Digital Press, 2nd Edition.

Czeck, E.W., and Siewiorek, D. 1990. *Effects of Transient Gatelevel Faults on Program behavior*. In Proc. Intl. Symp. On Fault-Tolerant Computing.

Degalahal V., Vijaykrishnan, N., and Irwin, M. J. 2003. *Analyzing soft errors in leakage optimized SRAM design*. Proceedings of VLSI Design Conference.

Degalahal, V.S.R. 2005. *Soft Errors:Modeling and Interactions With Power Optimizations*. PhD Dissertation (Doktora Tezi). The Pennsylvania State University, USA. s: 33-37.

Handy, J. 1998. *The Cache Memory Book The autoritative reference on cache design* (2nd edition). Academic Press. Santa Clara, USA.

Hareland, S., Maiz, J., Alavi, M., Mistry, K., Walstra, S., and Dai, C. 2001. *Impact of CMOS scaling and SOI on soft errorrates of logic processes*. VLSI Technology Digest of Technical Papers.

Hazucha, P., Svensson, C. 2000. *Impact of CMOS technology scaling on the atmospheric nuutron soft error rate*. In IEEE Transactions on Nuclear Science, 47(6).

Hennessy, J.L., and Patterson, D.A. 2003. *Computer Architecture:A Quantitative Approach*. Morgan Kaufmann, third edition.

Hiremane, R. 2005. *From Moore's law to Intel innovation-prediction reality.* IntelCorporation. http://www.intel.com/cd/corporate/techtrends/emea/tur/210462.htm

Howe, D. 2006. *The Free Online Dictionary of Computing.* http://www.cacs.louisiana.edu/~mgr/404/burks/foldoc/41/32.htm

Hu, Z., Kaxiras, S., and Martonosi, M. 2002. *Timekeeping in the memory system: predicting and optimizing memorybehavior*. Proceedings of International Symposium on Computer Architecture.

Intel. 2006. *An Owerviev of Cach.* http://www.intel.com/design/intarch/papers/cache6.htm

Jouppi, N.P. 1990. *Improving Direct-Mapped Cache Performance by the Audition of a Small Fully-Associative Cache and Prefetch Buffers.* Proc. Int'l Symp. Computer Architecture (ISCA).

Kadayif, I., Kandemir, M., and Li., F. 2006. *Prefetching-aware cache line turnoff for saving leakage energy*. Proceedings of Asia and South Pacific Design Automation Conference. Yokohoma, Japan.

Kadayif, I. and Kandemir, M. 2007. *Modeling and improving data cache reliability.* In Proc. the International Conference on Measurement and Modeling of Computer Systems (ACM SIGMETRICS), San Diego, California.

Karnik, T., Bloechel, B., Soumyanath, K., De, V., and Borkar, S. 2001. *Scaling trends of cosmic rays induced soft errors in static latches beyond 0.18µ*. Proceedings of Symposium on VLSI Circuits Digest of Technical Papers.

Karnik, T., Hazucha, P., and Patel, J. 2004. *Characterization of soft errors caused by single event upsets in CMOS processes*. IEEE Transactions on Dependable and Secure Computing, 1(2):128–143.

Kaxiras, S., Hu, Z., and Martonosi, M. 2001. *Cache decay: exploiting generational behavior to reduce cache leakage power*. Proceedings of International Symposium on Computer Architecture.

Kim, S. ve Somani, A. 1999. *Area Efficient Architectures for Information Integrity Checking in Cache Memories.* Proc. Int'l Symp. Computer Architecture, pp. 246-256.

Kim, S., and Somani, A.K. 2002. *Soft Error Sensitivity Characterization for Micro- processor Dependability Enhancement Strategy*, Proceedings of the International Conference on Dependable Systems and Networks (DSN).

Kin, J., Gupta, M., Mangione-Smith W.H. 1997. T*he Filter Cache: An Energy Efficient Memory Structure* Proc. MICRO.

Lai, A. C., Fide, C., and Falsafi, B. 2001. *Dead-block prediction and dead-block correlating prefetchers*. Proceedings ofInternational Symposium on Computer Architecture.

Li, X., Adve, S. V., Bose, P., and 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., Reinhardt, S. K., and Austin, T. 2003. *A systematic methodology to compute the ar-chitectural vulnerability factors for a high-performance microprocessor*. In Proc. 36th International Symposium on Microarchitecture.

Mukherjee, S. S., Emer, J., and Reinhardt, S. K. 2005. *The soft error problem: an architectural perspective*. Proceedings of International Symposium on High-Performance Computer Architecture.

Nguyen, H. T., and Yagil, Y. 2003. *A systematic approach to SER estimation and solutions*. In Proc. IEEE International Reliabilty Physics Symposium.

Normand, E. 1996. *Single Event Upset at Ground Level*. Nuclear Science, IEEE Transactions on, Volume 43, Issue 6, Part 1, Page(s):2742–2750.

Pas, R. 2002. *Memory Hierarchy in Cache-Based Systems*. High Performance Computing. Sun Microsystems, Inc.

Pradhan, D.K. 2003. *Fault-tolerant computer system design*. Computer Science Press.

Ray, J., Hoe, J.C., and Falsafi, B. 2001. *Dual Use of Superscalar Datapath for Transient-Fault Detection and Recovery*," Proc. MICRO.

Rotenberg, E. 1999. *Exploiting large ineffectual instruction sequences.* Technical Report, NC State University.

Saggese, G.P., Wang, N.J., Kalbarczyk, Z.T., Patel, S.J., Iyer, R.K. 2005. *An experimental study of soft errors in microprocessors.* Micro, IEEE Volume 25, Issue 6. Page(s):30–39.

Scarbrough, C., Gomaa, M.A., Vijaykumar, T.N., Pomeranz I. 2003. *Transient Fault Recovery for Chip Multiprocessors.* 30th Annual International Symposium on Computer Architecture (ISCA).

Seifert, N.; Xiaowei Zhu; Massengill, L.W. 2002. *Impact of scaling on soft-error rates in commercial microprocessors,* Nuclear Science, IEEE Transactions on, Volume 49, Issue 6, Part 1,Page(s):3100–3106.

Shivakumar, P., Kistler, M., Keckler, S., Burger, D., Alvisi, L. 2002. *Modeling the Effect of Technology Trends on Soft Error Rate of Combinational Logic*. Proceedings of the International Conference on Dependable Systems and Networks.

Skadron, K. and Clark, D. W. 1997. *Design issues and tradeoffs for write buffers*. Proceedings of International Symposium on High-Performance Computer Architecture.

Wang, N., et al. 2004. *Characterizing the Effects of Transient Faults on aModern High-Performance Processor Pipeline*. In Proc. Intl. Conf. on Dependable Systems and Networks.

Weaver, C., Emer, J., Mukherjee, S. S., Reinhardt, S. K. 2004. *Techniques to reduce the soft error rate of a high-performance microprocessor*. Proceedings of 31st International Symposium on Computer Architecture.

Zhang, W., Gurumurthi, S., Kandemir, M., and Sivasubramaniam, A. 203. *ICR: in-cache replication for enhancing datacache reliability*. International Conference on Dependable Systems and Networks.

Zhang, W. 2004. *Enhancing data cache reliability by the addition of a small fully-associative replication Cache*. Proceedings of ACM International Conference on Supercomputing.

Zhang, W. 2005. *Computing Cache Vulnerability to Transient Errors and its Implication.* Dept of ECE, Southern Illinois Univ. Carbondale, IL 62901.

Ziegler, J. F. 1996. *Terrestrial cosmic rays*. IBM Journal of Research and Development, 40(1):19–39.

Zilles, C. 2004. *Computer Architecture.*

ÇİZELGELER

Çizelge No **Çizelge Adı** **Sayfa**

Çizelge 2.1 -1024 boyutlu iki dizinin elemanlarının toplanması 12

Çizelge 2.2 - Erişilen verilerin isabet/ıska durumları 12

Çizelge 4.1 - Temel yapılandırmamızda kullanılan yapılandırma parametreleri ve onların değerleri. 40

Çizelge 4.2 - Deneylerimizde kullanılan ölçüm programları ve onların önemli özellikleri 43

ŞEKİLLER

Şekil No **Şekil Adı** **Sayfa**

Şekil 2. 1 - Temel bellek sistemi..... 4

Şekil 2. 2 - Bellek sisteminin çoklu seviyeleri..... 5

Şekil 2. 3 - Genel sistem mimarisi..... 11

Şekil 2. 4 - Doğrudan haritalanmış önbellek 14

Şekil 2. 5 - Tam çağrışımlı önbellek..... 15

Şekil 2. 6 – 2 yollu küme çağrışımlı önbellek 17

Şekil 3. 1 - Aygıta çarpan tanecik aygıt boyunca bir elektron boşluğu oluşturur..... 21

Şekil 3. 2 - Mikroişlemcide ki bozulan bir bitin muhtemel sonuçlarının tasnifi 26

Şekil 4. 1 - **(a)** İki seviye önbellekli bellek sistemine SC'nin entegrasyonu.
(b) SC bloğu..... 38

Şekil 4. 2 - Bu üç yöntemin kopyalama yetenekleri 44

Şekil 4. 3 - Bu üç yöntemin kopyalı bloktan okuma yetenekleri..... 45

Şekil 4. 4 – Bu üç yöntemin kopyalı kirli bloktan okuma yetenekleri 46

Şekil 4. 5 - SC yönteminin kopyalı bloktan okuma ve kopyalı kirli bloktan
Okuma değerleri..... 50

YAŞAM ÖYKÜSÜ

Kişisel Bilgiler

Adı Soyadı	:Davut AKÇİÇEK
Doğum yeri ve yılı	:Manisa/Salihli-1983
Adres	:Çanakkale Onsekiz Mart Üniversitesi Mühendislik-Mimarlık Fakültesi Bilgisayar Mühendisliği Bölümü Terzioğlu Yerleşkesi, 17100 ÇANAKKALE
Tel	:0 505 854 55 21
e-posta	:dakcicek@comu.edu.tr

Eğitim Durumu

2000-2004 : Çanakkale Onsekiz Mart Üniversitesi-Bilgisayar Mühendisliği Bölümü

1996-2000 : Salihli Yabancı Dil Ağırlıklı Türkbirliği Lisesi, Salihli/MANİSA

1993-1996 : Köprübaşı Atatürk İlköğretim Okulu, Köprübaşı/MANİSA

1988:1993 : Köprübaşı Milli Egemenlik İlkokulu, Köprübaşı/MANİSA

Stajlar

Temmuz 2003 : Oyak Teknolojileri Bilişim ve Kart Hizmetleri A.Ş., İSTANBUL

Temmuz 2002 : Figen Yazılım Evi, ÇANAKKALE

Mesleki Deneyim

2004-2007 : Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü, Arş.Gör.

2003-2004 : Çanakkale Figen Yazılım Evi (Part-time)

Çalışma ve İlgi Alanları

Yazılım, Sistem Yönetimi, Veritabanı Yönetimi