ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

ÖNBELLEKLERDE ÖNCEDEN GETİRME İLE SOFT ERRORLAR ARASINDAKİ İLİŞKİLERİN ARAŞTIRILMASI

Olcay KABAL

Yrd. Doç. Dr. İsmail KADAYIF

Ocak, 2007
ÇANAKKALE

ÖNBELLEKLERDE ÖNCEDEN GETİRME İLE SOFT ERRORLAR ARASINDAKİ İLİŞKİLERİN ARAŞTIRILMASI

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

Hazırlayan
Olcay KABAL

Danışman
Yrd. Doç. Dr. İsmail KADAYIF

Ocak, 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 : Doç. Dr. İsmail TARHAN

Üye : Yrd. Doç. Dr. İsmail KADAYIF

Üye : Yrd. Doç. Dr. İbrahim TÜRKYILMAZ

Kod No :

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım.

Enstitü Müdürü

TEŞEKKÜR

Yüksek lisans tez çalışmama destekleri ve yardımlarından ötürü danışmanım Sayın Yrd.Doç.Dr. İsmail KADAYIF'a;

Maddi ve manevi desteklerini benden hiçbir zaman esirgemeyen aileme;

Bana destek olan ve bilgilerini benden esirgemeyen çalışma arkadaşlarım Arş.Gör. Davut AKÇİÇEK ve Arş.Gör. Selçuk KOYUNCU'ya sonsuz teşekkürlerimi sunarım.

Olcay KABAL
2007

SİMGELER VE KISALTMALAR

ACE (Architecturally Correct Execution): Mimarisel doğru çalışma bitleri

ALU (Arithmetic Logic Unit): Aritmetik mantık birimi

AP (Active Period): Aktif Periyot

APEC (Active Period Error Contribution): Aktif periyot hata katkısı

AVF (Architectural Vulnerability Factor): Mimari bozulabilirlik faktörü

AVFC (Architectural Vulnerability Factor for Caches): Önbellekler için mimari bozulabilirlik faktörü

C: Sığa

CPU (Central Processing Unit): Merkezi İşlem Birimi

D-Cache (Data Cache): Veri Önbelleği

DFG (Data Flow Graph): Veri akış grafı

DP (Dead Period): Ölü periyot

DPEC (Dead Period Error Contribution): Ölü periyot hata katkısı

DRAM (Dynamic RAM): Dinamik RAM

DRE (Detected Recoverable Errors): Sezilebilir düzeltilebilir hatalar

DUE (Detected Unrecoverable Errors): Sezilebilir düzeltilemez hatalar

EG (Error Generation): Hata oluşumu

EGS (Error Generation Set): Hata oluşumu kümesi

EP (Error Propagation): Hata yayılımı

EPS (Error Propagation Set): Hata yayılımı kümesi

ECC (Error Correcting Codes): Hata düzeltme kodları

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

FIT (Failure in Time): Zaman içindeki başarısızlık

FP: Floating pointer

I-Cache (Instruction Cache): Komut önbelleği

ISA (Industry Standard Architecture): Endüstri standardı mimari

L1 (Level 1): Birinci seviye

L2 (Level 2): İkinci seviye

L3 (Level 3): Üçüncü seviye

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

LSQ (Load Store Queue): Yükleme kaydetme kuyruğu

MTBF (Mean Time Between Failures): Başarısızlıklar arasındaki ortalama zaman

MTTF (Mean Time to Failure): Başarısızlığa olan ortalama zaman

NMR (N modular redundancy): N mod fazlalık

OBL (One Block Lookahead): Sonraki bir bloğu önceden getirme

PC (Program Counter): Program sayacı

RAM (Random Access Memory): Rasgele erişimli bellek

RPT (Reference Prediction Table): Referans tahmin tablosu

RUU (Register Update Unit): Yazmaç güncelleme birimi

SDC (Silent Data Corruption): Sessiz veri bozulması

SECDED (Single Error Correct Double Error Detect): Tekli hata düzeltme çiftli hata sezme

SEU (Single Event Upset): Tekil durum değişimi

SOI (Silicon-on-Insulator): Silikon yalıtkanı

SRAM (Static RAM): Statik RAM

TEGP (Total Error Generation and Propagation): Toplam hata üretimi ve yayılımı

TEP (Total Error Propagation): Toplam hata yayılımı

VLSI (Very Large Scale Integration): Çok geniş ölçekte tümleşiklik

V: Volt

ÖNBELLEKLERDE ÖNCEDEN GETİRME İLE SOFT ERRORLAR ARASINDAKİ İLİŞKİLERİN ARAŞTIRILMASI

ÖZET

CPU ve ana bellek arasındaki hız farkını azaltmak için tasarlanan önbellekler yüksek performanslı modern bilgisayar sistemlerinde artarak kullanılmaya devam etmektedir. Sıkça kullanılan verileri depolayarak ve CPU'nun veri ihtiyaçlarını oldukça yavaş olan DRAM'den ziyade hızlı erişilebilir bir SRAM'den karşılamasını sağlayarak sistem performansını artırmaktadırlar. Önbellekler oldukça iyi performans hızlanması sağlamalarına rağmen, bazı uygulamalarda CPU, çalışma zamanına göre azımsanmayacak bir zamanı sadece veri istekleri için harcar. Veri önceden getirmesi bu problemi azaltmak için sunulan çözümlerden biridir. Veri önceden getirmesi ile verinin düşük seviyeli bir bellek bileşeninden yüksek seviyeli bir bellek bileşenine ihtiyaç duyulmadan önce getirilerek veriye gerçekten ihtiyaç duyulduğunda CPU'nun veriyi bellek hiyerarşisinde en yüksek seviyede bulması sağlanmaktadır.

Öte yandan, boyutları giderek azalan ölçekte gelişen işlem teknolojisine paralel olarak soft errorlar önemli bir tasarım problemi olarak ortaya çıkmıştır. Yüksek enerjili tanecik çarpmalarından kaynaklanan soft errorlar SRAM bellek birimlerinde depolanan değerleri değiştirerek işlemcilerin güvenilir bir şekilde çalışmalarına büyük bir engel teşkil etmektedir. Bir milyar transistör içeren işlemcilere yönelen eğilim önbelleklerde transistör bütçesine %90'dan fazla kaynak ayrılmasını beraberinde getirmiştir. Bunun anlamı, mikroişlemci temelli bir sistemin güvenilirliğinin büyük oranda doğru önbellek işlemlerine bağlı olacağıdır.

Bu tezde, L1 veri önceden getirmesinin soft errorlar üzerine etkilerini inceledik. Bu amaçla, önce L1 veri önbellekleri için bir soft error modeli üzerine odaklanıyoruz ve bu modeli temel alarak AVFC (Architectural Vulnerability Factor for Caches – Önbellekler İçin Mimari Bozulabilirlik Faktörü) denilen bir ölçüt tanımlıyoruz. AVFC, önbellekteki bir bozulmanın programın sonuç çıktısında görülebilme olasılığını ifade eder. Simülasyon ortamında AVFC ölçütünü hesapladık ve SPEC2000 kullanarak deneyler yaptık. Değerlendirmemiz, önceden getirmenin, genel olarak, kirli önbellek bloklarının L2 önbelleğine blok temelli geri yazılması durumunda önbelleğin soft errorlara maruz kalma riskini azalttığıdır.

Anahtar Sözcükler: Önbellekler, Önceden Getirme İşlemi, Soft Error

CAPTURING THE INTERACTIONS BETWEEN PREFETCHING AND SOFT ERRORS IN CACHE MEMORIES

ABSTRACT

Designed for abridging speed gap between the CPU and main memory, cache memories have been increasingly used in modern high perpormance computer systems. They improve system performance by storing frequently used data, and allowing CPU to satisfying most of the data requirements in a fast accessible SRAM instead of accessing a quite slow DRAM. Although cache memories can bring a quite performance speedup, for some applications, the CPU spends a considerable portion of the execution time on just waiting for data demands. Data prefetching is one of the solutions to mitigate this problem. With data prefetching, bringing data from a low level memory component to a high level memory component is started earlier than needed so that the CPU finds the data at the highest level in the memory hierarchy when the data are actually needed.

On the other hand, in parallel to ever-scaling process technology, soft errors have been emerging an important design problem. Induced by high energetic particle strikes, soft errors may flip the values stored in SRAM cells, and thus presenting a significant obstacle for processors to run in a reliable manner. With the current trend towards billion transistor processors, the transistor budget of caches will be more than 90%, meaning in practice that the reliability of a microprocessor based system will largely be depend on the correct operation of cache memories.

In this thesis, we investigate the effects of the L1 data prefetching on soft errors. For this aim, we first focus on a soft error model for L1 data caches, and then based on this model we define a metric called AVFC (Architectural Vulnerability Factor for Caches). AVFC represents the probability that a fault in the cache can be visible in the final output of the program. We have calculated the AVFC metric within a simulation platform and made experiments using the SPEC2000. Our evaluation indicates that data prefecthing, in general, reduces the risk of cache's capturing soft errors when dirty cache blocks are written back to the L2 cache on a block basis.

Key words: Caches, Prefetching, Soft Error

İÇERİK **Sayfa**

TEZ SINAVI SONUÇ BELGESİ i
TEŞEKKÜR ii
SİMGELER VE KISALTMALAR iii
ÖZET v
ABSTRACT vi

BÖLÜM 1 – GİRİŞ 1

BÖLÜM 2 – ÖNBELLEKLERE GENEL BAKIŞ 2
  2.1. Önbellek Nedir? 2
  2.2. Önbellek Mimarisi 4
  2.3 Önbelleğe Veri Yazma Yöntemleri 7
  2.4 Önbellek Organizasyonları 9
    2.4.1 Doğrudan Haritalanmış Önbellek (Direct Mapped Cache) 9
    2.4.2 Tam Çağrışımlı Önbellek (Fully Associative Cache) 10
    2.4.3 Küme Çağrışımlı Önbellek (Set Associative Cache) 11
  2.5 Yer Değiştirme Algoritmaları 13
    2.5.1 Son Zamanlarda En Az Kullanılan Algoritması (Least Recently Used - LRU) 13
    2.5.2 Rasgele Algoritması 14
    2.5.3 İlk Giren İlk Çıkar Algoritması (First In First Out-FIFO) 14

BÖLÜM 3 – ÖNCEDEN GETİRME İŞLEMİ 15
  3.1 Önceden Getirme İşlemi Nedir? 15
  3.2 Önceden Getirme İşleminin Tarihçesi 18
  3.3 Önceden Getirme İşleminin Türleri 18
    3.3.1 Yazılım Tabanlı Önceden Getirme İşlemi (Software Prefetching) 18
    3.3.2 Donanım Tabanlı Önceden Getirme İşlemi (Hardware Prefetching) ... 20
      3.3.2.1 Bir Sonraki Bloğu Önceden Getirme (Next-Line Prefetching) 21
      3.3.2.2 Markov Yöntemi 22
      3.3.2.3 Akış Tamponu (Stream Buffer) Yöntemi 23
      3.3.2.4 Mesafeye Dayalı Önceden Getirme (Stride-based Prefetching) 23

3.3.3 Donanıma ve Yazılıma Dayalı Önceden Getirme İşlemleri Arasındaki Farklar 24
3.4 Önceden Getirme Yaparken Dikkat Edilmesi Gereken Hususlar 25

BÖLÜM 4 – SOFT ERROR KAVRAMI 26
4.1 Soft Error Nedir? 26
4.2 Soft Error Türleri 28
4.3 Soft Error Ölçümleri ve Soft Error Oranı Azaltma Yöntemleri 29
4.4 Sanayide Soft Error Etkileri 32

BÖLÜM 5 – SOFT ERROR MODELİ 34
5.1 Hata Oluşumu 36
5.2 Hata Yayılımı 36
5.3 AVFC (Architectural Vulnerability Factor for Caches) 37
5.4 Örnek 39

BÖLÜM 6 – ÖNCEDEN GETİRME İLE SOFT ERRORLAR ARASINDAKİ İLİŞKİLER 41
6.1 Kullanılan Araçlar 41
6.2 Deneysel Sonuçlar 43
6.2.1 Performans Kazancı 43
6.2.2 Soft Error Etkisi 46
6.2.3 APEC ve DPEC Değerlerinin Dağılımları 49
6.2.4 Değiştirilmiş Sözcüklerin Birikimsel Dağılımları 53

BÖLÜM 7 – SONUÇ VE TARTIŞMA 55

KAYNAKLAR 56

Çizelgeler I
Şekiller II
Yaşam Öyküsü V

BÖLÜM 1

GİRİŞ

Bilgisayar mimarisi alanındaki teknolojik gelişmeler günümüzde de artarak devam etmektedir. Daha fazla transistör kullanımı daha yüksek işlem gücünü işlemcilere kazandırmaktadır. Moore kanuna göre artan transistör sayıları gün geçtikçe daha küçük hacimler kaplamaya başlamıştır. Bununla birlikte mikroişlemci ve bellek performanslarının gelişim hızları arasındaki fark giderek açılmaktadır. Bu durum işlemcilerin bellek beklemeleri için duraksamalarına neden olmaktadır. Bellek beklemelerinin etkisini azaltmanın bir yolu veriyi önceden işlemcinin yakınına getirmeye çalışmak ve veriye ihtiyaç duyulduğunda onu mümkünse bellek hiyerarşisinin en üst seviyesinde hazır bulmaktır.

Soft errorlar gelişen önişlemci teknolojisine paralel olarak önemli bir sorun olarak ortaya çıkmaktadır. Temel sebebi paketleme maddelerinden ısı etkisiyle kaynaklanan alfa tanecikleri ve kozmik ışınlardan nötron tanecikleri gibi taneciklerin çarpmalarıdır (Mukherjee ve diğ., 2003). Önbellekler CPU'nun transistör bütçesinin önemli bir kısmını oluşturduğundan, doğru hesaplamalar büyük bir oranda önbelleklerin güvenilir olmasına bağlı olmaktadır.

Bu çalışmamızda temel amaç önceden veri getirmenin soft errorlara olan etkisini araştırmaktır. Bunun için önce önbellekler için bir soft error modeli tanımladık. Bu modele göre AVFC kavramını tanımladık. AVFC, önbellekte oluşacak bir soft errorın hangi olasılıkla kendisini program çıktısında göstereceğini ifade eder. Önbellekler için soft error niteliği bu modele göre hesaplandı.

Tez çalışmasında ilerleyen bölümler şu şekilde düzenlenmiştir. Bölüm 2'de önbellekler hakkında temel kavramlar ve çalışma prensipleri, bölüm 3'te önceden getirme işlemi ve türleri, bölüm 4'te soft error kavramı ve bölüm 5'te önerdiğimiz soft error modeli anlatılmaktadır. 6. bölüm deneysel sonuçlarımızın gösterildiği ve açıklandığı bölümdür. Çalışmanın neticesinde sonuçların değerlendirilmesi kısmı bölüm 7'de yer almaktadır.

BÖLÜM 2
ÖNBELLEKLERE GENEL BAKIŞ

2.1 Önbellek Nedir?

Teknolojideki gelişmelere rağmen mikroişlemciler ana bellekten hala çok hızlıdır. Bütün uygulamaların performansındaki bellek erişim zamanı artarak bir darboğaz haline geldi. Sonuç olarak bir uygulama, zamanının önemli bir bölümünü verileri beklemek için harcamaktadır. Bu da performansı olumsuz etkilemektedir. Dolayısıyla işlemcinin saat frekans hızı artsa da ondan faydalanamamaktadır. Bu problemin üstesinden gelmek için bir çözüm, işlemci ile ana belleğin arasına küçük ve çok hızlı bir tampon bellek yerleştirmektir. Bu tampon bellek genellikle önbellek (cache memory) diye adlandırılır (Sun, 2006). Önbellek, çok küçük, hızlı, ana belleğin en son erişilen kısımlarını (veri ve komutları) tutan statik RAM (SRAM) bellektir (Intel, 2006).

Uygulamanın veriyi ana bellek yerine önbellekten getirmesi avantaj sağlar. Veriye erişim zamanı kısalır. Uygulamanın performansı artar. Tabii ki ana bellek ile önbellek arasında hala bir trafik vardır. Ama çok küçüktür. Uygulamaların çok büyük bir kısmı önbelleklerden faydalanır (Sun, 2006).

En son erişilen verileri tutan küçük ve yüksek hızlı bellek, aynı verilere tekrar erişimi hızlandırmak için tasarlanmıştır. Genelde verileri ve komutları tutan önbellekler ayrıdır. Veri ana bellekten okunduğunda veya ana belleğe tekrar yazıldığında, bir kopyası da önbellekte saklanır. İlgili bellek adresi de saklanır. İşlemci işleyeceği veriyi önce önbellekte arar. Eğer önbellekte var ise geriye döndürülür ve ana belleğe gidilmez. Eğer bulunamaz ise o zaman veri ana bellekten okunur sonra önbelleğe yazılır ve oradan da yazmaçlara kaydedilir (Howe, 2006).

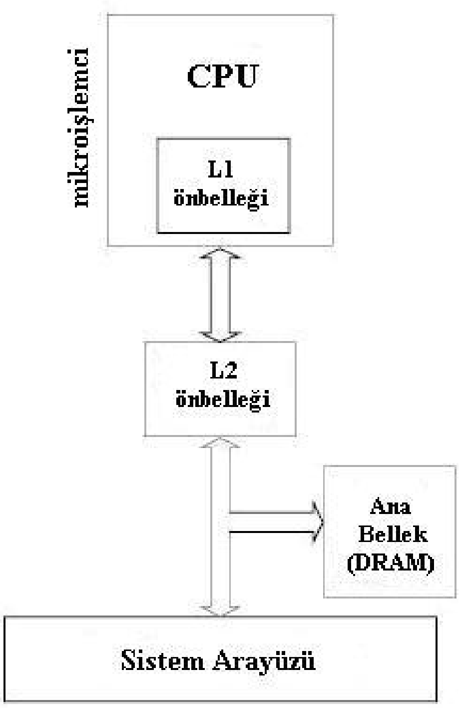
Önbellek ana bellekten daha hızlı çiplerden yapıldığı için önbellek erişiminin tamamlanması, ana belleğin erişiminin tamamlanmasından daha hızlı olur. Önbellek CPU ile aynı entegre devre üzerinde bulunabilir. Bu da erişim zamanını azaltır. Bu önbelleğe genelde birinci seviye önbellek (Level 1 – L1) denir. Daha yavaş ve CPU'nun çipinin dışındaki önbelleğe de ikinci seviye önbellek (Level 2 - L2) denir (Howe, 2006). Tasarımcılar üçüncü seviye önbellek (Level 3 – L3) mimarilerini de bilgisayar performansını iyileştirmesi bakımından geliştirmektedirler.

Önbelleğin en önemli özelliği isabet oranıdır (hit rate). Bu da tüm bellek erişim istekleri ile bunların kaçında önbellekten faydalanıldığının yüzdesidir. Bu önbelleğin tasarımına bağlı olsa da çoğu zaman ana belleğin boyutuyla ilişkilidir. Önbelleğin boyutu ise hızlı bellek çiplerinin pahalılığı yüzünden kısıtlanmıştır. İsabet oranı aynı zamanda çalıştırılan programın erişim şekline de bağlıdır. Çalıştırılan program, ardışık adreslere erişiyorsa isabet oranı artar. Rasgele adreslere erişiyor ise isabet oranı azalacaktır (Howe, 2006).

Dinamik bellek performanslarının gelişimi mikroişlemcilerin gelişimi ivmesini yakalayamamıştır. Mikroişlemci bir istekte bulunduğu zaman bu bilginin gelmesi için bir süre beklemektedir. Bu bekleyiş süresi statik RAM'lerden oluşan önbelleklerin kullanımıyla azaltılmaya çalışılmıştır. Ancak statik RAM kullanmak tüm bellek yapıları için uygun olmamaktadır. Statik RAM maliyet açısından külfetlidir. Bu sebeple ana belleğin yapısında dinamik RAM, önbelleklerde ise statik RAM kullanılmaya başlanmıştır.

2.2 Önbellek Mimarisi

Şekil 2.1‘de gösterildiği gibi önbellek işlemci ile ana bellek arasına yerleştirilmiştir. Küçük fakat yüksek hızlı bir bellektir (Intel, 2006).



Şekil 2.1 Örnek bir önbellek temelli bellek sistemi (Intel, 2006).

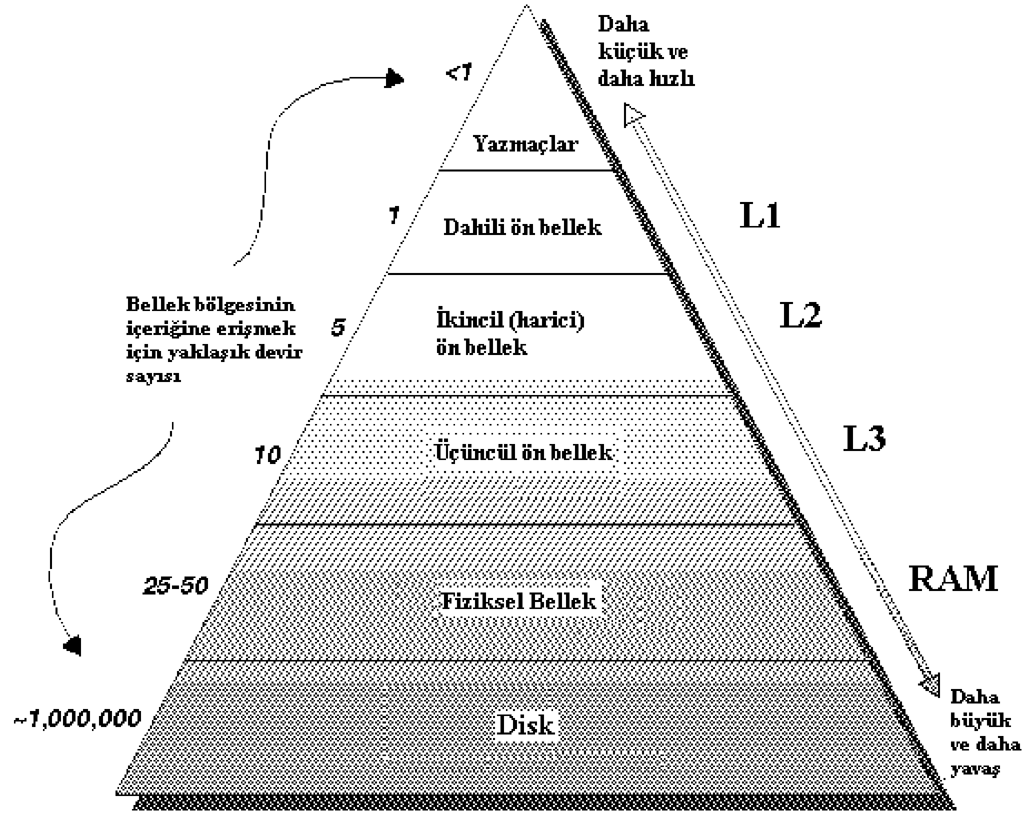
Sistem, işlemcinin ihtiyaç duyduğu veriyi önce ana bellekten önbelleğe kopyalar. Daha sonra önbellekten işlemci içindeki yazmaçlara kopyalar. Sonuçların depolanması da önce yazmaçlardan önbelleğe sonra önbellekten ana belleğe doğrudur (Sun, 2006).

Ana bellekten önbelleğe veri transferi genel olarak önbellek bloğu (cache block) veya önbellek satırı (cache line) denilen bellek birimleri sayesinde yapılır.

Yarı iletken teknolojisindeki gelişmelere göre, çoklu seviyeli önbellek geliştirmek mümkündür. Bu seviyelerin bazıları mikroişlemcinin bir bölümü olacaktır. Bu kavrama on-chip veya çip üzerinde denilmektedir. Diğer seviyeler ise mikroişlemcinin dışında harici olarak bulunacaktır. Bu önbellekler arasında ayrım yapmak için seviye kavramı kullanılmaktadır. En yüksek seviye işlemciye en yakın, en düşük seviye işlemciye en uzak olandır (Şekil 2.1). L1 işlemci ile aynı çip üzerindedir. Oysa L2 önbelleği mikroişlemci dışında bir seviyedir.

Şekil 2.1'de de görüldüğü gibi CPU ile mikroişlemci kavramsal olarak birbirinden farklı kavramlardır. CPU mikroişlemcinin çalıştırma bölümüdür, mikroişlemci ise CPU'yu da içine alan bütün çipe denmektedir (Sun, 2006).

Şekil 2.1'de bellek boyutu en küçük olan L1 önbelleği CPU ile tümleşik çip üzerinde ona en yakın şekilde tasarlanmıştır. CPU'dan uzaklaştıkça bellek boyutları artar ancak hız azalır. Bazı tasarımlarda 3 seviyeli önbellek mevcuttur. Bellek hiyerarşisi seviyeleri basamaklar halinde Şekil 2.2'de verilmiştir.



Şekil 2.2 Bellek hiyerarşisi (Surriel, 2006).

Önbellekler, komut önbelleği (I-Cache – Instruction Cache) ve veri önbelleği (D-Cache – Data Cache) olmak üzere iki türden oluşur. Komut önbelleği mikroişlemci tarafından istenilen veya ileride istenilme olasılığı olan komutları saklamak için, veri önbelleği ise verileri saklamak için kullanılır. Günümüzde hem komut hem veri önbelleğinin boyutları genellikle 16 - 128 KB arasında değişmektedir. Tez çalışmamızda L1 veri önbelleği odaklı çalışacağız.

Mikroişlemci tarafından ihtiyaç duyulan bellek bölgesinin önbellek içerisinde bulunmasına isabet (cache hit), ihtiyaç duyulan bellek bölgesinin önbellek içerisinde bulunamamasına ıska (cache miss) denir.

Önbellek boyutu ve organizasyon stratejileri isabet oranını (yani CPU saat frekanslarının yüzde kaçının önbelleğe isabet ettiğini) etkiler (Handy, 1998).

1960'larda IBM'deki araştırmacılar, bazı program parçacıklarının tekrarlı çalıştığını keşfettiler. Böylece lokalite kavramını ortaya attılar. Bu da bilgisayarın iş akışının iyileştirilmesine avantaj sağlaması için kullanıldı. Eğer tekrarlı kullanılan herhangi bir veri ya da komut kümesi küçük ama yüksek hızlı bir bellekte depolanırsa bekleme zamanının azaltılabileceği görüldü (Handy, 1998).

Uzaysal Lokalite (Spatial Locality): Çoğu program genelde bellekte ardışık duran veri veya komutları ardı ardına çalıştırır. Yani şu an bir X adresi çalıştırılıyor ise bir sonraki adımda X'ten bir sonraki adresteki veri/komut önbelleğe getirilip çalıştırılacak. Daha sonra ondan bir sonraki adresteki veri/komut önbelleğe getirilip çalıştırılacak. Böyle durumlarda önbellekte ıska olma durumunda ana bellekten ilgili adresteki veri/komut getirilirken bir sonraki adresteki de getirilir. Muhtemelen bir sonraki adımda o da isteneceği için tekrar ana belleğe erişime gerek kalmayacaktır (Handy, 1998). Bir bellek adresine erişildiyse ondan sonra onun yanındaki bellek adresine de erişilebilir. Bundan dolayı bellek seviyeleri arasındaki veri transferi bloklar halindedir (Howe, 2006).

Zamansal Lokalite (Temporal Locality): Bir adresteki veri veya komuta bir defa erişilmişse kısa bir süre sonra muhtemelen tekrar aynı veriye/komuta erişilebilecektir. Yani, işlemci yakın zamanda eriştiği bellek bölgesine kısa bir süre sonra tekrar ihtiyaç duyabilecektir. Önbellek bölgesinin erişildikten sonra belirli bir zaman kadar önbellekte saklanması zamansal lokaliteyi açıklar. Üzerinden zaman geçtikçe tekrar erişilme olasılığı düşmektedir (Handy, 1998).

Bu iki lokalitenin birlikte kullanılmasıyla programlarda daha sonra kullanılması muhtemel adreslerdeki veri ve komutların da önbelleğe getirilmesi ve önbellekte duran, bir defa erişilmiş adreslerin yakın zamanda tekrar erişilme olasılığından önbellekte tutulması çok ciddi faydalar sağlamaktadır. Bu da ana belleğe erişim ihtiyacını büyük bir oranda azaltmaktadır (Handy, 1998).

2.3 Önbelleğe Veri Yazma Yöntemleri

Tutarlılık ve hız önbelleğe yazmanın en önemli unsurlarındandır. Önbellek ve ana bellek arasındaki verinin tutarlılığı çoklu işlemcili sistemlerde ve paylaşımlı bellekli sistemlerde çok önemlidir (Intel, 2006). Yazma işlemi için 2 yaklaşım kullanılmaktadır:

Doğrudan Yazma (Write Through): Veri önbelleğe yazıldığı zaman ana belleğe de yazılır (Howe, 2006). Geliştirmesi daha kolay ve daha ekonomiktir. Performansı düşüktür, çünkü ana bellek trafiğini önemli ölçüde artırmaktadır. Önbelleğin isabet veya ıska olmasına bakılmayarak her yazma erişiminde hem önbelleğe hem de ana belleğe yazılır. Her yazma işleminde ana belleğe erişmek veri tutarlılığını sağlar ama işlemcinin hızını düşürür (Intel, 2006). Önbellek içinde yapılan her değişiklikte, değişiklikler hem önbelleğe hem de ana belleğe yapılır. Gerçekleştirmesi çok kolay ama performansı düşüren bir yöntemdir (Handy, 1998).

Geriye Yazma (Write Back): Veri sadece önbellek bloğundan çıkarılmaya zorlandığında ana belleğe yazılır (Howe, 2006). Önbellek tampon gibi çalışır. İşlemci önbelleğe yazmaya başlar, işi bitene kadar ve sistem veri yolu erişilebilir olana kadar yazar. Blok önbellekten çıkarıldığında, ilgili veri ana belleğe geri yazılır. Bu olay ana belleğe erişim sıklığını azaltır. Bu yöntemin en büyük özelliği performansı artırmasıdır (Intel, 2006).

Geriye yazma yöntemini kullanan önbelleklerin, verinin ana belleğe yazılmayıp önbelleğe yazıldığını izlemesi gerekmektedir. Değişen blokları belirlemek için geçerlilik bitleri ile bir de kirlenme (dirty) biti kullanılır. Böyle bir yöntemde her bir önbellek kaydı için fazladan 1 bit kullanılır. Önbellekteki veri değiştirildiğinde bu bit 0→1 yapılarak ilgili bloğun değiştirildiği kaydı tutulur. Önbelleğe ıska olduğunda ve bloğun yeri değiştirilecekse bu kirli bit kontrol edilir. Eğer veri değiştirilmiş yani kirli ise önce önbelleğe yazılmak zorundadır. Kirli bit kullanımı, veri tutarlılığını sağlasa da, erişim gecikmesini ve karmaşıklığı artırır (Intel, 2006). Her güncellemede ana belleğe gidilmez. Veriyi her güncellemede ana bellek yerine yüksek hızlı önbellekte güncellemesi ana bellek trafiğini azaltacaktır. Bunun 3 ana avantajı vardır; birincisi, önbelleğe yazma işlemi ana belleğe yazma işleminden çok daha hızlı gerçekleşir. İkincisi, bazı yazma işlemleri yığıt gibi döngülü yazmalar içerir. Önbelleğe böyle döngülü yazmaların gerçekleşme süresi ana bellekten kat ve kat daha hızlıdır. Üçüncüsü, işlemci ana belleğe nazaran sadece kısa bir süre çalışır. Tabii ki bu avantajlar beraberinde ek maliyet getirir (Handy, 1998).

Önbellek bloğunun yeri değiştirilmediği sürece ana belleğe yazılmaz. Kirli bit, blok güncellendiğinde ayarlanır. Kirli bit ayarlanmazsa bloğun içeriği yok edilir. Bloğun kirli olup olmadığını belirten kirli bit değeri 1 (true) olarak ayarlanmışsa ve bloğun üzerine başka blok yazılacaksa blok ana belleğe geri yazılır (Handy, 1998).

Eğer bütün yazma istekleri doğrudan yazma kullansaydı, önbelleğe bir yazma erişimi ana belleğe de yazma erişimi gerektirecekti. Bu da sistemin ana bellek hızına düşmesine sebep olacaktır. Doğrudan yazma geriye yazmadan daha kolaydır. Çünkü, doğrudan yazmada önbellekteki bir bloğun başka bir blokla yer değiştirmesi gerektiğinde zaten bir kopyası ana bellekte olduğundan yer değiştirebilir. Ama geriye yazmada ise bir blok başka bir blokla yer değiştireceği zaman yer değiştirilecek blok kirli ise yani içeriği değiştirilmiş ise yer değiştirmeden önce ana belleğe erişilip yazılmak zorundadır. Bununla birlikte geriye yazma daha verimlidir. Çünkü ana belleğe erişmeden önbellek içinde veri defalarca değiştirilebilir (Howe,2006).

Önbellek tamamen dolduğunda ve yine başka veri için bir önbellek bloğu istenildiğinde bir blok seçilir ve ana belleğe yazılır. Sonra yeni gelen satır da oraya yazılır. Blok seçimi ise bir yer değiştirme algoritması ile yapılır (Howe, 2006).

Önbellek ve ana bellekle ilişkili iki önemli kavram gecikme (latency) ve bant genişliğidir (bandwidth). Bir bellek bileşeninin gecikmesi, bir bellek bloğunun veya satırının getirilmesi veya transferi için geçen zaman olarak ölçülür. Bant genişliği ise bellek bileşeninin başka bir hız ölçüm değeridir. Bu değer bellek yapılarının geniş boyutlu verileri ne kadar hızlı iletip alabileceğini ölçer. Bant genişliği CPU’dan uzağa hareket ettikçe azalır (Sun, 2006).

2.4 Önbellek Organizasyonları

Önbelleğin ne kadar verimli olduğunu belirlemek için en önemli etken önbelleğin ana belleğe nasıl haritalandığıdır (Howe, 2006). Önbelleklerin belirli organizasyonları vardır. Organizasyon, önbellek bloklarının nasıl yerleştirileceğini gösterir (Sun, 2006).

Hangi önbellek bölgesinin hangi ana bellek bölgesinin kopyasını depolamak için kullanılacağını belirlemeye haritalama denmektedir (Handy, 1998).

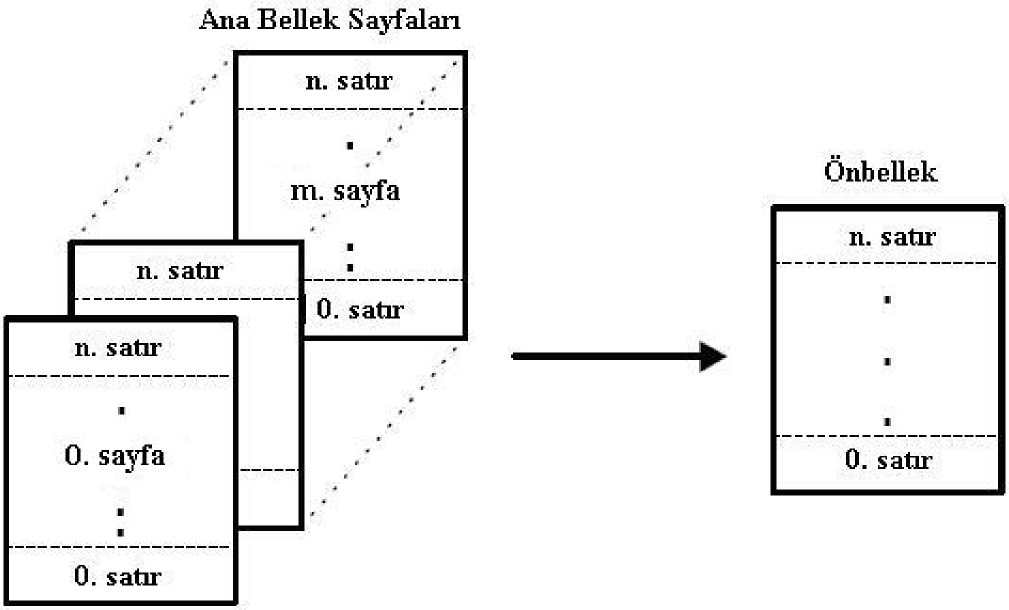
2.4.1 Doğrudan Haritalanmış Önbellek (Direct Mapped Cache)

Doğrudan haritalama basit ve verimli bir organizasyondur. Gelen bloğun bellek adresleri önbelleğin hangi bölgesinin kullanılacağını belirler ve kontrol eder (Sun, 2006).

Çoğu durumda bu tasarım iyi çalışır, ancak aday adresin bellek adresince kontrol edildiğinden, gerekli bilgi içeren önbellek bloğunun yer değiştirilmesi muhtemeldir. Bu da olumsuz bir tarafıdır (Sun, 2006).

Bu organizasyonda her bloğun gideceği sadece 1 yer olduğundan ıska olması durumunda hangi bloğun çıkarılacağı ile ilgili bir tercih yapmaya gerek yoktur. Olumsuz bir yönü bir program aynı önbellek bölgesine haritalanmış farklı adreslere erişmeye çalışırsa, her erişimde ıska olacaktır (Howe, 2006).

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



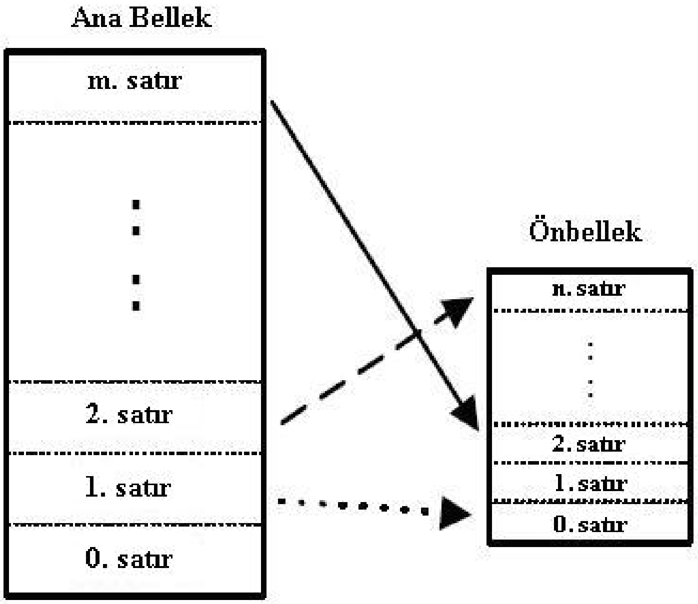
Şekil 2.3 Doğrudan haritalanmış önbellek.

Doğrudan haritalanmış önbelleğe 1-yollu küme çağrışımlı önbellek de denir. Şekil 2.3 bunu göstermektedir. Bu organizasyonda ana bellek, önbellek sayfalarına bölünür. Her bir sayfanın boyu önbelleğin boyutuna eşittir. Ana belleğin belirli satırlarını, belirli önbellek satırlarında depolayabilir. Örneğin, ana belleğin 0. satırı önbelleğin 0. satırında depolanmalıdır. Bu nedenle, önbellekte 0. sayfanın 0. satırı depolandığında eğer 1. sayfanın 0. satırı istenirse, 0. sayfanın 0. satırı ile 1. sayfanın 0. satırı yer değiştirir (Intel, 2006).

Doğrudan haritalanmış önbellek en az karmaşık olan yöntemdir. Bu yöntemde mevcut adresin sadece, 1 önbellek adresi ile kıyaslanması gerekmektedir. Daha az karmaşık olduğundan daha hesaplıdır. Olumsuz tarafı daha az esnek olmasıdır. Özellikle sayfalar arasında dallanmalarda performansı düşüktür (Intel, 2006).

2.4.2 Tam Çağrışımlı Önbellek (Fully Associative Cache)

Bu organizasyonda ana bellekteki bir satırın önbelleğin herhangi bir yerinde depolanmasına müsaade edilir. Tam çağrışımlı önbellek sayfa kullanmaz. Sadece satırları kullanır. Ana bellek ve önbellek eşit boyutlu satırlara bölünür (Intel, 2006).



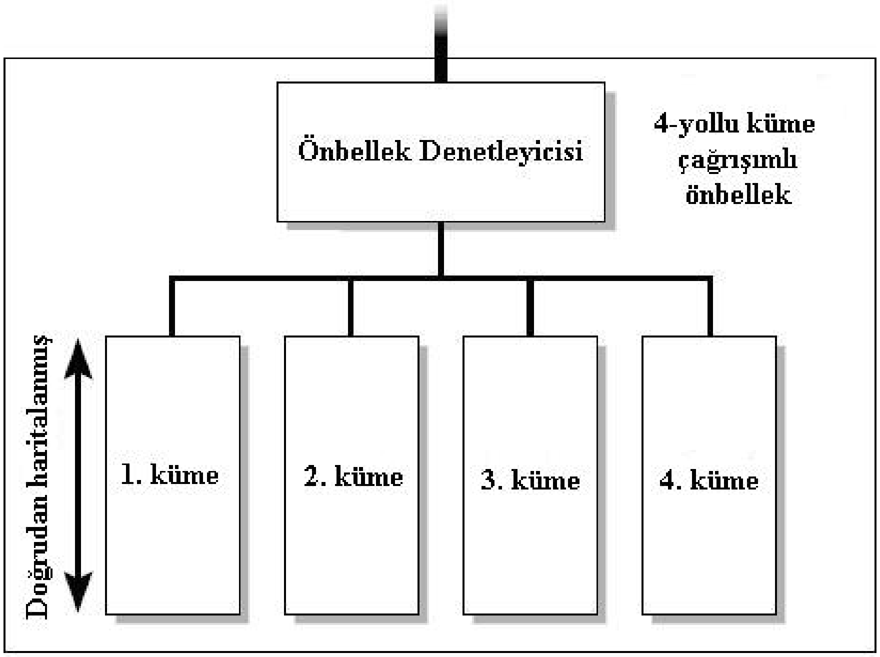
Şekil 2.4 Tam çağrışımlı önbellek.

Bu tasarımın olumsuz tarafı maliyetidir. Satırların kullanımını izlemek için ilave mantıksal devreler gereklidir. Önbelleğin boyutu büyüdükçe fiyatı artmaktadır. Bu nedenle bu teknolojiyi çok geniş veri önbelleklerine ölçeklemek zordur (Sun, 2006).

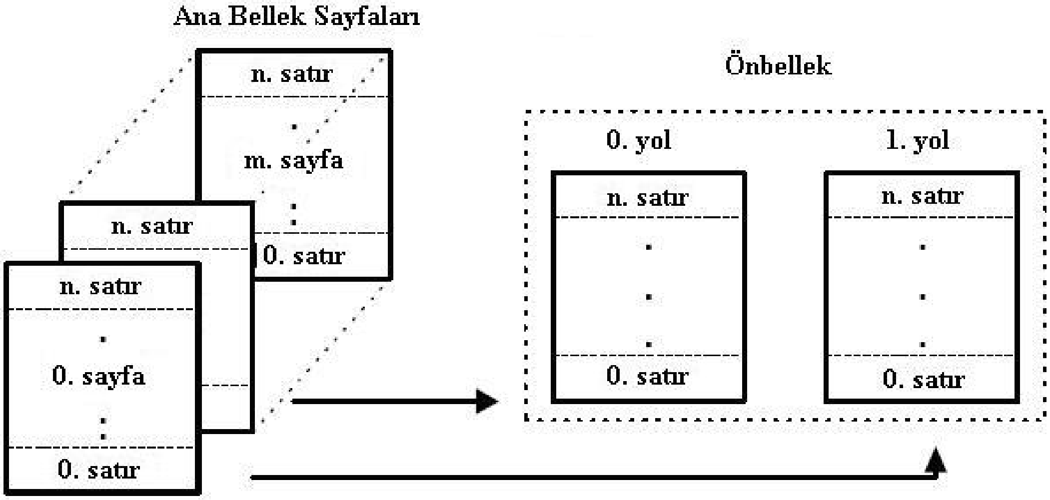
Belleğin herhangi bir bölgesinin önbelleğin herhangi bir yerinde depolanabileceğinden tam çağrışımlı önbellek organizasyonu en iyi performansı sağlar. Olumsuz yönü gerçekleştiriminin karmaşık olmasıdır. Karmaşık olmasının sebebi istenilen verinin önbellekte bulunup bulunmadığına karar vermenin getirdiği külfetin çok oluşudur. Mevcut adresin önbellekteki bütün adresler ile kıyaslanması gerekir. Bunun için de çok sayıda karşılaştırıcı gerekmektedir. Bunlar da önbelleğin karmaşıklığını ve fiyatını artırmaktadır. Bu nedenle bu tipteki önbellekler genellikle 4K'dan daha az, küçük önbelleklerde kullanılır (Intel, 2006).

2.4.3 Küme Çağrışımlı Önbellek (Set Associative Cache)

Küme çağrışımlı önbellek tasarımında birden fazla doğrudan haritalanmış önbellek kullanılır. Önbelleğin kümelerden (set) oluştuğu kabul edilir. Gelen isteğe göre önbellek denetleyicisi hangi satırın içine gideceğine karar verir. Küme içinde doğrudan haritalanmış şema kullanılır. Adını da doğrudan haritalanmış önbellek sayısına göre alır. Örneğin Şekil 2.5'teki tasarımda 4 tane doğrudan haritalanmış önbellek kullanıldığından bu tasarıma "4-yollu küme çağrışımlı önbellek" denir (Sun, 2006).



Şekil 2.5 4-yollu küme çağrışımlı tasarım (Sun, 2006).



Şekil 2.6 2-yollu küme çağrışımlı önbellek.

Küme çağrışımlı önbellek şeması tam çağrışımlı ile doğrudan haritalanmış önbellek şemalarının karışımıdır. Küme çağrışımlı şema, SRAM'i eşit bölümlere bölmektedir ve bunlara önbellek yolu (cache way) denir. Önbellek sayfa boyutu önbellek yol değerine eşit olur. Her bir önbellek yolu küçük bir doğrudan haritalanmış önbellek gibi davranır. Bu şemada belleğin iki satırı herhangi bir anda depolanabilir. Bu da önbellek satırlarının üzerine yazılma sayısını azaltır (Intel, 2006).

Bu yöntem, tam çağrışımlı önbellekten daha az karmaşıktır. Çünkü karşılaştırıcı sayısı önbellek yolu sayısına eşittir. 2-yollu küme çağrışımlı önbellek sadece 2 karşılaştırıcıya gereksinim duyar. Bu da bu organizasyonu tam çağrışımlı önbellekten daha ucuz yapmaktadır (Intel, 2006). Çizelge 2.1, farklı önbellek haritalama teknikleri ve onların performans kıyaslamalarını vermektedir.

Çizelge 2.1 Farklı önbellek haritalama teknikleri ve onların performans kıyaslamaları.

Önbellek Tipi	**İsabet Oranı**	**Arama Hızı**
Doğrudan Haritalanmış Önbellek	İyi	En iyi
Tam Çağrışımlı Önbellek	En iyi	Orta
N Yollu Küme Çağrışımlı Önbellek, N>1	Çok iyi, N arttıkça iyileşir	İyi, N arttıkça kötüleşir

Günümüzde doğrudan haritalanmış ve küme çağrışımlı önbellekler daha yaygın olarak kullanılmaktadır. Doğrudan haritalanmış önbellek tasarımı genellikle daha çok ana kart üzerinde L2 önbellekler için, yüksek performanslı küme çağrışımlı önbellek tasarımı ise işlemci ile aynı çip üzerindeki küçük L1 önbellekler için kullanılmaktadır (Howe, 2006).

2.5 Yer Değiştirme Algoritmaları

Başka bir tasarım parametresi ise önbellek satırları üzerinde seçme işlemi yapacak olan algoritmalardır (Sun, 2006). Önbellek dolduğunda ve ana bellekten yeni bir blok getirildiğinde bu bloğun bir blok ile yer değiştirmesi gerekmektedir. Hangi bloğun seçileceğini bu algoritmalar belirler (Handy, 1998).

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

Bu algoritmada önbellek içinde erişimler gözlenir ve bunlar zamana, erişim sayısına göre sıralanır. Son zamanlarda en az erişilmiş olan bloğu kurban olarak seçer (Handy, 1998).

2.5.2 Rasgele Algoritması

Bunun herhangi bir temeli, dayanağı yoktur. Önbellek içinde yer değiştirmek için rasgele bir blok seçer (Handy, 1998).

2.5.3 İlk Giren İlk Çıkar Algoritması (First In First Out - FIFO)

Bu yöntemde, önbelleğe gelen her bloğun bir zaman mührü (time stamp) vardır. Bir yer değiştirme için blok seçileceği zaman, zaman mührü en eski olan yani önbelleğe en erken getirilmiş olan blok seçilir (Handy, 1998).

BÖLÜM 3
ÖNCEDEN GETİRME İŞLEMİ

3.1 Önceden Getirme İşlemi Nedir?

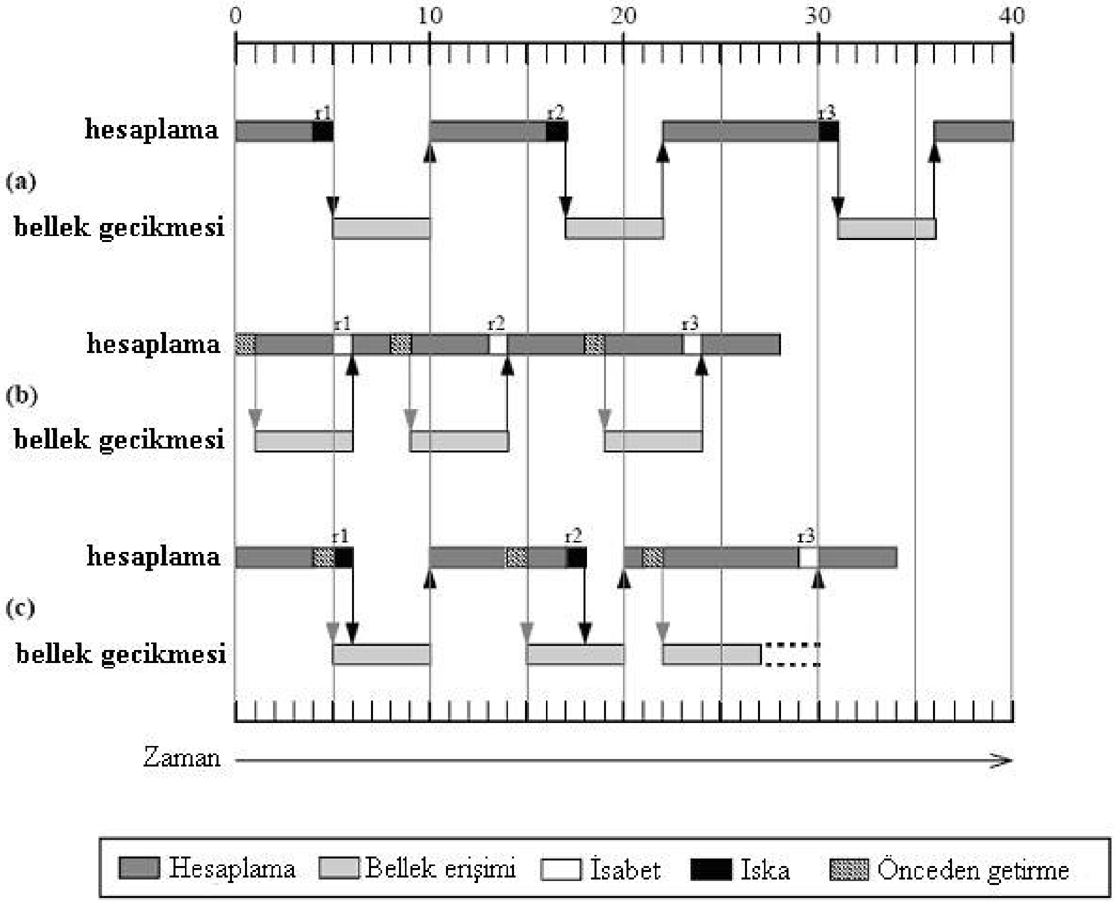
Önceden getirme modern mikroişlemcilerde verileri veya komutları ihtiyaç duyulmadan önce işlemciye daha yakın konuma getirerek uzun bellek gecikmelerini gizlemek için kullanılan yaygın bir tekniktir.

Mikroişlemciler ve DRAM performansı arasında açılan fark ana bellek erişim gecikmesini ortadan kaldırmak veya azaltmak için yeni tekniklerin geliştirilmesini gerekli kılmıştır. En sık kullanılan veriler için bu gecikmenin etkin şekilde azaltılması büyük önbellek hiyerarşilerinde sağlanmışsa da hala birçok program çalışma zamanlarının yarıdan fazlasını bellek isteklerini beklemekle harcamaktadırlar. Veri önceden getirmesi, erişim gecikmesini ortadan kaldırmak için teklif edilmiş bir tekniktir. Verinin bellekten alınıp getirilmesinin (memory fetch) başlatılması için bir önbellek ıskası beklemektense veri önceden getirmesi böyle ıskaları önceden sezer ve gerçek bellek adresiyle bellek sistemine verinin getirilmesi (fetch) çağrısı yapar. Verimli olması açısından, önceden getirme işlemi uygulanırken önceden getirmeler zamanında, yararlı olacak şekilde ve en az masrafla yapılmalıdır. Önbellek kirlenmesi ve artan bellek band genişliği (bandwidth) ihtiyaçları da dikkate alınmalıdır. Bunca kıstasa rağmen, önceden getirmenin bellek erişim gecikmelerinde sağladığı düşüş programın toplam çalışma süresini oldukça geliştirmek için büyük bir potansiyele sahiptir. Çeşitli önceden getirme stratejileri mevcuttur ve henüz tek bir stratejinin en iyi performansı sağladığı kabul görmemiştir (VanderWiel ve Lilja, 2000).

İlk başlarda önbellek hiyerarşilerinin kullanımı, başlı başına, bellek beklemelerinden kaynaklanan gecikme süresini azaltmada ana unsurdu (Smith, 1982). Günümüzde önbelleklerde kullanılan statik RAM (SRAM) bellekler işlemcinin bellek ihtiyaçlarına cevap vermeye çalışmaktadır; ancak bir bellek teknolojisi olarak çok pahalı olmaya devam etmektedir. Bellek hiyerarşilerinin kullanımının adresleme yapılarında yüksek lokalite gösteren programların ortalama bellek erişim beklemelerini azaltmada etkin olmasının kesin görülmesine rağmen bilimsel ve diğer benzeri veri yoğunluklu programların çalışma zamanlarının yarıdan fazlasını bellek isteklerini beklemekle geçirdiği görülmektedir (Mowry ve diğ., 1992). Bu tarz uygulamaların temelini oluşturan büyük matris işlemlerinde tipik olarak veri tekrar kullanımı az görüldüğünden önbellekleme stratejilerinde güçlükler ortaya çıkabilmektedir.

Bu uygulamaların önbellekten gerektiğince faydalanamamalarının kısmen sebebi çoğu önbellekte “istek üzerine” (on demand) bellek erişimi (memory fetch) çağrılması tutumudur. Bu tutum, ana bellekten önbelleğe veriyi yalnız işlemci onun için bir talepte bulunduktan sonra ve onu önbellekte bulamadığında alıp getirir. Şekil 3.1’de bu durum açıklanmaktadır. Şekilde r1, r2 ve r3 bellek erişim taleplerine karşılık gelen veri blokları önbellek hiyerarşisinde bulunmamaktadır ve bu nedenle ana bellekten getirilmelidir (fetch). Temel yapıda sıralı (in-order) çalıştırma birimine sahip olduğunu varsayarsak işlemci uygun önbellek bloğunun getirilmesi için beklemede kalacaktır. Veri ana bellekten önbelleğe oradan da işlemcinin kullanımına hazır edildiğinde hesaplama süreci tekrar devam etmektedir. Veriyi bu şekilde getirme işlemi önbellek bloğuna ilk erişim için önbellekte sadece daha önceden erişilmiş veriler tutulduğundan daima bir önbellek ıskasıyla sonuçlanacaktır. Bu tarz önbellek ıskalarına soğuk başlangıç (cold start) veya zorunlu ıskalar (compulsory misses) denir. Yine, eğer erişilen veri büyük bir dizinin bir kısmıysa ve kısım kullanıldıktan sonra dizinin yeni elemanlarına yer açmak için bu kısımda yer değiştirme yapıldıysa, daha sonra bu kısımdan bir veri için talep geldiğinde oluşan ıskaya kapasite ıskası (capacity miss) denir (VanderWiel ve Lilja, 2000).

Önceden getirme işlemi ile önbellek ıskalarının azaltılması mümkündür. Iska beklemeden ıskaların olacağının tahmini ile ilgili adresler için önceden çağırma işlemi başlatılır. İdeal olan, veriye ihtiyaç duyulduğu anda önceden getirilmiş olması ve işlemcinin hiç beklemeden veriye erişmiş olmasıdır. Bu durumda programın çalışma devir sayısının (execution cycle) azaltılması söz konusu olacaktır.



Şekil 3.1 Üç duruma göre çalıştırma diyagramları: (a) önceden getirmesiz, (b) mükemmel önceden getirmeli ve (c) düşük verimli önceden getirmeli (VanderWiel ve Lilja, 2000).

Bir veri önceden getirmesi için işlemci tarafından “fetch” talimatı verilmektedir. Bu talimat önbelleğe getirilecek olan adres bilgisini içerir. Şekil 3.1(c)’de r3 isteği olmadan epey önce bir önceden getirme başlatılmış ve sona ermiş; ancak bu durumda da önceden getirilmiş verinin yer değiştirmeye maruz kalması söz konusudur. Bu tarz yapılan önceden getirmeler yararsız olarak nitelenir. Vaktinden önce getirilen verinin işlemci tarafından o anda kullanılabilecek veriyi yerinden etmesi, onun yerine kendini yüklemesi de mümkündür. Bu durum önbellek kirliliği (cache pollution) olarak bilinir (Casmira ve Kaeli, 1995).

Şekil 3.1(a)’da üç bellek erişim isteğinin ilk 31 devirde, Şekil 3.1(b)’de ise ilk 19 devirde gerçekleştiğine dikkat edilmelidir. Veri yolu band genişliğinin etkin kullanımı özellikle çok işlemcili sistemlerde kritik öneme haizdir.

3.2 Önceden Getirme İşleminin Tarihçesi

VanderWiel ve Lilja (2000)‘ya göre önceden getirme işlemi 1960’ların ortalarından beri bir şekilde yapılıyordu. Önbellek tasarımı konusundaki ilk çalışmalarda (Anacker ve Wang, 1967) ana bellekten önbelleğe sözcüklerin toplu halde getirilmesinin yararlarını gözlemlediler. Bu durumda toplu getirilen sözcükler sayesinde bellek adreslerinin uzaysal lokalitesi avantajından yararlanılması umuluyordu. Ayrı önbellek bloklarının donanım temelli önceden getirmesi IBM 370/168 ve Amdahl 470V’de gerçekleştirildi (Smith, 1978). Yazılıma dayalı teknikler daha sonraları ortaya atılmıştır. Bu konuda kullanışlılığından tereddüt etse de dolaylı olarak görüşlerini ilk belirten (Smith, 1982) olmuştur. Sonrasında (Porterfield, 1989) “cache load instruction” fikrini ortaya atmıştır. Yine aynı tarihlerde Motorola 88110 makinesinde “touch load” komutunu tanıtmış ve Intel de i486’da “dummy read” işlemlerinin kullanımını önermiştir (Fu ve diğ., 1989). Günümüzde neredeyse tüm mikroişlemci ISA’ları işlemcinin bellek hiyerarşisine veriyi getirmek için tasarlanmış açık talimatlar içermektedir. Önceden getirme işlemi aslında sadece ana bellekten getirmeyle alakalı değildir; önbellek hiyerarşisinde nesnelerin üst seviyelere taşınmasını da bu tarz işlemlerden sayabiliriz.

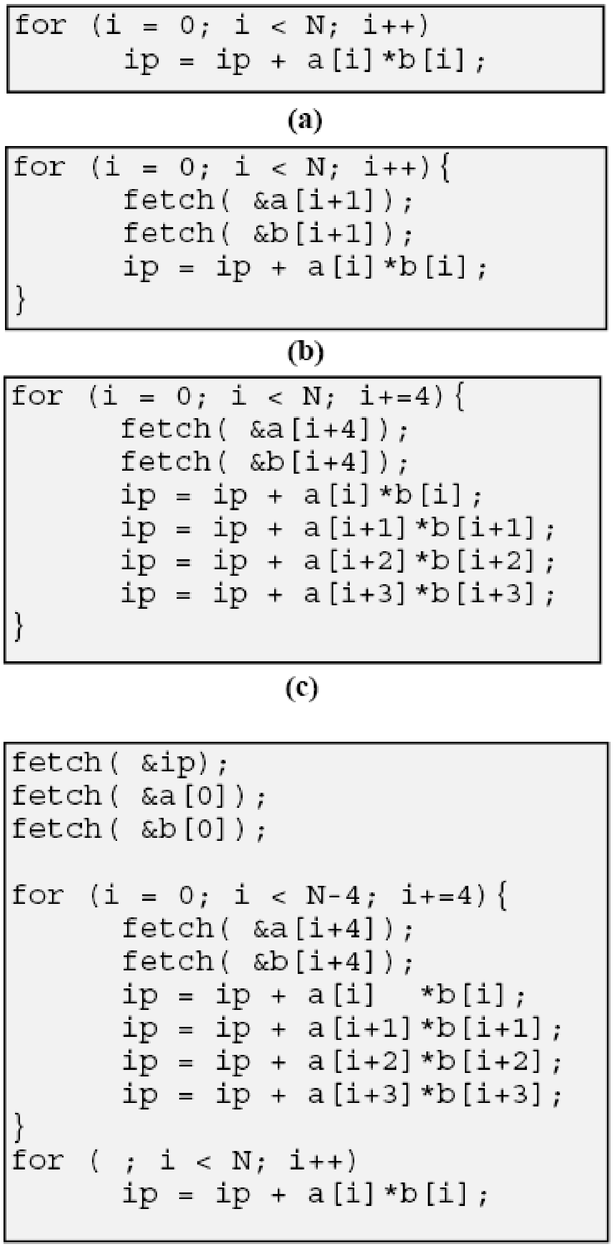
3.3 Önceden Getirme İşleminin Türleri

Önceden getirmenin bellek erişim gecikmelerinde sağladığı düşüş programın toplam çalışma süresini oldukça geliştirmek için büyük bir potansiyele sahiptir. Önceden getirme işlemi yazılım tabanlı ve donanım tabanlı olmak üzere iki ana kategoriye ayrılmaktadır.

3.3.1 Yazılım Tabanlı Önceden Getirme İşlemi (Software Prefetching)

Yazılım tabanlı önceden getirmenin gerçekleştirilmesi için gereken donanımsal gereksinimler diğer önceden getirme stratejilerine göre çok daha azdır. Bu yaklaşımın en karmaşık, zor yanı hedef uygulamada önceden alıp getirme komutlarının nereye yerleştirileceğidir. Pratikte Şekil 3.1(b)’nin her zaman doğru gerçekleştirilmesi elbette zordur, hatta imkansızdır. Bu komutlar en iyileştirme sürecinde programcı veya derleyici tarafından kaynak koda eklenir. Mowry ve Gupta, (1991) sadece birkaç önceden getirme direktifinin eklenmesinin bile önemli performans geliştirmesi sağlayabileceğini çalışmalarında göstermiştir. Ancak programlama gayreti en aza indirilmek isteniyorsa veya birçok önceden getirme fırsatı mevcutsa derleyici desteğine ihtiyaç olabilir.

Önceden getirme çoğunlukla büyük dizilerin hesaplama işlemleri için kurulan döngülerde kullanılır. Bu döngülerde erişilmek istenen adreslerin tahmin edilebilir olması dolayısıyla bir sonraki döngü adımında ihtiyaç duyulacak verilerin getirilmesi o anki döngüde tetiklenebilir.



```
for (i = 0; i < N; i++)
      ip = ip + a[i]*b[i];
```

(a)

```
for (i = 0; i < N; i++){
      fetch( &a[i+1]);
      fetch( &b[i+1]);
      ip = ip + a[i]*b[i];
}
```

(b)

```
for (i = 0; i < N; i+=4){
      fetch( &a[i+4]);
      fetch( &b[i+4]);
      ip = ip + a[i]*b[i];
      ip = ip + a[i+1]*b[i+1];
      ip = ip + a[i+2]*b[i+2];
      ip = ip + a[i+3]*b[i+3];
}
```

(c)

```
fetch( &ip);
fetch( &a[0]);
fetch( &b[0]);

for (i = 0; i < N-4; i+=4){
      fetch( &a[i+4]);
      fetch( &b[i+4]);
      ip = ip + a[i]   *b[i];
      ip = ip + a[i+1]*b[i+1];
      ip = ip + a[i+2]*b[i+2];
      ip = ip + a[i+3]*b[i+3];
}
for ( ; i < N; i++)
      ip = ip + a[i]*b[i];
```

(d)

Şekil 3.2 (a) Önceden getirmesiz, (b) basit önceden getirmeli, (c) adım atlamalı önceden getirmeli ve (d) yazılım iş hattı (software pipelining) teknikleri ile döngüsel hesaplamalar (VanderWiel ve Lilja, 2000).

Döngü temelli önceden getirme örneği olarak Şekil 3.2(a)'ya bakalım. Bu şekilde a ve b dizileri kullanılarak N adım boyunca temel bir toplama işlemi yapılıyor. Önbellek bloğunun 4 sözcüklük olduğunu varsayarsak her dördüncü iterasyonda bir önbellek ıskası olacaktır. Bu önbellek ıskalarından sakınmak için Şekil 3.2(b)'de gösterilen önceden getirme direktiflerini ekliyoruz. Kodların, derleyici tarafından oluşturulacak olan makine dili kodlarının kaynak kodu ifadesi olduğuna dikkat edelim.

Önceden getirmeye bu temel yaklaşımın birkaç sorunu vardır. Birincisi, döngünün her adımında önceden getirmeye gerek yoktur; çünkü zaten her bir getirme (fetch) komutu önbelleğe 4 sözcük (bir önbellek bloğu) getirmektedir. Bu nedenle Şekil 3.2(b)'deki gibi önceden getirme çağrıları gereksizdir veya performansı düşürür. a ve b'nin önbellek bloğunda hizalı (cache block aligned) yerleştiğini varsayarsak önceden getirme sadece her 4 adımda bir yapılmalıdır. Bunun için döngüde i mod 4 = 0 koşulu sağlandığında veri getirme çağrısı başlatılabilir; ancak bu koşulun işletilmesi önceden getirmeden yapılacak kazancı zayıflatacaktır.

Başka bir yol da fetch komutunun bir kerede getirdiği sözcük sayısı (4) kadar artışla döngünün çalıştırılmasıdır (Şekil 3.2(c)). Bu durumda da döngünün ilk adımında önbellek ıskalarıyla karşılaşılacaktır. Ayrıca son adımda gereksiz önceden getirmeler dizi sınırlarını aşarak tetiklenmektedir. Bu gibi sorunlar yazılım iş hattı (software pipelining) denilen teknikler ile giderilebilir (Şekil 3.2(d)). Bu şekilde yaklaşımlar elbette ki karmaşık derleyici algoritmalarında çok daha fazla geliştirilmiştir. VanderWiel ve Lilja (2000) 'ya göre Bernstein ve diğ. (1995) PowerPC 601 temelli bir sistem üzerinde 12 bilimsel testin çalışma zamanlarını önceden getirme kullanmadan ve kullanarak ölçmüştür. Burada önceden getirme çalışma zamanlarını ortalama %12 iyileştirmiştir. Ancak bu örneklere rağmen yazılım önceden getirmesinde işlemcinin fazladan getirilecek adreslerle de uğraşması, bu adreslerin kod içerisinde doğrudan açık çağrı olarak yer alması, yazılımdaki kod şişkinliği (ki özellikle komut önbelleği) performansı düşürebilmektedir. Yazılım tabanlı önceden getirme statik olarak yapıldığı için önceden getirme çağrısı yapılan bloğun daha önceden alınmış olup olmadığını bilmeden çağrı başlatır.

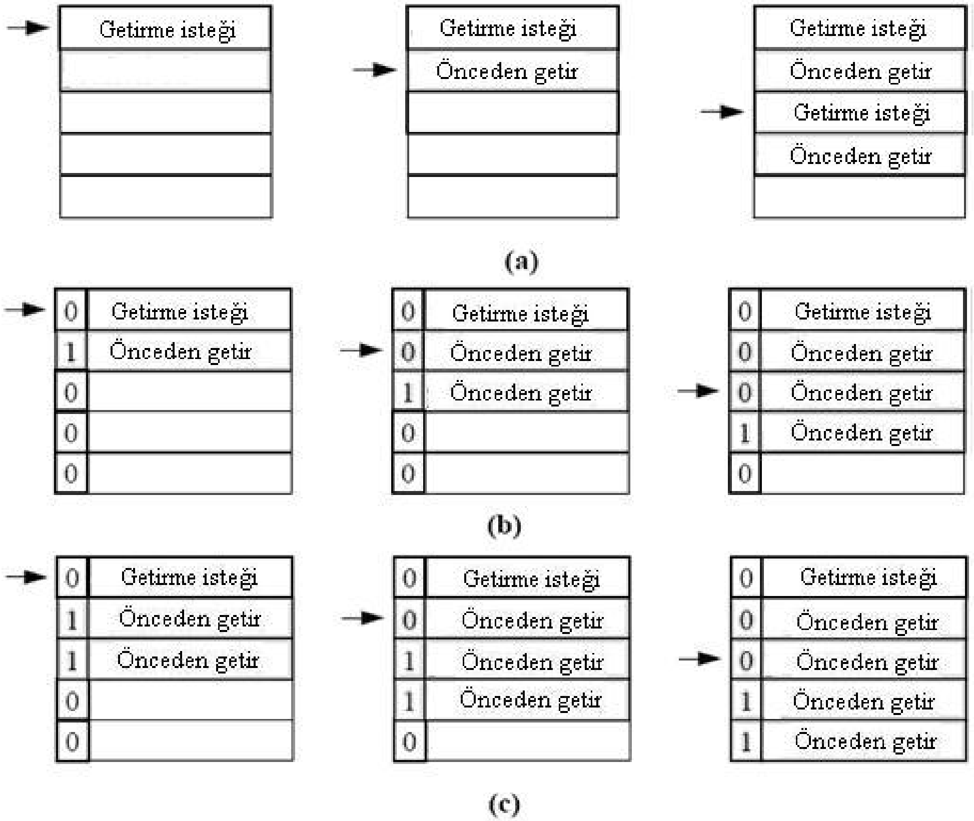
3.3.2 Donanım Tabanlı Önceden Getirme İşlemi (Hardware Prefetching)

Donanım tabanlı önceden getirme planları programcıya veya derleyicide iyileştirmelere ihtiyaç duymadan mikroişlemci mimarisine önceden getirme yetenekleri eklenerek geliştirilebilir. Çalışan programlarda herhangi bir değişikliğe gitmeye gerek yoktur. Dolayısıyla komut masrafı tamamen ortadan kaybolur. Ayrıca donanım tabanlı önceden getirme işlemi çalışma zamanı bilgisini dinamik kullanabildiği için potansiyel olarak önceden getirmeyi daha verimli yapacak kapasiteye sahiptir.

Çoğu önceden getirme planı verilerin ana bellekten işlemciye yakın belleklere önbellek blok birimleri halinde getirilmesi esas alınarak tasarlanır. Aslında çoklu sözcüklerden oluşan önbellek bloklarının getirilmesi de başlı başına önceden getirme sayılabilir. Çünkü CPU yazmacı sayesinde sözcük bazlı çalışır.

3.3.2.1 Bir Sonraki Bloğu Önceden Getirme (Next-Line Prefetching)

En basit sıralı önceden getirme planları sonraki bir bloğu önceden getirme (one block lookahead - OBL) yaklaşımına dayanan çeşitlemelerdir. Bir sonraki bloğu önceden getirme işlemi bir b bloğuna erişim olduğunda b+1 bloğunun önceden getirilmesi işlemini başlatır. Burada b bloğuna erişim türüne göre ıska olduğunda önceden getirme (prefetch-on-miss) ve etiketlemeli önceden getirme (tagged prefetch) olmak üzere iki farklı yaklaşım mevcuttur (Smith, 1982). Iska olduğunda önceden getirme yaklaşımında b bloğuna erişim hamlesi ıska ile sonuçlanırsa b+1 bloğu için önceden getirme işlemi başlatılır. Ancak b+1 zaten önbellekte bulunuyor ise erişim başlatmaya gerek yoktur (Şekil 3.3).



Şekil 3.3 Bir sonraki bloğu önceden getirme işleminin 3 yöntemi: a) Iska olduğunda önceden getirme, b) Etiketlemeli önceden getirme, c) Sonraki iki bloğu tetiklemeli önceden getirme (K=2). (VanderWiel, 1998).

Etiketlemeli önceden getirme yaklaşımında her önbellek bloğuna sonraki bloğun ne zaman önceden alınıp getirilmesi gerektiğini belirten bir işaret biti eklenmiştir (Şekil 3.3(b)). Önceden getirilen bloğun işaret biti 0 yapılır. Erişilen bloğun işaret biti 0 ise bir sonraki sıralı bloğun önceden getirilme işlemi başlatılır ve esas bloğun işaret biti 1 yapılır.

Etiketlemeli önceden getirmede b+1 bloğu için erişimin tetiklenmesi için b'ye iki farklı durumda erişim olması gerekmektedir. Birincisi, b'ye erişildiğinde eğer ıska olmuşsa; ikincisi, eğer b önbelleğe önceden getirme ile gelmiş ise ve ona ilk defa erişiliyor ise. Yapılan bu tez çalışmasında donanım tabanlı önceden getirme tekniği olan etiketlemeli önceden getirme yöntemi kullanılacaktır.

HP PA7200 bir sonraki bloğun önceden getirilmesi donanım temelli yöntemini kullanan modern mikroişlemcilere bir örnektir (Chan, 1996).

(Smith, 1982), etiketlemeli önceden getirme yönteminin hem komut hem de veri önbelleği için ıska oranını %50 ila %90 arasında azalttığını bulmuştur.

3.3.2.2 Markov Yöntemi

Önceden getirmede kullanılan yöntemlerden bir diğeri olan Markov yönteminde gelecekte ana bellekte erişilebilecek veya tekrarlamakta olan adres dizilerini tutmak amacıyla Markov tablosu denilen bir tablo kullanılmaktadır. Markov yöntemi önbelleğe alınan adresler arasındaki ilişkileri kullanmaktadır. Düzenli bellek erişim biçimi gösteren bilimsel çalışmalar için verimli olmasına rağmen, tekrarlanmayan biçimde adres erişimlerine sahip uygulamalarda Markov yöntemi ile gelecekte istenebilecek bellek adreslerinin tahmin edilmesi mümkün değildir (Kim, 2004).

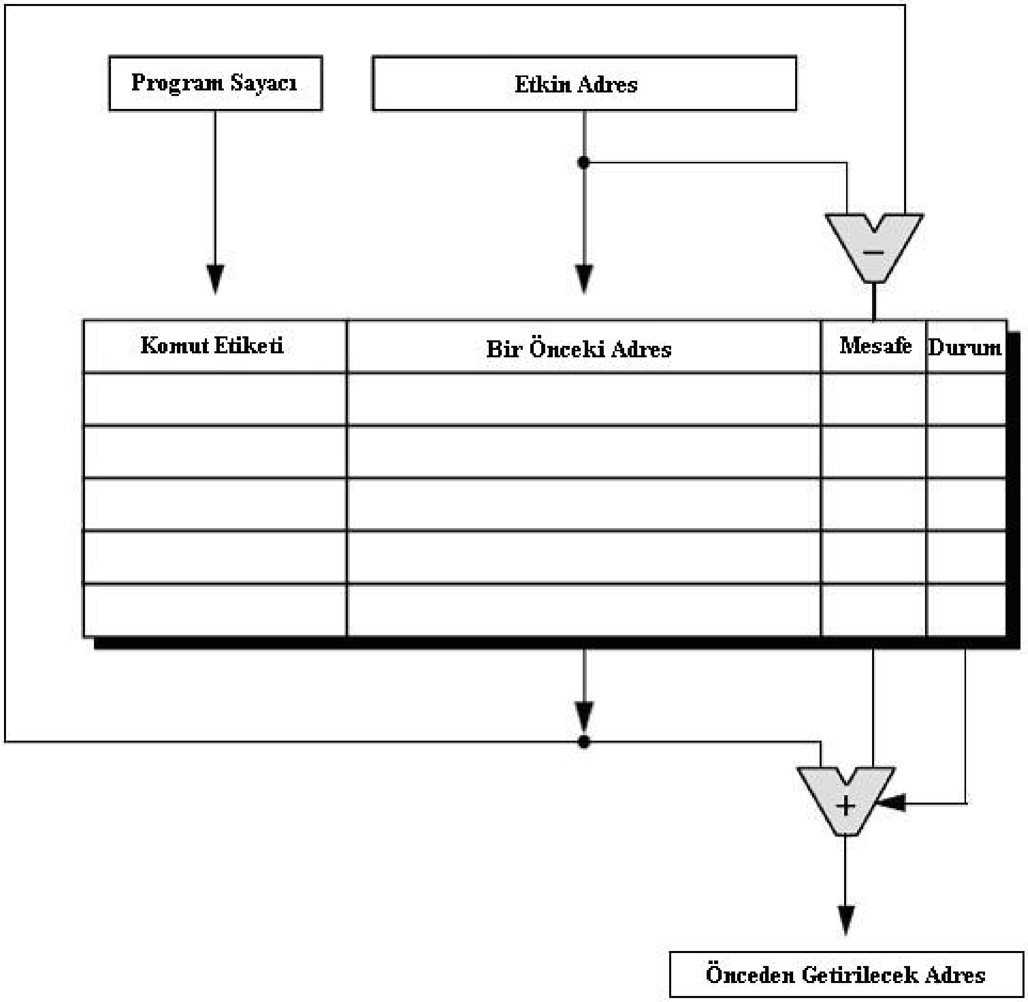
VanderWiel ve Lilja (2000) 'ya göre, Joseph ve Grunwald (1997) donanım temelli veri önceden getirme mekanizması uygulamak için bir Markov tahmin edicisi (Markov predictor) üzerinde çalışmışlardır. Bu mekanizma, oluşan ıskaların önceki ıska adresleriyle benzerlik göstererek tekrarlanıp tekrarlanmadığını anlamak için önbellek ıska ardışıklıklarını Alexander ve Kedem (1996) 'in kullandığı donanım tablosuna benzer bir tabloya dinamik olarak kaydederek tahmin etmeye çalışır. O anki önbellek ıskasıyla eşleşen adres tabloda bulunduğu anda sıradaki tahmin edilen ıska adresi için önceden getirme çağrısı başlatılır.

3.3.2.3 Akış Tamponu (Stream Buffer) Yöntemi

VanderWiel ve Lilja (2000) 'ya göre, Jouppi (1990) donanım temelli önceden getirme tekniği olarak akış tamponu yöntemini önermiştir. Akış tamponu FIFO algoritmasına göre çalışan geçici bellektir. Akış tamponu yönteminde ıska olduğunda bulunamayan blok önbelleğe ve sonraki K tane sıralı blok da akış tamponuna getirilir. Akış tamponunun başındaki kayıt referans edildiğinde, o blok önbelleğe getirilir, kalan K-1 blok kuyruğun başına doğru ötelenir ve kuyruğun son kısmına yeni bir blok önceden getirilir. Önceden getirilen bloklar doğrudan önbelleğe yerleştirilmediği için akış tamponu yöntemi önbellek kirliliğine sebebiyet vermez. Ancak, önbellekte bulunmayan bloğun akış tamponunun baş kısmında da bulunmaması durumunda tamponun boşaltılması işlemi gerçekleştirilir. Akış tamponlarının performans kazancı sağlaması için önceden getirilen bloklara tampona getirilme sıralarına göre erişilmelidir.

3.3.2.4 Mesafeye Dayalı Önceden Getirme (Stride-based Prefetching)

İşlemcinin talep ettiği adres yapıları eğer ardışık olmayan bellek bloklarında belirli mesafelere göre bulunuyorsa sıralı önceden getirme gereksiz önceden getirmelere ve dolayısıyla verimsizliğe neden olur. Bu gibi nedenlerle yükleme (load) veya depolama (store) komutlarının kullandığı ardışık adresleri karşılaştırarak mesafe (stride) denilen aradaki uzaklık bulunur. Eğer önceden getirme donanımı özel yükleme veya depolama komutu için tahmin edilebilir bir bellek adresleme yapısı sezerse o zaman otomatik olarak bir önceden getirme tetikler. Bu tahmin edilebilirlik mesafelerin hesaplanmasından yola çıkılarak karar verilir. Mesafeye dayalı önceden getirme yaklaşımında bir bellek komutu tarafından kullanılan bir önceki adres, eğer mevcutsa son sezilen mesafe ile birlikte kayıt altında tutulması gerekmektedir. Her bir bellek erişim adresinin kaydedilmesi imkansız olacağından Referans Tahmin Tablosu (Reference Prediction Table - RPT) denilen ayrı bir önbellek mekanizması geliştirilmiş ve buraya son zamanlarda en çok kullanılan bellek adresleri kaydedilmiştir. RPT tablosundaki bir kayıt bellek komutunun adresini, komut tarafından erişilen bir önceki adresi, mesafe değerini ve kaydın o anki durumunu içeren bilgileri tutmaktadır. Şekil 3.4'te RPT tablosunun yapısı verilmiştir.



Şekil 3.4 Referans Tahmin Tablosunun Yapısı (VanderWiel ve Lilja, 2000).

RPT tablosu, CPU'nun program sayacı (Program Counter - PC) değerine göre indekslenmiştir. İşlemci m_i bellek komutunu ilk kez çalıştırdığında komutu RPT tablosuna durumu başlangıç (initial) olarak kaydeder. Bu, RPT'nin bu komut için henüz önceden getirme başlatmadığını gösterir. Eğer m_i komutu RPT tablosundaki kaydı silinmeden önce tekrar işletilirse RPT, komutun o an eriştiği adresten tabloda daha önce kayıtlı olan önceki eriştiği adresi çıkararak mesafe değerini hesaplar.

3.3.3 Yazılım ve Donanım Tabanlı Önceden Getirme İşlemleri Arasındaki Farklar

Yazılıma dayalı tekniklerde ek donanım maliyetleri çok daha azdır. Dolayısıyla donanıma dayalı tekniklerin gerçekleşmesi için daha fazla kaynak gerekmektedir. Yazılıma dayalı önceden getirmede kullanıcı programında uygun yerlere veriyi alıp getirme komutları yerleştirmek için derleme zamanı bilgisi kullanılmaktadır (VanderWiel ve Lilja, 2000). Bu durumda kaynak kod boyutu artmakta ve yeniden derleme işlemi ile fazladan zaman külfetini beraberinde getirmektedir. Önceden getirme komutları işlemci tarafından yapılan işin miktarını artırabilmektedir. Donanım önceden getirmesi ise komut artışına neden olmamakla beraber yazılım önceden getirmesine nazaran daha fazla gereksiz önceden getirme üretir. Bu fazlalığın sebeplerinden biri de donanım önceden getirmesinin gelecekte erişileceği umulan adreslere erişim başlatırken derleme zamanı bilgisinden faydalanamamasıdır. Gereksiz önceden getirmeler programın doğru çalışmasını değil de önbellek kirliliği ve bellek band genişliği tüketimine yol açtığı için programın hızını etkiler. Donanıma dayalı teknikler derleyici desteği olmaksızın çalışma zamanı önceden getirme fırsatları ile ilgilenir. Bu durumda daha karmaşık yapılar kullanılabilmektedir. Yazılım önceden getirmesinde kullanılabilecek verinin getirilmesine çalışılır, donanımsal yaklaşımda ise daha spekülatif şekilde önceden getirme çağrıları başlatılır.

3.4 Önceden Getirme Yaparken Dikkat Edilmesi Gereken Hususlar

Önceden getirmenin yapılma zamanı büyük öneme haizdir. Eğer önceden getirme çok erken yapılırsa, önceden getirilen verinin bellek hiyerarşisinin daha üst kısımlarında bulunan yararlı verileri yerinden etmesi veya kendisinin üzerine başka verinin yazılması mümkündür. Eğer çok geç yapılırsa bu sefer asıl bellek çağrısından önce veri ulaşmamış olacağı için yine işlemcinin bellek beklemeleri söz konusu olacaktır.

Önceden getirilen verinin bellek hiyerarşisinde bir üst seviyeye yerleştirilmesi performans kazancı sağlamak açısından önemlidir. Birçok önceden getirme tekniğinde getirilen veriler çeşitli türlerdeki önbelleklere yerleştirilir. Diğer bazı tekniklerde veriyi erken önbellek boşaltmalarından ve önbellek kirliliğinden korumak için önceden getirilen veriler özel tamponlara yerleştirilir.

Önceden getirme işlemi sözcük, önbellek bloğu, öbek halinde önbellek blokları, program veri nesneleri bazında yapılabilmektedir. Alıp getirilecek veri miktarına karar verilirken önbellek ve bellek organizasyonu yapısı göz önünde bulundurulmalıdır. Tek işlemcili sistemler için en uygun miktar önbellek bloğu iken, büyük, dağıtık bellek kullanan çoklu işlemcili sistemlerin ağında bir veri transferi başlatma maliyetini telafi etmek için en uygun miktar öbek halinde bellek bloklarıdır (VanderWiel ve Lilja, 2000).

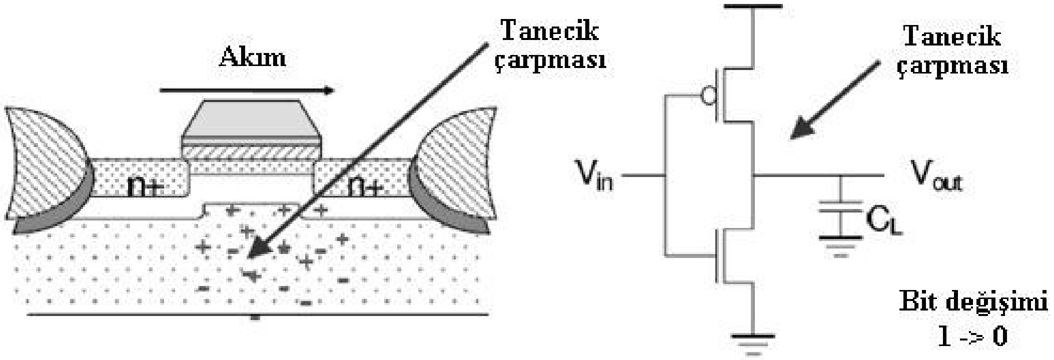
BÖLÜM 4

SOFT ERROR KAVRAMI

Tez çalışmamızın bu bölümünde soft error kavramını ve ilgili terimleri açıklayacağız.

4.1 Soft Error Nedir?

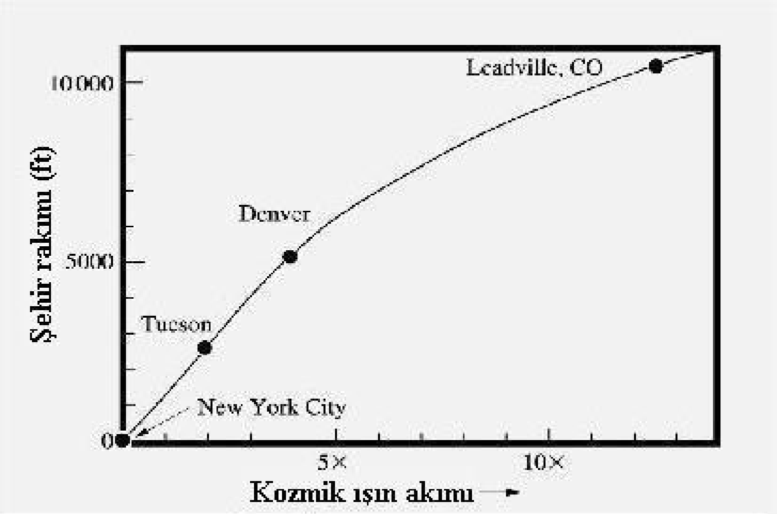
Modern mikroişlemcilerin tasarımı için performans ve güvenilirlik çok önemli kıstaslardır. Yeni nesil teknolojilerle Moore Kanununa göre çip başına düşen transistör sayısındaki büyük artış yarı iletken aygıtların performans ve işlevselliklerini artırmış dolayısıyla mikroişlemcilerin büyük bir gelişme kaydetmesini sağlamıştır. Ancak bu hızlı gelişim beraberinde soft error (hata) denilen hata türünün bilgisayar mimarisini olumsuz etkileyebilecek potansiyele kavuşmasına sebep olmuştur. Tekil durum değişimi (Single Event Upset - SEU) de denilen soft error yarı iletken bir aygıtta bit değişimidir (bit flip) (Ziegler ve diğ., 1996).



Şekil 4.1 Aygıt ve devre üzerinde tanecik çarpmasıyla soft error oluşumu (Tosun, 2005).

Paketleme maddelerinden ısı etkisiyle yayımlanan alfa tanecikleri ve kozmik ışınlardan gelen nötron tanecikleri gibi tanecik darbelerinden oluşan donanım geçici hataları işlemcilerin güvenilir bir şekilde çalışmaları için büyük bir engel oluşturur. Bu hataların nedenleri arasında elektriksel gürültüler ve elektromanyetik etkileşimler (çok yakında bulunan diğer elektronik devrelerdeki işlemlerin etkileri) de bulunur (Zero, 2006). Bu tekil durum değişimlerinden biriken şarj yanlış çıktılara, uygulama veya sistem yazılım çökmelerine neden olabilecek mandal (latch) gibi, SRAM bellek birimleri ve kapılar gibi mantıksal aygıtların durumunu (0,1) değiştirebilir (Ziegler, 1996). Küçülen aygıt boyutları (daha küçük kapasitans), giderek düşen voltaj eşik değerleri ve yüksek saat hızlarında işlem yapma işlemciyi soft error denilen bu tür geçici hatalara daha duyarlı hale getirmiştir. Ayrıca sızıntı enerjisini azaltmaya yönelik optimizasyonlar soft error oluşumu sorununu daha da kötüleştirmektedir (Degalahal ve diğ., 2003). İşlemcinin herhangi bir bileşeni soft errorlardan etkilenebilmektedir; ancak önbellekler de soft errorlara oldukça duyarlıdır. Çünkü günümüz mikroişlemcilerinde önbellekler çip üzerinde %60'lara varan oranlarda yer işgal etmektedir. Bir milyar transistör içeren işlemcilere yönelen eğilim (ki önbelleklerde transistör bütçesine %90'dan fazla kaynak ayrılır) mikroişlemci temelli bir sistemin güvenilirliğinin büyük oranda önbellek işlemlerine bağlı olacağı anlamına gelmektedir. Önbellekler CPU'ya çok yakın oldukları için onlardaki soft errorlar yazmaç (register) dosyasıyla diğer bileşenlere kolayca yayılabilir. Bu nedenle önbelleklerdeki bu geçici hatalar önlenmeye çalışılmalıdır.

Aygıtın tekil durum değişimlerinden kaynaklanan hata oranı hem karşılaştığı tanecik akımına hem de devre özelliklerine bağlıdır. Tanecik akımı çevre şartlarına göre değişebilmektedir. Kozmik ışın kaynaklı soft errorlar atmosferin yüksek kesimlerinde daha fazla görülmektedir. Örneğin, 1,5 km yükseklikte -Denver'ın (Colorado) yüksekliği- kozmik ışınlardan kaynaklanan nötron akımı deniz seviyesindekinden 3 ila 5 kat daha yüksektir (Şekil 4.2) (Mukherjee ve diğ., 2003; IBM, 2006).

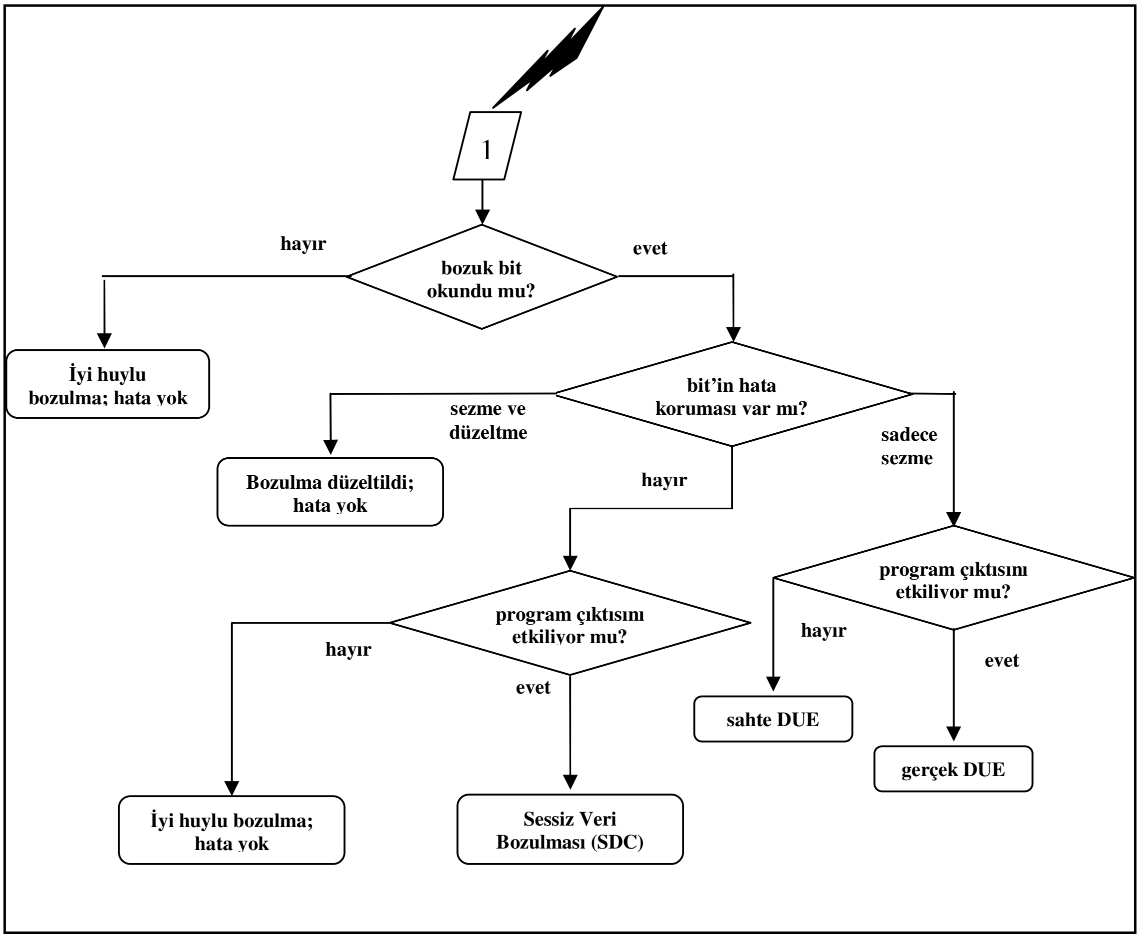


Şekil 4.2 Kozmik ışın akımının yüksekliklere göre değişimi (IBM, 2006).

DRAM üreticileri bellek birimi kapasitanslarındaki ve sağlanan voltajdaki düşüşü yavaşlatarak her yeni teknoloji neslinde bit başına düşen soft error oranının (Soft Error Rate - SER) çok fazla artmasını engellemişlerdir. Ancak SRAM aygıtları için durum hiç de öyle olmamıştır; her bir SRAM bellek hücresine düşen soft error oranı ya aynı kalmıştır veya çok az iyileştirilmiştir (Mukherjee ve diğ., 2005). Bir yandan da VLSI (Very Large Scale Integration) teknolojisindeki ilerlemeler önbellek güvenilirliği üzerinde pozitif bir etkiye sahip olabilir. Örneğin transistör boyutundaki azalma herhangi bir aygıt üzerindeki durum değişimini daha az olası hale getirir. Diğer yandan, işlemci teknolojisindeki eğilimler (daha küçük kapasitans, ihtiyaç duyulan ve eşik voltaj değerlerinin düşüklüğü ile yüksek saat hızlarında çalışma) önbellek güvenilirliğini olumsuz etkileyebilir (önbellekleri hatalara karşı daha bozulabilir yaptığı için). Örneğin, özellikle düşük sızıntı enerji için optimize edilmiş bir önbellekte daha düşük miktarda bir voltaj değişimi hataya neden olmayı sağlayabilmektedir. Bu ters etkilerin biri diğerini dengeler ve transistör başına düşen hata oranının yakın gelecekte hemen hemen aynı kalacağı tahmin edilmektedir. Sonuç olarak, önbelleklerin hata oranının barındırdıkları transistör sayısıyla orantılı olarak artacağı öngörülmektedir (Hareland ve diğ., 2001; Karnik ve diğ., 2001).

4.2 Soft Error Türleri

Soft errorlar sezilebilir ve sezilemez olmak üzere iki ana kategoriye bölünebilir. Sessiz Veri Bozulması (Silent Data Corruption - SDC) da denilen sezilemez hatalar sistemin hatalı çıktılar üretmesine neden olabilecek en sinsi hatalardır. Sezilebilir hataların iki biçimi vardır: sezilebilir düzeltilebilir hatalar ve sezilebilir düzeltilemez hatalar. Sezilebilir Düzeltilebilir Hatalar (Detected Recoverable Errors - DRE) hem sezilebilir hem de bazı güvenilirlik mekanizmaları kullanılarak düzeltilebilir. Sonuçta düzeltilebildikleri için büyük sorun değillerdir. Öte yandan Sezilebilir Düzeltilemez Hatalar (Detected Unrecoverable Errors - DUE) sezilebilir; ancak düzeltilemez. Bu DUE hatalarının programın sonuç çıktısını etkileyenlerine gerçek DUE etkilemeyenlerine sahte (false) DUE denir. Belki de bunlarla başa çıkmanın en iyi yolu sezildiklerinde programın çalışmasını durdurmaktır. Bu hata türlerinin oluşumunu anlamak için Şekil 4.3'e dikkat edelim.



Şekil 4.3 Bozulan bir bitin neden olabileceği mümkün çıktıların sınıflandırılması (Mukherjee ve diğ., 2005).

4.3 Soft Error Ölçümleri ve Soft Error Oranı Azaltma Yöntemleri

Güvenilirliği ölçmek için pratikte kullanılan iki yaygın teknik FIT (Zaman içindeki başarısızlık - Failure in Time) ve MTBF (Başarısızlıklar arasındaki ortalama zaman - Mean Time Between Failures) kıstaslarıdır. FIT bir milyar saat içerisinde karşılaşılan hata sayısını tutar. MTBF, FIT ile ters orantılıdır (FIT=1/MTBF). Bir çipin SDC ve DUE FIT oranları her bir bileşeninin hataya katılımları toplanarak bulunur. Her bir bileşenin FIT oranı iki etkenin ürünü olarak ortaya çıkar: durum değişimlerinden kaynaklanan geçici bozulmaların oranını gösteren ham (raw) aygıt hata oranı ve aygıtta bir durum değişiminin mimari olarak görülebilir SDC veya DUE hataya neden olması olasılığı olan mimari bozulabilirlik faktörüdür (Architectural Vulnerability Factor - AVF) (Mukherjee ve diğ., 2005).

Hesaplama sistemi bileşenleri üzerinde soft error oluşumu nadiren olmasına rağmen, yukarıda bahsedilen eğilimlere bağlı olarak performans, form faktörü, düşük güç tüketimi ve düşük maliyet gibi tasarım kriterlerinin yanında son zamanlarda önemli bir kriter haline gelmiştir. SER, diğer sistem tasarım kriterlerini de etkileyebilir (örn., güç.). Veri önbellekleri için geçici (transient) hatalar ve sızıntı enerjisi tasarruf teknikleri arasındaki etkileşimler (Li ve diğ., 2004)'de çalışılmıştır. Li ve diğ. (2003) çeşitli dinamik gürültü türleri etkisindeki çip üzerindeki ara bağlantılar için enerjiyi verimli kullanan uyarlanabilir hata koruma planı sundular. İşlemci üreticileri genel olarak ürünlerinin SER bütçelerini hedef pazarlarının güvenilirlik gereksinimini karşılayacak şekilde belirlediler. Örneğin, IBM, Power 4 sistemi için 114 SDC FIT (1000 yıllık MTTF - Mean Time to Failure), 4.556 sistem-öldürücü DUE FIT (25 yıllık MTTF) ve 11.415 süreç-öldürücü DUE FIT (10 yıllık MTTF) hedeflemektedir (Mukherjee ve diğ., 2005). Genel olarak, tasarımcılar hedefledikleri SER değerlerine ulaşabilmek için en hesaplı uygun güvenilirlik artırıcı mekanizmayı seçmeliler.

Bir çipin SER'ini azaltmak için sunulan birçok teknik bulunmasının yanında bu teknikler performans, alan, güç veya tasarım zamanı konusunda ek maliyetleri beraberinde getirmektedir. Donanım seviyesi güvenilirlik artırıcı teknikler başlıca 3 kategoriye bölünebilir: süreç, devre ve mimari çözümler (Mukherjee ve diğ., 2005). Süreç teknolojisi temelli çözümlerde silikon yalıtkanı (SOI) aygıtı hatalara karşı korumak için kullanılır. SOI aygıtı daha ince bir silikon katmana kıyasla ışınımdan daha az miktarda voltaj toplar, bu durumda darbenin mantıksal aygıtın durumunu (0,1) tersine çevirmesi çok olası değildir. IBM, SOI teknolojisi kullanılarak SRAM aygıtların SER'inin 5 kat azaltılmasını başarmanın mümkün olduğunu raporluyor (Cannon ve diğ., 2004). Devre temelli çözümlerde (Calin ve diğ., 1996), ışınıma karşı güçlendirilmiş bellek birimleri, bellek birimi kapasitansı ve/veya sahip olunan voltaj gibi aygıt parametreleri ayarlanarak tasarlanır. Bunlar, bellek hücresi durumunu (1,0) değiştirebilmek için gereken minimum voltajı yükselttiği için SER değeri düşer. Bununla birlikte, tasarımda ışınıma karşı güçlendirilmiş bellek hücrelerinin kullanılması büyük güç ve alan masraflarını beraberinde getirmektedir.

Önbelleklerin güvenilirliğini artırmak için kullanılan mimari çözümler arasında eşitlik (parity) sınaması ve hata düzeltme kodları (Error Correcting Codes - ECC) (Pradhan ve diğ., 2003), π bit (Weaver ve diğ., 2004), N mod fazlalık (N modular redundancy - NMR) gibi kopyalama bileşenleri ve kopyalama önbellek (Zhang ve diğ., 2003) gibi bütünlük kontrolü yapan teknikler yer almaktadır. Sekiz bitlik veriye bir fazladan bit ekleyen byte eşitliği mekanizması önbelleklerde hata kontrolü için yaygın olarak kullanılır. Eşitlik (parity) basit ve az enerji tüketmesine rağmen %12,5 fazladan alan harcanmasına neden olmaktadır. Ayrıca, sadece tek sayıda hataları sezebilir ve bunları düzeltemez. Sonuç olarak, özellikle yüksek seviyede güvenilirlik talep eden veya gürültülü ortamlarda (örn. yüksek yükseltiler) işleyen birçok sistem için tek başına geçerli bir çözüm sağlamaz.

Popüler bir ECC, her 64 bit'lik parçaya fazladan 8 bit ekleyen SECDED (Single Error Correct Double Error Detect)'dir. Eşitliğin aksine, SECDED'in uygulanması daha karmaşıktır ve tekil bit hatalarını düzeltebilmektedir. Bununla birlikte, önbellek erişiminin bir devirde (cycle) tamamlanmaması (kritik yola yüklemelerin yerleşmesi) dolayısıyla özellikle yüksek saat frekanslı işlemciler için önbellek erişim gecikmesini artırabilir ve buna bağlı olarak güç tüketimini yükseltebilir. Bu nedenle ECC temelli koruma planları önbellek hiyerarşisinde L2 ve L3 önbellekler gibi daha nadir erişilen bileşenler için daha uygundur.

NMR temelli teknikler çoğunluk oylaması (majority voting) veya benzer mekanizmalar kullanarak veri güvenilirliğini önemli ölçüde artırabilmesine rağmen önemli miktarda olan masrafını beraberinde getirmektedir. Örneğin, üçlü mod fazlalıktan (Triple modular redundancy) (Carmichael, 2001) kaynaklanan fazladan alan masrafı %200'dür; boyut kısıtlamaları olan sistemler için bu çok maliyetlidir.

π bit, sahte (false) DUE'ları (sahte DUE, çıktıyı etkilemeyen DUE hatalarıdır) azaltmak için kullanılan bir hata yayılım mekanizmasıdır. Bir hata sezildiğinde (eşitlik sınamasıyla örneğin) makine kontrolü için istisnai bir durum sinyali vermektense, etkilenen komutun π bit'i ayarlanır. Daha sonra paralel çalıştırma sürecinde, eğer hatalı komutun yanlış yol komutu (wrong path instruction) olduğuna karar verilirse sahte DUE olayından sakınılarak π bit görmezden gelinir.

Aynı komutun çoklu kopyalarının çalıştırılmasını temel alan fazladan çoklu-iş parçacıkları teknikleri (redundant multi-threading techniques) hata sezme ve düzeltme için önerilmiştir (Gomaa ve diğ., 2003; Ray ve diğ., 2001; Vijaykumar ve diğ., 2002; Reinhardt ve Mukherjee, 2000). Bununla birlikte, işlemci kaynaklarına fazladan baskı oluşturur. Bu fazlalığın performans etkisini düşürmenin bir yolu olarak Kumar ve Aggarwal (2006) tarafından fazladan kaynak kullanımının etkisinin azaltılması konusunda çalışma yapılmıştır. Gomaa ve Vijaykumar (2005) paralel işleme sırasında oluşan boş kaynakların ve L2 ıskalarının yol açtığı fazlalığı kullanarak dağıtım (issue) kuyruğunun soft error oranı ve performans arasındaki getiri-götürülerin çalışılmasına yardımcı olacak teknikler önermişlerdir.

Çeşitli donanım bileşenlerinin mimari bozulabilirlik faktörlerini (AVF) ölçmek için önerilen birkaç bozulma enjeksiyon (fault injection) çalışmaları da mevcuttur (Kim ve diğ., 2002; Wang ve diğ., 2003). Bozulma enjeksiyonu uygulayarak ve işlemcinin hataya en çok eğilimli kısımlarını belirleyerek, Wang ve diğ. (2004) bu bileşenleri geçici hatalara karşı korumak için teknikler önermişlerdir.

Önbellek güvenilirliğini artırmanın diğer bir yolu önbellekteki veriyi kopyalamaktır (çiftlemek-duplicate). Kopyalama önbelleğin kendi içinde yapılabilir (Zhang ve diğ., 2003) veya ayrı bir önbellek kullanılabilir (Zhang ve diğ., 2004); bu seçeneklerin her ikisi de alan ve performans bakımından değişik sonuçlar getirecektir.

4.4 Sanayide Soft Error Etkileri

2000 yılında Sun Microsystems firmasının yüksek seviyeli sunucularında ani ve esrarengiz çökmeler meydana geldi. Gerçek araştırıldığında çökmenin nedeni olarak UltraSparc II işlemcisinin ve dahili önbelleğinin soft errorlara karşı korunmaması olduğu gösterildi. Bell Atlanta, America Online, Ebay ve Verisign gibi Sun sunucuları üzerinde hizmetlerini veren şirketler sıkça ve uzun süreli sistem servis aksamaları yaşadılar. Sun firmasının müşterilerinden biri sorunu teşhis etmek ve düzeltmek için 3 milyon ABD doları harcandığını duyurdu. Bu konudaki başarısızlıklar güvenilir ve sorunsuz sunucu satışı pazarında Sun firmasına olan güvenin azalmasına ve firmanın rekabette geri düşmesine neden oldu. Bu süreç sonunda Sun Microsystems büyük müşterilerini rakiplerine kaptırmıştır.

Son yıllarda, üst düzey bilgisayar ağları ekipmanları soft errorlara karşı artarak dayanıksızlaşmaktadır. 2003 yılında Cisco 12000 hat kartlarında soft errorlar hizmet aksamasına neden olmuştur. Soft errorlardan kaynaklanan bellek eşitlik hataları (memory parity errors) ve uygulamaya yönelik tümleşik devre (application-specific integrated circuit - ASIC) hataları meydana gelmiştir.

2004 yılında, ASC Q süper bilgisayarı (Hewlett-Packard sunucularını temel alan) ve System X süper bilgisayarı (Apple sunucularını temel alan) - ki bu bilgisayarlar Kasım

2003'te yayınlanan ilk 500 süper bilgisayar listesinde sırasıyla ikinci ve üçüncü en hızlı süper bilgisayar olarak yer alıyordu - radyasyon temelli soft errorların sıkça neden olduğu çok kritik başarısızlıklar yaşadılar. 2004 nisanın sonlarından 2004 temmuzun başlarına kadar olan 10 haftalık bir zaman diliminde Q kümeleme sistemi ortalama olarak haftada 26.1 CPU başarısızlığı yaşadı. HP ASC Q kümeleme sisteminin ilk geliştirme aşamalarında önbellek sistemlerinde umulmadık oranda yüksek soft error oranıyla karşılaşılınca soft error belirtileri zaten tespit edilmişti. Birçok test ve deney bu yüksek oranın sebebinin kozmik nötron ışınlarının akışı olduğunu doğrulamıştır (Zero, 2006).

Görev kritik uygulamalar soft errorlar tarafından tehdit altındadır. Bu konudaki örnekler Çizelge 4.1'de verilmiştir.

Çizelge 4.1 Sanayide karşılaşılan soft error durumlarına örnekler (Zero, 2006).

Sanayi Alanı	Firma Örneği	Örnek Sistem	Soft Error Etkisi	Soft Error Sonucu	Risk türü
Otomotiv & Taşımacılık	BMW, Audi	Antilock Fren Sistemi (ABS)	ABS durur	Emniyetsiz Frenleme	Hayati tehlike
Havacılık & Askeriye	Boeing, Lockheed-Martin	Uçaklar	Denge denetleyicisinin başarısızlığı	Emniyetsiz uçuş	Hayati tehlike
Dikilebilir Aygıt	Medtronic, Guidant	Kalp atışlarını düzenleyen aygıt	Mikrodenetleyici başarısızlığı	Düzensiz kalp atışı	Hayati tehlike
Bilgisayar Ağları	Cisco, Nextel, Juniper	Yönlendirici	Yönlendirici yeniden başlamaları	Bilgisayar ağının devre dışı kalması	İletişim kaybı
Üst Düzey Hesaplama	IBM, SUN, Hewlett-Packard	E-ticaret sunucusu	Mikroişlemci başarısızlığı	Veritabanına bozuk veri yazılır	Bilgi kaybı
Alt Düzey Hesaplama	Intel, AMD, ARM	Dizüstü bilgisayar	Uygulama donar	Kullanıcı uygulamayı yeniden başlatmalıdır.	Kullanıcı memnuniyetsizliği
Tüketici Elektroniği	Nokia, Samsung	Cep telefonu	Konuşma sürerken cep telefonu donar	Kullanıcı aygıtı yeniden başlatmalı ve tekrar aramalı.	Kullanıcı memnuniyetsizliği

BÖLÜM 5

SOFT ERROR MODELİ

Enerjili taneciklerin darbelerinin sebep olduğu soft errorlar özellikle gürültü barındıran ortamlarda çalışan hesaplama sistemleri için önemli bir güvenilirlik kaygısı ortaya çıkarmaktadır. Teknolojideki hızlı gelişmeler ve sızıntı enerjiyi azaltmaya yönelik denetim mekanizmaları bu geçici hatalardan kaynaklanan problemi çok daha şiddetlendirmektedir. Bundan dolayı, işlemci/bellek tasarımlarında soft errorlara karşı korunmak için güvenilirliği arttırıcı mekanizmaların kullanılması çok önemlidir. Bunu yapabilmek için önce soft errorları modellemeye ihtiyacımız var ve ardından modeli temel alarak güvenilirliği arttırıcı çeşitli teknikler arasında maliyet/güvenilirlik getiri-götürülerini sistem ihtiyaçlarını karşılayabilecek şekilde tespit etmek için çalışmalıyız.

Bu bölümde önbellekler için mimari seviyede soft error modelimizi tanıtacağız. Bu tez çalışmamızda esas alacağımız modeli açıklayacağız. Güvenilir sistem tasarımcıları için, özellikle önbellekler gibi soft errorlara karşı daha hassas sistem bileşenleri için mimari seviyede soft errorların analizini mümkün kılan modellere sahip olmak çok önemlidir. Bu, iki ana nedenden kaynaklanır. Birincisi, bu tür modelleri kullanarak sistem tasarımcıları çok çeşitli sistem bileşenleri için soft errorları daha iyi anlarlar. İkincisi, bu modeller tasarımcıların farklı güvenilirlik artırıcı teknikleri değerlendirirken uygun maliyet/güvenilirlik dengesi kurmalarına yardımcı olur ve onların sistem gereksinimlerini/hedeflerini ve bütçe kısıtlamalarını temel alarak en iyi alternatif tasarımı seçmelerini olanaklı kılar.

Yakın zamanlarda, soft errorların mimari seviyede modellenmesine odaklanan çalışmalar yapılmıştır. Mukherjee ve diğ. (2003) mimari olarak doğru çalışma bitlerini (Architecturally Correct Execution - ACE) temel alarak mimari bozulabilirlik faktörü (Architectural Vulnerability Factor - AVF) kavramını önermişlerdir. Bu modelde, işlemci durum bitleri iki gruba ayrılır: ACE ve ACE-olmayan bitler. ACE-olmayan bitler iki alt kümeye ayrılır: mikromimarisel ACE-olmayan bitler ve mimarisel ACE-olmayan bitler. ACE bitler programın doğru çalışması için doğru olması gereken bitlerdir. Dolayısıyla diyebiliriz ki, sadece ACE bitleri etkileyen hatalar hatalı bir çıkışa neden olur. Burada anahtar mevzu hangi bitlerin ACE hangi bitlerin ACE-olmayan olduğuna nasıl karar verileceğidir. Bu amaçla, her bir komut için Mukherjee ve diğ. (2003)’de detaylı bir şekilde anlatılan iş hattı durumları (pipeline stage) izlenir. Örneğin yanlış tahmin edilen bir dallanma yönünden alınan komutların etkilenerek değişen işlemci durum bitleri ACE-olmayan bitler olarak varsayılır. AVF ölçütü, işlemci bileşenindeki bir bozulmanın son program çıktısında görünür bir hataya neden olması olasılığıdır ve temel olarak veri yoluna odaklanır. IA-64 mimarisinin komut kuyruğu ve çalıştırma birimleri için AVF'ler Mukherjee ve diğ. (2003)'de raporlanmıştır.

Li ve diğ. (2005), mimari seviyede soft errorları modellemek ve çözümlemek için SoftArch denilen bir araç önerdiler. Bu, olasılığa dayanan bir hata oluşumu ve yayılımı modelini temel alır. Bir talimattan diğer türevlerine soft error yayılımını modellemek için veri akış grafı (Data Flow Graph – DFG) benzeri bir yaklaşım kullanılır. Bu amaçla, her bir veri değeri için temel bir hata kümesi tanımlanır. Temel hata kümesindeki her eleman bir soft errorın varlığını ifade eden temel bir hata olayıdır. Örneğin, önbellek gibi bir depolama yerinden bir veri okunduğu anda, son okunmasından bu yana geçen süre içinde hataya maruz kalması olasılığını ifade eden temel bir hata olayı onun temel hata kümesine eklenir. Tüm temel hatalar birbirinden bağımsızdır ve bir değerden diğerine hata yayılımını takip etmek için kullanılabilir.

Bu ilk modelleme çabalarına kıyasla bizim modelimiz önbellekleri hedefliyor. Bazı hataların programın sonuç çıktısını etkilemeyebileceğini temel alan gözlemlerimize dayanarak modelimizi açıklayacağız. Bu gözlemlerimizin iki durumdan kaynaklandığını belirtelim. Birincisi, bazı önbellek hataları önbelleğin kendi içinde ayrıştırılabilir ve diğer sistem bileşenlerine yayılmaz, bu tarz hatalara maskelenmiş hatalar (masked errors) diyeceğiz. Örneğin, geçersiz veri tutan bir blokta oluşan hata veya değişikliğe uğramamış bir bloğun erişilmemiş bir kısmında oluşan hata diğer sistem bileşenlerine yayılamayacaktır, dolayısıyla ayrıştırılmış olacaktır ve sonuç çıktıyı etkileyemez. İkincisi, bir önbellek hatası diğer sistem bileşenlerine yayılsa bile, programın sonuç çıktısında görünür bir hataya neden olmayabilir. Bu tür hatalara iyi huylu hatalar (benign errors) diyeceğiz. Örneğin, ölü/yanlış tahmin edilmiş bir talimatı besleyen veri önbelleği bloğundaki bir hata sistem güvenilirliğini etkilemeyecektir.

Bizim modelimiz iki bileşene sahiptir: hata oluşumu ve hata yayılımı. Her bir önbellek satırını bir sözcükler (words) kümesi gibi düşünüyoruz. Burada "sözcük" derken veriyi tutmak için kullanılan herhangi depolama birimini kastediyoruz, dolayısıyla boyutu tuttuğu veriye bağlı olarak değişir. Örneğin, bir sözcük türü C programlama dilindeki float veya int gibi temel veri türlerinden biri olan bir değişkene veya dizinin bir elemanına karşılık gelebilir.

l satırında *d* verisini tutan bir sözcük $w_{l,d}$ ile gösterilir. Blok ve satır kavramları ile aynı ifadeyi, önbellek satırlarını kastediyoruz. Mimari modelimizde her bir sözcüğe iki özellik eklenmiştir. Hata Oluşumu (Error Generation - *EG*) ve Hata Yayılım Kümesi (Error Propagation Set - *EPS*). Bu özellikleri ilerde açıklayacağız. Daha sonra *EG* ve *EPS*'yi temel alarak önbellek yapısındaki bir bozulmanın programın sonuç çıktısında görünür bir hataya neden olması olasılığını veren AVFC (Architectural Vulnerability Factor for Caches – Önbellekler İçin Mimari Bozulabilirlik Faktörü) ölçütünü tanımlıyoruz.

5.1 Hata Oluşumu

Bir veri öğesinde soft error, yüksek enerji taneciklerinin çarpmaları sonucu biriken elektriksel gücün veriyi tutan aygıtın (aygıtların) çıktısını değiştirmesi, dönüştürmesi sonucu oluşur. Modelimizde SRAM önbelleğindeki ham hataların (raw errors) sabit bir oranda (Li ve diğ., 2005), rasgele ve uniform olarak dağıtıldığını (Mukherjee ve diğ., 2003) varsayıyoruz. Bundan dolayı, bir veri öğesinde hata oluşma olasılığı verinin boyutuyla ve tanecik darbelerine maruz kalabileceği zaman dilimiyle doğrudan orantılıdır. Bunu ifade etmek için, *w* sözcüğü (*l* satırında bulunan ve *d* verisini tutan) için *EG*'yi şöyle tanımlıyoruz:

$EG(w_{l,d})$ = boyut ($w_{l,d}$) x maruz kalma süresi ($w_{l,d}$).

5.2 Hata Yayılımı

Program çalışırken değerler önbellekten yazmaçlara okunur ve bu değerler veri yolunda, sonrasında yine önbellekte tutulacak olan yeni değerlerin hesaplanması için kullanılabilir. İlk değerlerdeki soft errorların yazmaç dosyası vasıtasıyla yeni değerlere kolayca yayılabilmesi mümkündür. Bu hatalar, eğer düzeltilmezse kirli veriler nedeniyle önbellek hiyerarşisinde daha alt seviyelere yayılabilir. *l* satırında değişikliğe uğramış her *d* verisi için $EPS(w_{l,d})$ kümesini tanımlıyoruz. Bu ifade, $w_{l,d}$ 'deki değerin hesaplanmasında kullanılan verinin hata yayılım etkisini göstermek için kullanılan bir kümedir. Daha iyi anlamak için şu örneğe bakalım: $v1 = v2 + v3$. Burada *v2* ve *v3*'ün önbelleğin *l* satırında bulunan iki değeri ifade ettiğini düşünelim. *v2* ve *v3* değerlerinin işlemciye sırasıyla *R2* ve *R3* yazmaçlarıyla getirildiğini ve toplama işleminin sonucunun *R1* yazmacında tutulduğunu varsayalım. Eğer önbellekteki *v2* değeri hatalıysa *R2* bozulacaktır, bu da *R1*'i bozacaktır. Sonuç olarak, *R1* önbellekte *v1*'e karşılık gelen yere yazıldığında hatalı bir değer içerecektir.

Başka bir deyişle, $v2$ değerindeki hata $R2$ ve $R1$ yazmaçları vasıtasıyla önbellekte $v1$'e yayılmış olacaktır. Sonuç olarak, bu örnekte, $w_{l,v1}$ sözcüğü için *EPS* şu şekilde hesaplanır:

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

Bu ifadeden anlaşılacağı üzere, diğer başka değerlerden $v2/v3$'e sıçrayarak yayılan hatalar da $v1$'in güvenilirliğini etkilemektedir.

EG değerlerini devir sayısını dikkatlice ölçen simülatör kullanarak hesaplayabiliriz. Her bir kirli (değiştirilmiş) önbellek satırı için satırın her sözcüğüne bir *EPS* ekliyoruz. *EPS*, satırın yaşam süresi boyunca bulunduğu sözcüğe biriken soft error yayılımını ifade eder, sözcüğün önbelleğe getiriliş anı başlangıç, bir başka sözcükle yer değiştirilmesi anı bitiş kabul edilir. Bu durumda, önbellek satırının *EPS* değeri, o satırda bulunan sözcüklerin *EPS* kümelerinin birleşimi alınarak elde edilir. Programın veri akış grafı (DFG) üretilen yeni bir değeri bulmak için hangi değerlerin kullanıldığını belirlemek için kullanılır. Bu da hata yayılımının yönünü belirlememize yardım eder.

5.3 AVFC

Maskelenmiş (masked) veya iyi huylu (benign) olmayan soft errorlara ciddi hatalar (serious errors) denir. Bunlar programın doğru çalışması üzerinde güvenilirlik problemlerine yol açabilecek çok önemli soft errorlardır. Bu çalışmada, ciddi (serious) olduğu düşünülen iki hata türü üzerine odaklanıyoruz. İlk tür değiştirilmiş (kirli) önbellek bloklarındaki hataları içerir. Bu noktada, her değiştirilmiş bloğun programın görünür çıktısını değiştirebileceğini varsayıyoruz. İkinci tür, programın çalışma denetim akışını (bir dallanma komutunun hangi yöne/hedefe yöneleceğini belirleyen değerler) değiştiren değerlerdeki hatalara karşılık gelir. Bu iki ciddi hata türünü temel alarak, önbellekte değiştirilen bloklar için toplam hata üretimi ve yayılımı ($TEGP_C$ - Total Error Generation and Propagation), verileri besleyen dallanma komutları için toplam hata yayılımı (TEP_B - Total Error Propagation) ve bizim AVFC terimimiz şöyle ifade edilir:

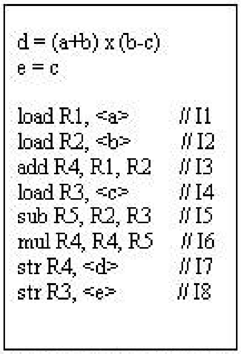
$$TEGP_C = \sum_{\substack{\text{yer değiştirilen}\\ \text{kirli } l \text{ satır}}} \sum_{d} \left(EG(w_{l,d}) + \sum_{\substack{e \in\\ EPS(w_{l,d})}} |e| \right) \quad (1)$$

$$TEP_B = \sum_{b\ dallanma} \sum_{d} \sum_{\substack{e\ \in \\ EPS(w_{b,d})}} |e| \qquad (2)$$

$$AVFC = \frac{TEGP_C + TEP_B}{\textit{Önbellek Boyutu}\ \mathrm{x}\ \textit{Çalıştırma Devir Sayısı}} \qquad (3)$$

Denklem (1) önbellekteki hataların etkisini göstermektedir. Bu eşitlikten görüldüğü gibi, sadece değiştirilen (kirli) önbellek bloklarını etkileyen hatalar dikkate alınıyor. Değiştirilen önbellek bloklarındaki hataların iki kaynağı vardır. İlki hata oluşumu, ikincisi diğer bozulmuş önbellek bloklarından hata yayılımıdır. Bunların her ikisinin de yukarıda verilen ilk denklemden elde edildiğine dikkat edelim. Yer değiştirilen kirli önbellek satırı *l*'nin hata toplamının sahip olduğu her bir sözcüğün hata katılımı düşünülerek hesaplandığına da dikkat edelim. Burada $|e|$ *l* satırında *d* verisini tutan sözcüğün *EPS* kümesinde bir eleman olan *e* hatasının bayt x devir (byte x cycle) biriminde hataya katılımını ifade eder. Yukarıdaki denklem (2) çalışma akışını değiştirebilecek değerlerdeki hataların etkisini ifade eder. *b* dallanmasının yönünü ve/veya hedefini belirlemenin kararında gereken hatalı *d* verisini tutan sözcük $w_{b,d}$ ile gösterilir. CPU'da hata oluşumu olmadığını varsaydığımız için ikinci denklemdeki dallanmaların gerçekleştiriminde paylaşılan değerlerde CPU'ya getirildikten sonra hata oluşmadığını kabul ediyoruz. Bunun anlamı, bu değerlerdeki hataların sadece önbellekteki değerlerden hata yayılımı yüzünden olduğunun kabulüdür. İlk iki denklem esas alınarak, AVFC yukarıda verilen denklem (3)'te gösterildiği gibi ifade edilebilir.

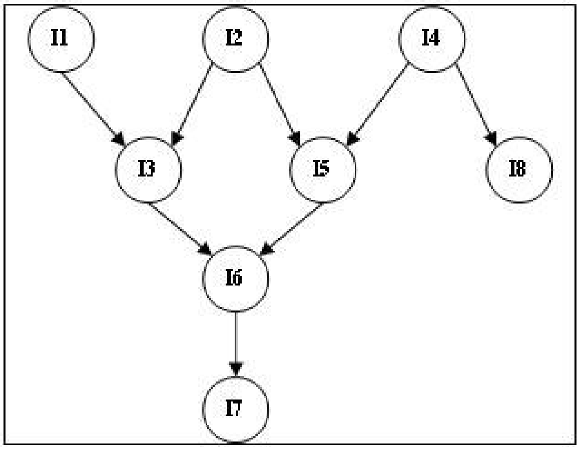
Modelimizde L2 önbelleğinde hata oluşumu görülmediğini ve gereken verinin L1'e daha önceki gelişlerinden birinde güncellenmiş olsa bile L2 önbelleğinden L1 önbelleğine veri akışlarının daima hatasız olduğunu kabul ediyoruz. L2'den L1'e veri akışlarının L1 önbelleğine ziyaretlerinden önce bazı soft errorlar taşıyabileceği olasılığını hesaplamak için modelimiz kolayca değiştirilebilmektedir.



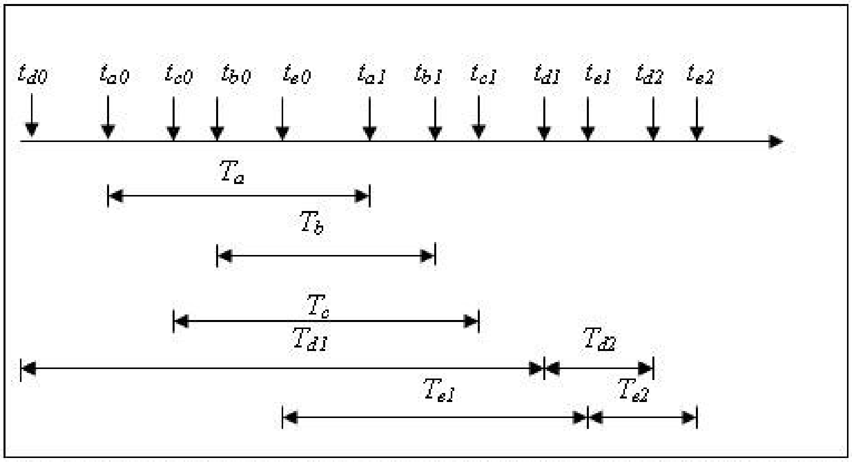
```
d = (a+b) x (b-c)
e = c

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

Şekil 5.1 Bir kaynak kod parçası (üstte) ve karşılık gelen makine dili kodu (altta).



Şekil 5.2 Şekil 5.1'deki kod parçasının veri akış grafı.



Şekil 5.3 Şekil 5.1'deki kod parçası tarafından işlenen veriler için önbelleğe ziyaretlerini içeren zaman çizelgesi şekli.

5.4 Örnek

Şimdi hata oluşum ve yayılımı modelimizin pratikte nasıl kullanılacağını örnek üzerinde gösterelim. Bu örnekte, Şekil 5.1'de verilen kod parçasına ve ona karşılık gelen makine dili (assembly) koduna odaklanıyoruz. Şekil 5.2, Şekil 5.1'deki makine dili kodu için DFG'yi göstermektedir. Koddaki verinin L1 veri önbelleğine olan ziyaretlerini gösteren zaman çizelgesi Şekil 5.3'te verilmektedir. a verisini tutan önbellek satırındaki her değişkenin önbelleğe t_{a0} anında getirildiğini ve a'nın t_{a1} anında R1 yazmacına yüklendiğini varsayıyoruz. Her değişkenin farklı bir önbellek bloğunda yerleştiğini de varsayıyoruz.

Bu örnekte sözcüklerin ifadesinde ikinci küçük alt indeksi kullanmıyoruz. boyut(a) = 4 bayt olduğu varsayımıyla önbellekte soft errorlara maruz kalma süresi T_a olduğundan $EG(w_a)$, 4 bayt x T_a olacaktır. Benzer olarak b ve c için sırasıyla T_b ve T_c zaman dilimleri süresince soft errorlara maruz kaldığını varsayalım. Dolayısıyla EG değerleri $EG(w_b)$ = 4 bayt x T_b ve $EG(w_c)$ = 4 bayt x T_c olmaktadır. d verisini tutan w_d sözcüğü için *EPS* değeri

$$EGS(w_a) \cup EGS(w_b) \cup EGS(w_b) \cup EGS(w_c)$$

olarak hesaplanır. İlk iki değer *R4* yazmacından, son iki değer *R5* yazmacından yayılır. *R1*, *R2* ve *R3* yazmaçlarıyla taşınan hata kümeleri ayrık olmasına rağmen *R4* ve *R5* tarafından taşınan hata kümelerinin ayrık olmadığına dikkat edelim. Sonuç olarak $EPS(w_d)$ kümesi $\{EG(w_a), EG(w_b), EG(w_c)\}$ şeklinde ifade edilir. $|EPS(w_d)|$ ile $EPS(w_d)$'nin hataya katılımı 4 bayt x ($T_a + T_b + T_c$) şeklinde hesaplanır. Eğer d verisini içeren satırın önbelleğe t_{d0} anında getirildiğini, d'ye önbellekte t_{d1} anında yazma yapıldığını ve d'yi içeren satıra t_{d2} anında yer değiştirme işlemi yapıldığını varsayarsak satırda bulunan w_d haricinde her bir sözcüğün *EG* değeri 4 bayt x ($T_{d1} + T_{d2}$) olur. Öte yandan, $EG(w_d)$ 4 bayt x T_{d2} olur. Bu, şunu gösterir: T_{d1} süresi boyunca oluşan soft errorların üzerine w_d için t_{d1} anında yazma işlemi yapıldığı için bu w_d'nin güvenilirliğini etkilemeyecektir. Şimdi bir önbellek satırının w_d dahil 8 sözcüğe sahip olduğunu düşünelim. Satır için toplam *EG*

$$7 \text{ x } 4 \text{ bayt x } (T_{d1} + T_{d2}) + 4 \text{ bayt x } (T_{d2} + T_a + T_b + T_c)$$

şeklinde hesaplanacaktır. Bir de e'nin d'nin tutulduğu satırdan başka bir satırda yer alan bir veri olduğunu varsayalım. e, t_{e0} anında önbelleğe getirilmiş, t_{e1} anında I7 talimatına karşılık gelen depolama (store) işlemi gerçekleşmiş ve sözü edilen önbellek satırı için t_{e1} anında yer değiştirme işlemi yapılmış olsun. Bu varsayımlara dayanarak, veri depolayan sözcüğün EPS değerinin hesaplanmasında T_c süresi boyunca d'yi depolayan satır zaten dikkate alınmış olduğundan tekrar dikkate alınmaz. Sonuç olarak, içerisinde e barındıran satırın e'yi tutan sözcüğü için hataya maruz kalış süresi T_{e2} iken bu sözcüğün dışındaki tüm sözcükler için hataya maruz kalış süresi $T_{e1} + T_{e2}$'dir.

BÖLÜM 6

ÖNCEDEN GETİRME İLE SOFT ERRORLAR ARASINDAKİ İLİŞKİLER

6.1 Kullanılan Araçlar

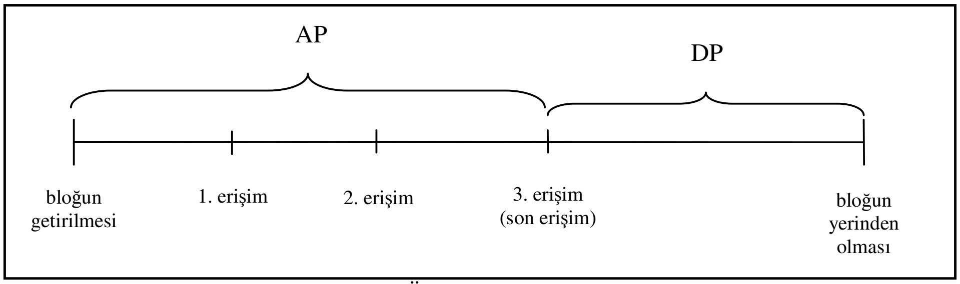
Yapılan tez çalışmasında, önceden getirme işlemi ile soft errorlar arasındaki ilişkilerin araştırılması için SimpleScalar 3.0 (Burger ve Austin, 1997) simülatörü, üzerinde değişiklikler yapılarak, kullanılmıştır. SimpleScalar, uygulama programlarını hızlı çalıştırma temelli simülasyon kullanarak modern işlemciler ve sistem mimarileri üzerinde simüle etmeye yarayan bir araç setidir. Bu çalışmada bu araç setinden sim-outorder bileşenini kullandık.

Çalışmamızda SPEC2000 (SPEC) performans ölçüm testlerinden 20 tanesi kullanılmıştır. 4 tanesinin derlenmiş ikili kodları olmadığından bunlarla testler yapamadık. Herhangi bir testi simüle etmek çok uzun süre aldığından, ilk 1 milyar komutu hızlı çalıştırıp sonraki 500 milyon komutu simüle ettik. Çizelge 6.1 başlıca simülasyon parametrelerinin varsayılan değerlerini göstermektedir.

Çizelge 6.1 Başlıca yapılandırma parametreleri ve deneylerimizde kullanılan değerleri.

İşlemci Çekirdeği	
Fonksiyonel Birimler	8 integer ve 8 FP ALU 4 integer çarpıcı/bölücü 4 FP çarpıcı/bölücü
LSQ boyutu	64
RUU boyutu	256
Fetch/Decode/Issue/Commit genişliği	8 komut/devir
Fetch kuyruk boyutu	8 komut
Önbellek ve Ana Bellek Yapısı	
L1 komut önbelleği	64 KB, 64 byte blok boyutu 4-yollu, 1 devir gecikme süresi
L1 veri önbelleği	64 KB, 64 byte blok boyutu 4-yollu, 1 devir gecikme süresi
L2 önbellek	1 MB birleştirilmiş, 128 byte blok boyutu 8-yollu, 12 devir gecikme süresi
Veri/Komut TLB	128 kayıt, tam çağrışımlı 30 devir ıska gecikme süresi
Ana bellek	160 devir gecikme süresi

Deneysel sonuçlarımızı daha kolay yorumlayabilmemiz için iki parametre tanımlayacağız: Aktif periyot (Active Period - AP) ve ölü periyot (Dead Period - DP). Aktif periyot, bir bloğun önbelleğe getirilme anı ile bu bloğa yapılan son erişim arasında geçen zamanı göstermektedir. Ölü periyot ise bir bloğa yapılan son erişim ile bu bloğun yer değiştirme işlemi için kurban blok olarak seçilmesi arasında geçen zamanı göstermektedir. Şekil 6.1'de bu kavramlar belirtilmiştir.



Şekil 6.1 Aktif Periyot (Active Period - AP) ve Ölü Periyot (Dead Period - DP) kavramlarının şekilsel izahı.

AP ve DP kavramlarını temel alarak soft errorları iki gruba ayırabiliriz. Birinci grup aktif periyot süresince oluşan ve yayılan hataları tutan aktif periyot hata katkısı (Active Period Error Contribution - APEC) değeridir. İkincisi ise ölü periyot süresince oluşan ve yayılan hataları ifade eden ölü periyot hata katkısı (Dead Period Error Contribution - DPEC) değeridir. Bu tanımları temel alarak AVFC terimimizi aşağıdaki gibi tekrar ifade edelim (4):

$$AVFC = \frac{APEC + DPEC}{\text{Önbellek Boyutu} \ \mathrm{x} \ \text{Çalıştırma Devir Sayısı}} \qquad (4)$$

6.2 Deneysel Sonuçlar

6.2.1 Performans Kazancı

Çizelge 6.2 Deneylerimizde kullandığımız testlerin önceden getirme işlemi uygulanmadan önceki önemli özellikleri.

Testler	**dl1 Erişim Sayısı**	**dl1 Iska Sayısı**	**Çalışma Devri (Cycle)**
gzip	143557264	1797782 (%1.25)	182541553
vpr	224920169	8576286 (%3.81)	435777993
gcc	363932641	21254160 (%5.84)	205505032
mcf	172902384	14315504 (%8.28)	643453392
crafty	217578341	1341643 (%0.62)	256474649
parser	209547468	3052239 (%1.46)	371981624
eon	218647998	52477 (%0.02)	257583165
gap	185842459	500639 (%0.27)	212196109
bzip2	181840719	2251475 (%1.24)	231748913
twolf	192176856	9734976 (%5.07)	569294807
swim	164490324	14048763 (%8.54)	347981834
mgrid	175826101	6385456 (%3.63)	260406662
applu	175442665	11435581 (%6.52)	261960171
mesa	170935377	558926 (%0.33)	160314218
galgel	232632946	7482263 (%3.22)	191220565
art	191094790	63572117 (%33.27)	630700593
equake	148329761	29822 (%0.02)	131912422
lucas	82871734	6137845 (%7.41)	278925891
sixtrack	170181401	2041991 (%1.20)	195348706
apsi	180915628	7475780 (%4.13)	210605647

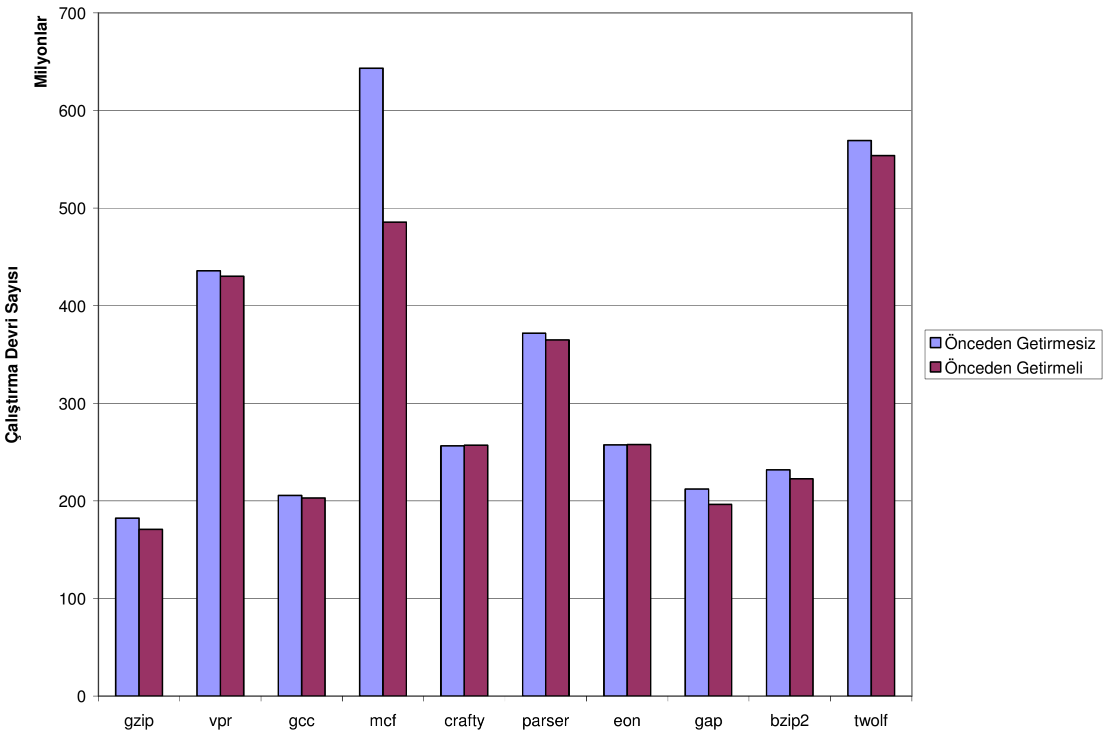
Çizelge 6.2'de testlerin önceden getirme işlemi uygulanmadan önceki veri önbelleği 1 (dl1) erişim ve ıska sayıları ile çalışma devri sayıları verilmiştir. İkinci sütunda dl1 ıska sayılarının yanında dl1 ıska oranları da verilmektedir. Bu sonuçlar temel durum için alınan sonuçlardır.

Çizelge 6.3 Deneylerimizde kullandığımız testlerin önceden getirme işlemi uygulandıktan sonra bazı önemli özellikleri.

Testler	**dl1 Erişim Sayısı**	**dl1 Iska Sayısı**	**Çalışma Devri (Cycle)**
gzip	145386629	1922999 (%1.32)	170888145
vpr	232737495	16497823 (%7.09)	430339082
gcc	385109858	21360028 (%5.55)	202996275
mcf	186391527	19570841 (%10.50)	485602479
crafty	219066280	2911564 (%1.33)	257069106
parser	211260676	4128146 (%1.95)	364782088
eon	218758149	72124 (%0.03)	257774452
gap	186628766	511696 (%0.27)	196396288
bzip2	183605019	3974318 (%2.16)	222590370
twolf	202003115	16907841 (%8.37)	553625913
swim	178513883	14090418 (%7.89)	204045936
mgrid	182069191	6510733 (%3.58)	160995015
applu	183067191	11524904 (%6.30)	205001094
mesa	171633145	844700 (%0.49)	155379594
galgel	240148180	9548326 (%3.98)	145760719
art	243107208	87193897 (%35.87)	598583392
equake	148317630	29889 (%0.02)	131316402
lucas	88521959	8077238 (%9.12)	268165298
sixtrack	171956037	2217406 (%1.29)	193498527
apsi	188313365	13245980 (%7.03)	199453913

Çizelge 6.3 ise testlerin önceden getirme işlemi uygulanarak alınan sonuçları göstermektedir. Çizelgede dl1 erişim ve ıska sayıları ile çalışma devri sayıları verilmiştir. İkinci sütunda dl1 ıska sayılarının yanında dl1 ıska oranları da verilmektedir.

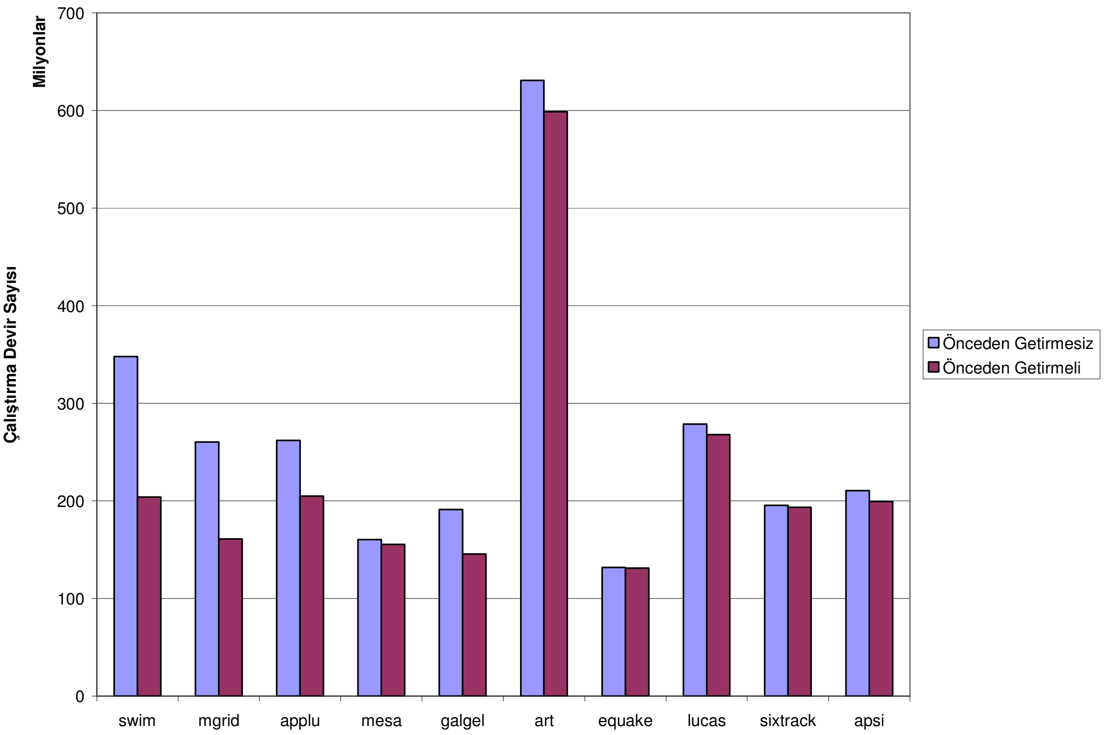
Çizelge 6.2 ve Çizelge 6.3 incelendiğinde önceden getirme işleminin dl1 erişimlerini artırdığını, ıska sayılarını ve paralelinde oranlarını da genel olarak artırdığını, ancak bununla birlikte çalışma devri sayısını ortalama %10 azaltarak performans kazancı sağladığı sonucunu çıkarabiliriz. Bu çalışma devri sayısındaki azalmayı birçok test kullandığımız için bu testleri integer ve float olarak ayrı ayrı grafiklerde belirtiyoruz.



Şekil 6.2 Integer testler için devir sayısı değişimi.

Şekil 6.2'de görülen ikili çubuklardan ilki karşılık gelen testin önceden getirme yapılmadan önceki çalıştırma devri sayısıdır. İkinci çubuk ise önceden getirme kodlarının eklendiği değiştirilmiş simülasyon koduyla yapılan ölçüm sonuçlarını göstermektedir.

Benzer şekilde float testler için çalıştırma devri sayısındaki iyileştirme Şekil 6.3'te verilmiştir.



Şekil 6.3 Float testler için devir sayısı değişimi.

Çalışma zamanı açısından ortalama %10 performans kazancı sadece dl1 önbellekler üzerine odaklandığımız için iyi bir sonuçtur. Tüm önbellekler üzerinde alternatif planların uygulanması ile daha yüksek performans kazancı elde edilmesi mümkündür.

6.2.2 Soft Error Etkisi

Önceden getirme yapmanın soft error oranlarını artırıcı veya azaltıcı etkide bulunup bulunmadığına karar vermek için soft error modelimizde bahsettiğimiz AVFC ölçütlerinin testlerimizde nasıl sonuçlar verdiğini aktaracağız.

Çizelge 6.4 ve Çizelge 6.5 ile testlerin blok bazında AVFC oranları verilmektedir.

Çizelge 6.4 Integer testler için blok bazında AVFC oranları.

Testler		gzip	vpr	gcc	mcf	crafty	parser	eon	gap	bzip2	twolf
AVFC	**Önceden getirme kullanmadan**	0.647	0.361	0.535	0.446	0.232	0.336	0.290	0.918	0.232	0.379
	Önceden getirme kullanarak	0.643	0.228	0.533	0.423	0.184	0.284	0.382	0.903	0.210	0.242

Çizelge 6.5 Float testler için blok bazında AVFC oranları.

Testler		swim	mgrid	applu	mesa	galgel	art	equake	lucas	sixtrack	apsi
AVFC	**Önceden getirme kullanmadan**	0.401	0.274	0.398	0.812	0.086	0.276	0.683	0.578	0.086	0.822
	Önceden getirme kullanarak	0.412	0.319	0.361	0.600	0.227	0.222	0.713	0.581	0.095	0.812

Blok bazında AVFC değerleri hesaplamaları sonucunda örneğin integer testlerde önceden getirmesiz durum için ortalama 0,437 olan AVFC değeri önceden getirmeli durumda ortalama 0,403'e indirgenmiştir. Bu durumda integer testlerde yaklaşık %8, float testlerde yaklaşık %2 olmak üzere ortalama %5'lik bozulabilirlik oranı azalışı söz konusudur. Çizelge 6.4 ve Çizelge 6.5 genel olarak bize önceden önbelleğe veri getirmenin bazı uygulamalar için AVFC üzerinde olumlu etkiye, bazı uygulamalar için ise olumsuz etkiye sebep olduğunu göstermektedir. Bunun başlıca sebebini şöyle açıklayabiliriz. Önceden getirme, değiştirilmiş blokları erkenden L2 önbelleğe yolladığından bu blokların ölü periyotlarını (DP) azaltmakta ve dolayısıyla soft errorlara maruz kalma periyodunu genel olarak azaltmaktadır (AVFC'ye olumlu etki). Öte yandan önceden getirme ile genel olarak blokların aktif periyot süreleri artacağından aktif periyotta soft error yakalama olasılığını da olumsuz yönde artırmaktadır. Dolayısıyla bu iki ters etki sonucunda önbelleğe önceden veri getirme bazı uygulamalar için blok bazında olumlu bazı uygulamalar içinse olumsuz etkiye sahip olabilmektedir.

L1 önbellekten L2 önbelleğe soft error yayılımını azaltmanın bir yolu, kirli bloklar yer değiştirmeye maruz kaldıklarında, kirli bloğun tamamını değil de sadece kirli sözcüklerini L2'ye geri yazmaktır. Böylece kirli blokların temiz sözcüklerinde oluşacak soft errorların L2 önbelleğine yayılımı önlenmiş olacaktır. Bu iş için her bir sözcük için 1 bit kullanılarak, ilgili sözcüğün kirli olup olmadığı kaydı tutulabilir. Bu bite göre, ilgili sözcüğün L2'ye geri yazılıp yazılmayacağına karar verilebilir.

Bu bağlamda Çizelge 6.6 integer testler için sözcük bazında AVFC oranlarını göstermektedir.

Çizelge 6.6 Integer testler için sözcük bazında AVFC oranları.

Testler		gzip	vpr	gcc	mcf	crafty	parser	eon	gap	bzip2	twolf
AVFC	**Önceden getirme kullanmadan**	0.163	0.043	0.134	0.044	0.083	0.085	0.150	0.232	0.058	0.044
	Önceden getirme kullanarak	0.617	0.067	0.268	0.110	0.116	0.140	0.314	0.847	0.175	0.052

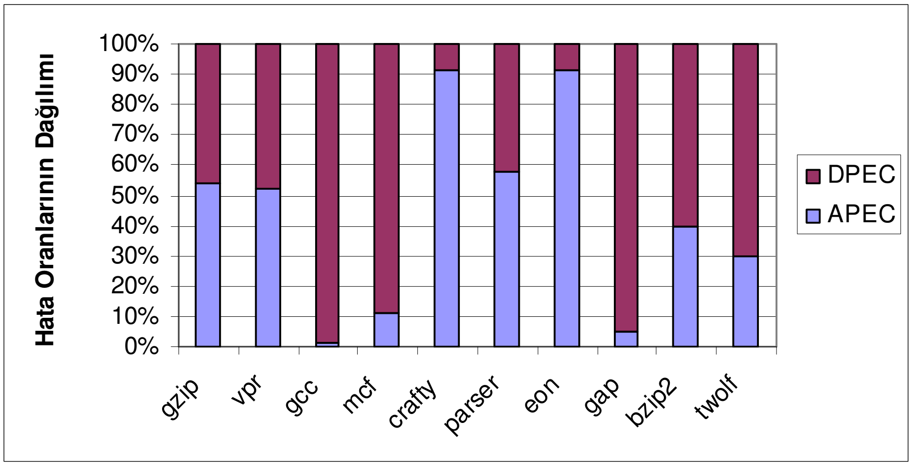
Çizelge 6.6'da önceden getirme yöntemi uygulanmadan ve uygulanarak hesaplanan bozulabilirlik oranları verilmiştir. Bu Çizelgedeki veriler dikkate alınarak yapılan hesaplamamıza göre integer testler için sözcük bazında AVFC oranları ortalama yaklaşık %161 artış göstermiştir.

Çizelge 6.7 Float testler için sözcük bazında AVFC oranları.

Testler		swim	mgrid	applu	mesa	galgel	art	equake	lucas	sixtrack	apsi
AVFC	**Önceden getirme kullanmadan**	0.106	0.083	0.101	0.129	0.059	0.019	0.124	0.131	0.039	0.137
	Önceden getirme kullanarak	0.215	0.178	0.182	0.326	0.207	0.026	0.658	0.393	0.067	0.236

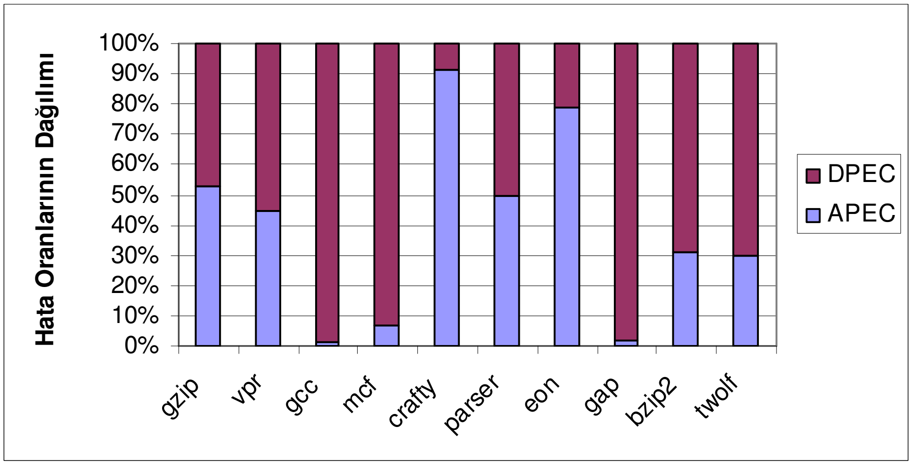
Çizelge 6.7 ile float testler için sözcük bazında hesaplanan AVFC faktörü değerleri verilmektedir. Bu çizelgedeki verilerden hareketle sözcük bazında float testlerde bozulabilirlik faktörünün önceden getirme yapılması durumunda yaklaşık %168 olarak artış gösterdiği bulunmuştur. Sözcük bazında önceden getirme işlemi ortalama yaklaşık %165 AVFC faktörü artışına sebep olmuştur.

6.2.3 APEC ve DPEC Değerlerinin Dağılımları



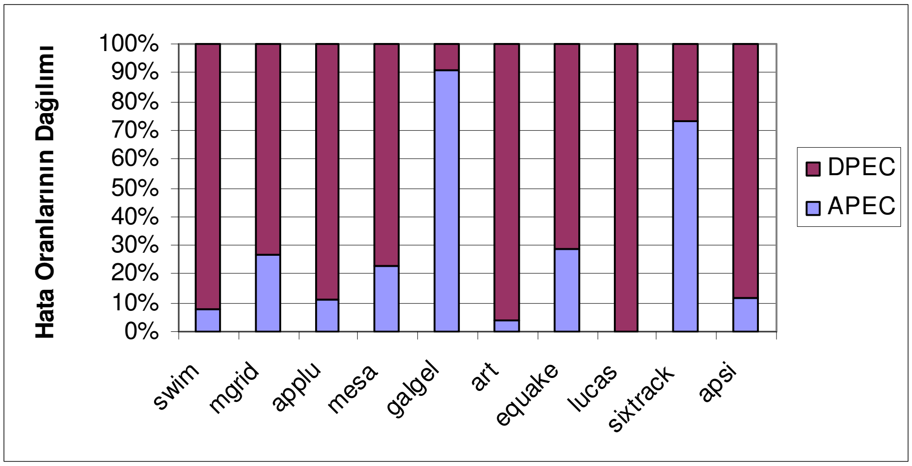
Şekil 6.4 Integer testlerde önceden getirme uygulanmadan AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı.

Şekil 6.4'te her bir çubuk karşılık gelen testte sezilen hataların yüzdelik olarak ne kadarının aktif periyot (AP) süresince ne kadarının ölü periyot (DP) süresince meydana geldiğini belirtmektedir. Önceden getirmesiz AVFC hesabında integer testler için APEC'in AVFC'deki ortalama katkısı yaklaşık %43, DPEC'in AVFC'deki ortalama katkısı yaklaşık %57'dir.

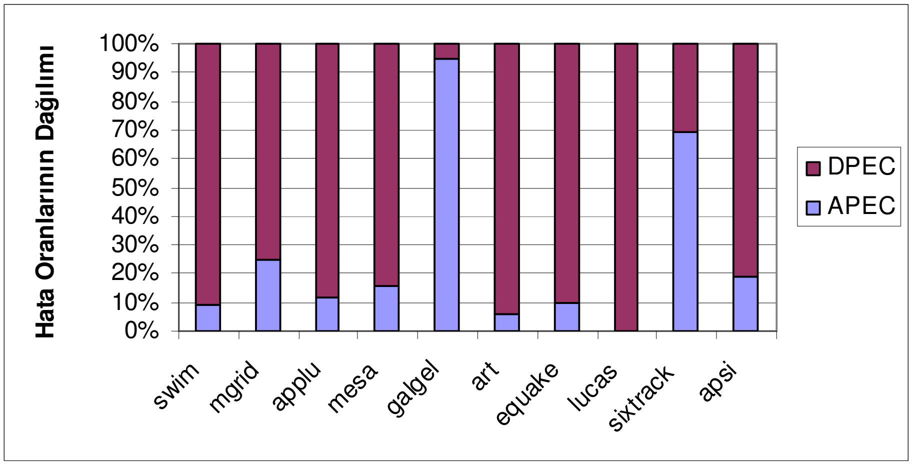


Şekil 6.5. Integer testlerde önceden getirme uygulanarak AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı.

Önceden getirme işlemi yapılarak gerçekleştirilen ölçümlerde integer testler için sözcük bazında AVFC hesabında APEC'in ortalama katkısı yaklaşık %38'e gerilemiş, DPEC'in ortalama katkısı ise yaklaşık %62'ye çıkmıştır (Şekil 6.5).



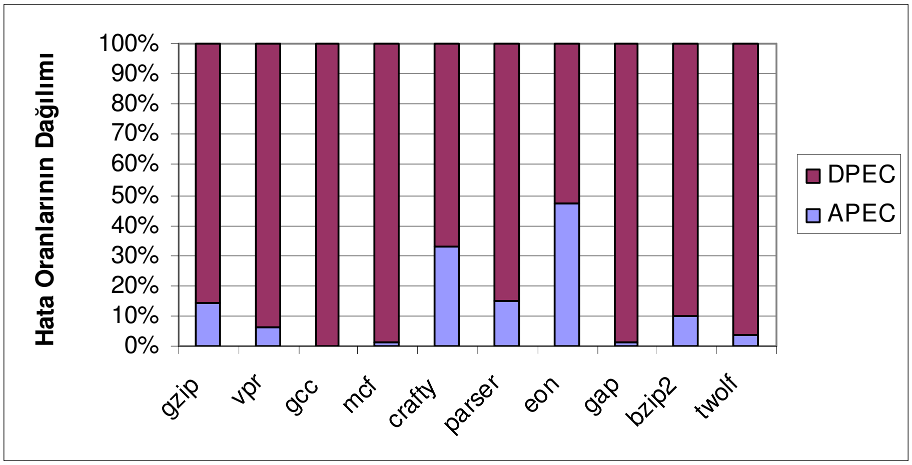
Şekil 6.6. Float testlerde önceden getirme uygulanmadan AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı.



Şekil 6.7. Float testlerde önceden getirme uygulanarak AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı.

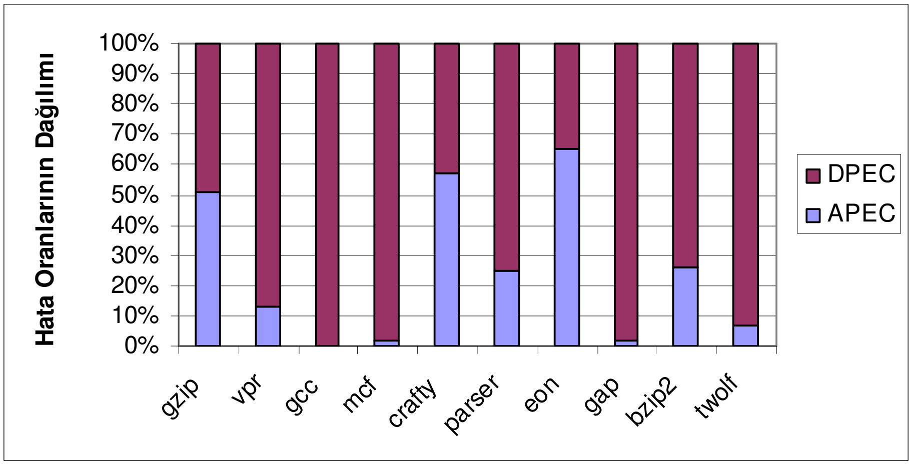
Önceden getirmesiz yapılan float test ölçümlerinde aktif periyot süresince sözcük bazında APEC'in ortalama katkısı yaklaşık %28, DPEC'in ortalama katkısı yaklaşık %72 olarak gerçekleşmiştir (Şekil 6.6). Önceden getirmeli işlemci simülasyonuyla yaptığımız ölçümlerde ise yaklaşık olarak APEC'in ortalama katkısı %26, DPEC'in ortalama katkısı %74 olarak görülmüştür (Şekil 6.7).

Önceden getirmenin soft error oluşumlarına etkisini yorumlayabilmek için APEC ve DPEC hataya katkı yüzdelerinin blok bazında ölçüm sonuçlarını da elde ettik. Şekil 6.8 ve Şekil 6.9 integer testlerde bu sonuçları sırasıyla önceden getirmesiz ve önceden getirmeli olarak vermektedir.



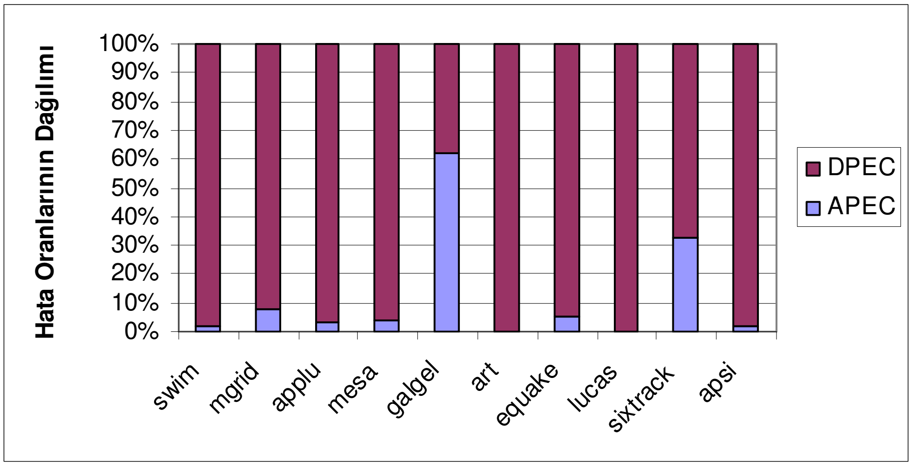
Şekil 6.8 Integer testlerde önceden getirme uygulanmadan AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı

Şekil 6.8'de aktardığımız ölçümlerimizde APEC'in ortalama katkısı yaklaşık %13, DPEC'in ortalama katkısı ise yaklaşık %87 olarak gerçekleşmiştir. Şekil 6.9'da önceden getirme uygulanarak alınan ölçüm sonuçlarında ise APEC'in ortalama katkısı yaklaşık %25'e çıkmış, DPEC'in ortalama katkısı ise %75'e gerilemiştir.



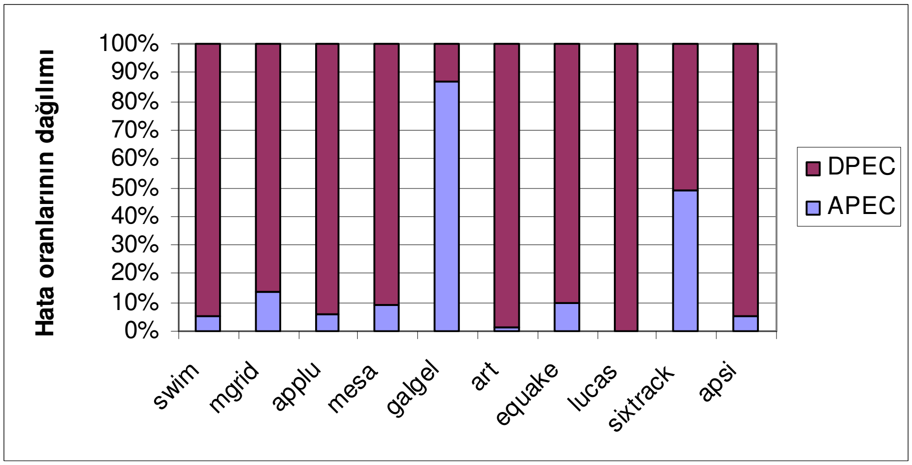
Şekil 6.9. Integer testlerde önceden getirme uygulanarak AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı.

Şekil 6.10 ve 6.11'de float testlerde sırasıyla önceden getirmesiz ve önceden getirmeli durumlar için blok bazında APEC ve DPEC hataya katkı yüzdeleri gösterilmektedir.



Şekil 6.10. Float testlerde önceden getirme uygulanmadan AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı.

Şekil 6.10'da verilen ölçüm sonuçlarına göre APEC'in ortalama katkısı yaklaşık %12, DPEC'in ortalama katkısı ise yaklaşık %88 olarak gerçekleşmiştir. Şekil 6.11'de önceden getirme uygulanarak alınan ölçüm sonuçlarında ise APEC için ortalama katkı yüzdesi yaklaşık %19'a yükselmiş, DPEC için ise bu değer %81'e gerilemiştir.

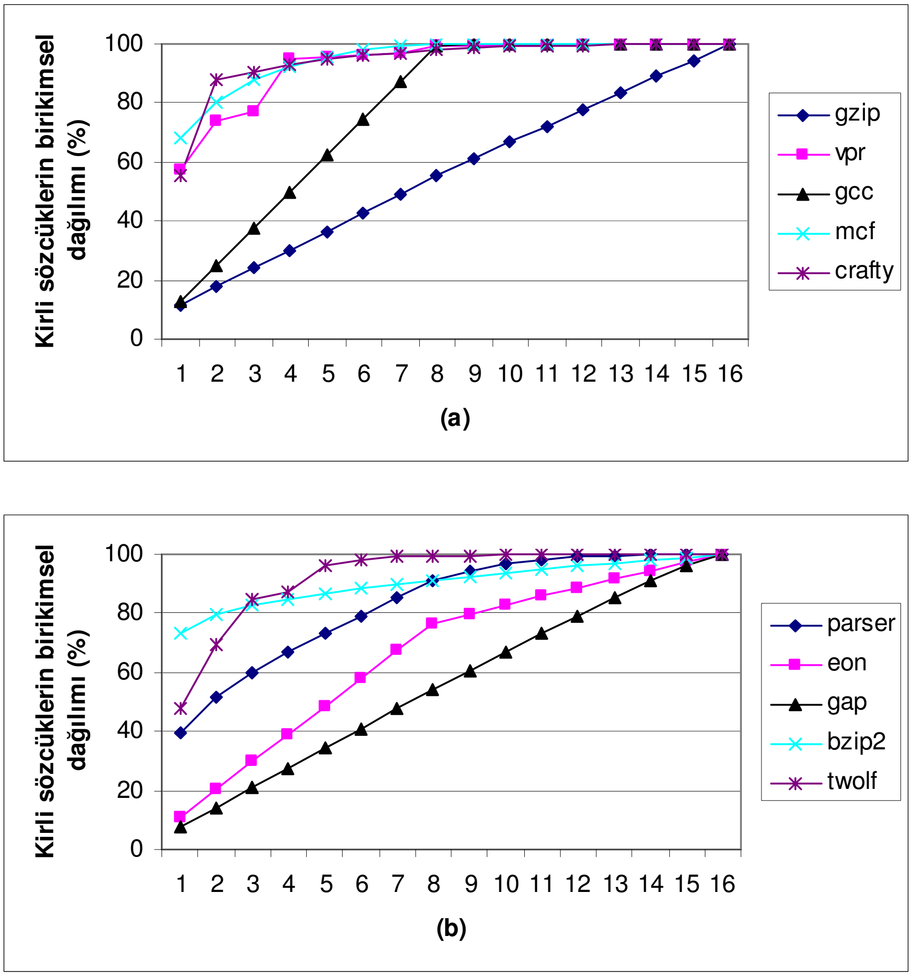


Şekil 6.11. Float testlerde önceden getirme uygulanarak AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı.

6.2.4 Değiştirilmiş Sözcüklerin Birikimsel Dağılımları

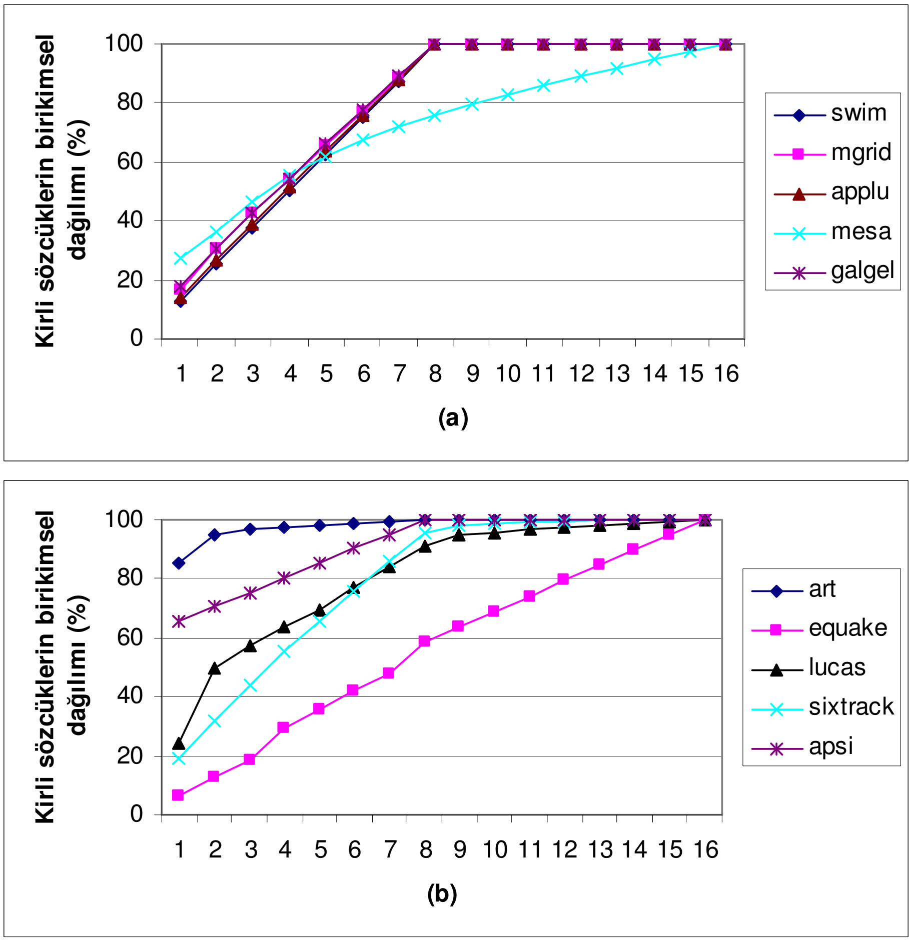
Çizelge 6.4, 6.5, 6.6 ve 6.7'lerden kirli bloklar yer değiştirmeye maruz kaldıklarında tüm bloğun değil de, sadece kirlenmiş bloktaki kirli sözcükleri L2 önbelleğe geri yazmanın AVFC üzerinde oldukça olumlu etkiye sahip olduğu görülmektedir. Bunun sebebini açıklamak için bir dizi deneyler daha yaptık.

Şekil 6.12 ile 6.13'teki testlerde değişikliğe uğramış sözcük sayılarının birikimsel olarak dağılımları verilmektedir. Bu sonuçlar 64 byte blok boyutundaki dl1 önbelleği için alındığından ve biz her sözcüğün 4 byte yer kapladığını varsaydığımızdan dolayı blokların 16 sözcüklük kapasitesi vardır.



Şekil 6.12 64 byte blok boyutuna (her bir sözcük 4 byte) sahip veri önbelleğinde önceden getirme yapılmış durum için kirli bloklardaki değiştirilmiş sözcüklerin sayısının integer testler için birikimsel dağılımı. Aksis değiştirilen sözcüklerin sayısını ifade etmektedir.

Örneğin Şekil 6.12(b)’de gösterilen twolf testinde 4 ve daha az sayıda sözcüğü kirlenmiş blokların oranı yaklaşık %90 olarak gerçekleşirken, benzer olarak Şekil 6.13(b)’de apsi için bu oran yaklaşık %80 olarak tespit edilmiştir. Böylece, kirlenmiş blokta sadece kirli sözcükleri L2 önbelleğine geri yazmanın iyi bir strateji olduğu açıkça görülmektedir.



Şekil 6.13 64 byte blok boyutuna (her bir sözcük 4 byte) sahip veri önbelleğinde önceden getirme yapılmış durum için kirli bloklardaki değiştirilmiş sözcüklerin sayısının float testler için birikimsel dağılımı. Aksis değiştirilen sözcüklerin sayısını ifade etmektedir.

BÖLÜM 7

SONUÇ VE TARTIŞMA

Bu tez kapsamında önceden getirme ile soft errorlar arasındaki ilişkiler araştırılmıştır. Önceden getirme yöntemi olarak donanım tabanlı etiketlemeli önceden getirme yöntemi kullanılmıştır. Bu yöntemin performans ve soft error oluşumlarına etkilerini incelemek amacıyla SPEC2000 testleri üzerinde ölçümler yapılmıştır. Deneysel sonuçlarımız hem önceden getirme uygulanmadan hem de uygulanarak sözcük ve blok bazında soft error etkisini ölçmek için tanımladığımız AVFC ölçütünün kıyaslamalarını içeren analizlerle desteklenmektedir.

Deneylerimizde kullandığımız etiketlemeli önceden getirme yöntemi dl1 önbelleği odaklı geliştirilmiştir. Bu yöntemin kullanılması önceden getirmesiz duruma göre %10 performans kazancı sağlamıştır. Bu yöntemin önbellek yapılarındaki bozulabilirliğe etkisi sözcük ve blok temelli olarak incelenmiştir. Sözcük bazında AVFC değerleri hesaplamaları sonucunda önceden getirme uygulandıktan sonra testlerde yaklaşık ortalama %165 AVFC etkisi artışı gözlenmiştir. Blok bazında yapılan testlerde ise yaklaşık ortalama %5'lik AVFC etkisi azalışı söz konusudur. Önceden getirme, değiştirilmiş blokları erkenden L2 önbelleğe yolladığından bu blokların ölü periyotlarını (DP) azaltmakta ve dolayısıyla soft errorlara maruz kalma periyodunu genel olarak azaltmaktadır. Öte yandan önceden getirme ile genel olarak blokların aktif periyot süreleri artacağından aktif periyotta soft error yakalama olasılığını da olumsuz yönde artırmaktadır. Bu iki ters etki sonucunda önbelleğe önceden veri getirme bazı uygulamalar için blok bazında olumlu bazı uygulamalar içinse olumsuz etkiye sahip olmakla birlikte ortalama olarak bozulabilirliği azaltıcı etki oluşturmuştur.

Değerlendirmemiz, önceden getirmenin, genel olarak, kirli önbellek bloklarının L2 önbelleğine blok temelli geri yazılması durumunda önbelleğin soft errorlara maruz kalma riskini azalttığıdır. Ancak, tüm blok değil de sadece değiştirilmiş sözcüklerin geriye yazılması düşünüldüğünde, bu durumda, genel olarak önceden veri getirmenin soft errorlar üzerinde olumsuz etkiye sebep olduğu gözlenmiştir. Bunun olumsuz etkisini azaltmak için önceden veri getirme işlemini hemen tetiklemek yerine belli bir süre bekleyip ondan sonra başlatmanın etkili olup olamayacağının araştırılması yeni bir çalışma konusu olacaktır.

KAYNAKLAR

Alexander, T., Kedem, G. 1996. Distributed prefetch-buffer/cache design for high performance memory systems. In *Proceedings of 2nd IEEE Symposium on High-Performance Computer Architecture*. IEEE Press, Piscataway, NJ, 254–263.

Anacker, W., Wang, C. P. 1967. Performance evaluation of computing systems with memory hierarchies. *IEEE Trans. Comput. 16*, 6, 764–773.

Bernstein, D., Cohen, D., Freund, A. 1995. Compiler techniques for data prefetching on the PowerPC. In *Proceedings of the IFIP WG10.3 Working Conference on Parallel Architectures and Compilation Techniques* (PACT ’95, Limassol, Cyprus, June 27–29), L. Bic, P. Evripidou, W. Böhm, and J.-L. Gaudiot, Chairs. IFIP Working Group on Algol, Manchester, UK, 19–26.

Burger, D.C., Austin, T. M. 1997. The SimpleScalar Toolset, version 3.0, Tech. Rep. http://www.simplescalar.com

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. SRAM SER 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.

Casmira, J. P., Kaeli, D. R. 1995. Modeling cache pollution. In *Proceedings of the Second IASTED Conference on Modeling and Simulation*. 123–126.

Chan, K. K. 1996. Design of the HP PA 7200 CPU. *Hewlett-Packard J. 47*, 1, 25–33.

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

Fu, B., Saini, A., Gelsinger, P. P. 1989. Performance and microarchitecture of the i486 processor. In *Proceedings of the IEEE International Conference on Computer Design* (Cambridge, MA). 182–187.

Gomaa, M., Scarbrough, C., Vijaykumar, T. N., Pomeranz, I. 2003. Transient-fault recovery for chip multiprocessors. *International Symposium on Computer Architecture.*

Gomaa, M. A., Vijaykumar, T. N. 2005. Opportunistic Transient-Fault Detection. In *Proc. International Symposium on Computer Architecture.*

Handy, J., 1998, *The Cache Memory Book*, Second Edition (The Morgan Kaufmann Series in Computer Architecture and Design), 5-16,54-65.

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

Howe D., 2006, The Free Online Dictionary of Computing (http://foldoc.doc.ic.ac.uk/), http://www.cacs.louisiana.edu/~mgr/404/burks/foldoc/41/32.htm

IBM, 2006, http://www.research.ibm.com/journal/rd/401/curtis.html

Intel, 2006, http://www.intel.com/design/intarch/papers/cache6.pdf

Joseph, D., Grunwald, D. 1997. Prefetching using Markov predictors. In *Proceedings of the 24th International Symposium on Computer Architecture* (ISCA ’97, Denver, CO, June 2–4), A. R. Pleszkun and T. Mudge, Chairs. ACM Press, New York, NY, 252–263.

Jouppi, N. 1990. Improving direct-mapped cache performance by the addition of a small fullyassociative cache and prefetch buffers. In *Proceedings of the 17th International Symposium on Computer Architecture* (ISCA ’90, Seattle, WA, May). IEEE Press, Piscataway, NJ, 364–373.

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

Kim, S., Somani, A. K. 2002. Soft error sensitivity characterization for microprocessor dependability enhancement strategy. In *Proc. International Conference on Dependable Systems and Networks.*

Kim, D. 2004. Compiler-based Pre-Execution. PhD Dissertation (Doktora Tezi). University of Maryland, USA. s:18.

Kumar, S., Aggarwal, A. 2006. Reducing resource redundancy for concurrent error dedection techniques in high performance microprocessors. In *Proc. International Symposium on High-Performance Computer Architecture.*

Li, L., Vijaykrishnan, N., Kandemir, M., Irwin, M. J. 2003. Adaptive error protection for energy efficiency. In *Proc. International Conference on Computer Aided Design.*

Li, L., Degalahal, V., Vijaykrishnan, N., Kandemir, M., Irwin, M. J. 2004. Soft error and energy consumption interactions: a data cache perspective. In *Proc. International Symposium on LowPower Electronics and Design*.

Li, X., Adve, S. V., Bose, P., Rivers, J. A. 2005. SoftArch: an architecture-level tool for modeling and analyzing soft errors. *Dependable Systems and Networks.*

Mowry, T., Gupta, A. 1991. Tolerating latency through software-controlled prefetching in shared-memory multiprocessors. *J. Paralel Distrib. Comput. 12*, 2 (June), 87–106.

Mowry, T. C., Lam, M. S., Gupta, A. 1992. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS-V, Boston, MA, Oct. 12–15), S. Eggers, Chair. ACM Press, New York, NY, 62–73.

Mukherjee, S. S., Weaver, C., Emer, J., Reinhardt, S. K., Austin, T. 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. *International Symposium on Microarchitecture.*

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

Porterfıeld, A. K. 1989. Software methods for improvement of cache performance on supercomputer applications. Ph.D. Dissertation (Doktora Tezi). Rice University, Houston, TX.

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

Ray, J., Hoe, J., Falsafi, B. 2001. Dual use of superscalar datapath for transient-fault dedection and recovery. In *Proc. International Symposium on Microarchitecture*.

Reinhardt, S.K., Mukherjee, S. S. 2000. Transient fault detection via simultaneous multithreading. In Proc. *International Symposium on Computer Architecture*.

Smith, A. J. 1978. Sequential program prefetching in memory hierarchies. *IEEE Computer 11*, 12, 7–21.

Smith, A. J. 1982. Cache memories. *ACM Computing Survey 14*, 3 (Sept.), 473–530.

SPEC, CPU2000 benchmark. 2006. http://www.spec.com

Sun Microsystems, 2006, Memory Hierarchy in Cache-Based Systems, http://www.sun.com/blueprints/1102/817-0742.pdf

Surriel, 2006, http://www.surriel.com/lectures/hierarchy.gif

Tosun, S. 2005. Reliability-Centric System Design for Embedded Systems. PhD Dissertation (Doktora Tezi), Syracuse University, Syracuse, NY. 22-23.

VanderWiel, S.P. 1998. Masking Memory Access Latency With A Compiler-Assisted Data Prefetch Controller. PhD Dissertation (Doktora Tezi). University of Minnesota, USA. 15-20.

VanderWiel, S.P., Lilja, D.J. 2000. Data Prefetch Mechanisms, *ACM Computing Surveys*, Vol. 32, (2): 174-199.

Vijaykumar, T., Pomeranz, I., Cheng, K. 2002. Transient-fault recovery using simultaneous multithreading. In *Proc.International Conference on Computer Architecture.*

Wang, N., Patel, S. 2003. Modeling the effect of transient errors on high performance microprocessors. *Center for Circuits, Systems, and Software*.

Wang, N. J., Quek, J., Rafacz, T. M., Patel, S. J. 2004. Characterizing the effects of transient faults on a high-performance processor pipeline. In *Proc. International Conference 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. In *Proc. International Symposium on Computer Architecture.*

Zhang, W., Gurumurthi, S., Kandemir, M., Sivasubramaniam. 2003. ICR: in-cache replication for enhancing data cache reliability. *International Conference on Dependable Sysstems and Networks.*

Zhang, W. 2004. Enhancing Data Cache Reliability by the Addition of a Small Fully-associative Replication Cache. In *Proc. ACM International Conference on Supercomputing.*

Ziegler, J.F. ve diğ. 1996. IBM experiments in soft fails in computer electronics (1978-1994). *IBM Journal of Research and Development*, vol. 40. no.1, 3-18.

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

Zero, 2006, http://www.zerosoft.com

ÇİZELGELER

Çizelge No **Çizelge Adı** **Sayfa**

Çizelge 2.1 Farklı önbellek haritalama teknikleri ve onların performans kıyaslamaları 13

Çizelge 4.1 Sanayide karşılaşılan soft error durumlarına örnekler 33

Çizelge 6.1 Başlıca yapılandırma parametreleri ve deneylerimizde kullanılan değerleri 41

Çizelge 6.2 Deneylerimizde kullandığımız testlerin önceden getirme işlemi uygulanmadan önceki önemli özellikleri 43

Çizelge 6.3 Deneylerimizde kullandığımız testlerin önceden getirme işlemi uygulandıktan sonra bazı önemli özellikleri 44

Çizelge 6.4 Integer testler için blok bazında AVFC oranları 46

Çizelge 6.5 Float testler için blok bazında AVFC oranları 47

Çizelge 6.6 Integer testler için sözcük bazında AVFC oranları 47

Çizelge 6.7 Float testler için sözcük bazında AVFC oranları 48

ŞEKİLLER

Şekil No **Şekil Adı** **Sayfa**

Şekil 2.1 Örnek bir önbellek temelli bellek sistemi 4

Şekil 2.2 Bellek hiyerarşisi 5

Şekil 2.3 Doğrudan haritalanmış önbellek 10

Şekil 2.4 Tam çağrışımlı önbellek 11

Şekil 2.5 4-yollu küme çağrışımlı tasarım... 12

Şekil 2.6 2-yollu küme çağrışımlı önbellek.. 12

Şekil 3.1 Üç duruma göre çalıştırma diyagramları: **(a)** önceden getirmesiz, **(b)** mükemmel önceden getirmeli ve **(c)** düşük verimli önceden getirmeli.............. 17

Şekil 3.2 **(a)** Önceden getirmesiz, **(b)** basit önceden getirmeli, **(c)** adım atlamalı önceden getirmeli ve **(d)** yazılım iş hattı (software pipelining) teknikleri ile döngüsel hesaplamalar... 19

Şekil 3.3 Bir sonraki bloğu önceden getirme işleminin 3 yöntemi: **(a)** Iska olduğunda önceden getirme, **(b)** Etiketlemeli önceden getirme, **(c)** Sonraki iki bloğu tetiklemeli önceden getirme (K=2)... 21

Şekil 3.4 Referans Tahmin Tablosunun Yapısı 24

Şekil 4.1 Aygıt ve devre üzerinde tanecik darbesiyle soft error oluşumu................ 26

Şekil 4.2 Kozmik ışın akımının yüksekliklere göre değişimi................................. 27

Şekil 4.3 Bozulan bir bitin neden olabileceği mümkün çıktıların sınıflandırılması .. 29

Şekil 5.1 Bir kaynak kod parçası (üstte) ve karşılık gelen makine dili kodu (altta) . 39

Şekil 5.2 Şekil 5.1'deki kod parçasının veri akış grafı... 39

Şekil 5.3 Şekil 1'deki kod parçası tarafından işlenen veriler için önbelleğe ziyaretlerini içeren zaman çizelgesi 39

Şekil 6.1 Aktif Periyot (Active Period - AP) ve Ölü Periyot (Dead Period – DP) kavramlarının şekilsel izahı 42

Şekil 6.2 Integer testler için devir sayısı değişimi 45

Şekil 6.3 Float testler için devir sayısı değişimi 46

Şekil 6.4 Integer testlerde önceden getirme uygulanmadan AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı 49

Şekil 6.5 Integer testlerde önceden getirme uygulanarak AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı 49

Şekil 6.6 Float testlerde önceden getirme uygulanmadan AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı 50

Şekil 6.7 Float testlerde önceden getirme uygulanarak AVFC hesabında sözcük bazında APEC ve DPEC değerlerinin dağılımı 50

Şekil 6.8 Integer testlerde önceden getirme uygulanmadan AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı 51

Şekil 6.9 Integer testlerde önceden getirme uygulanarak AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı 51

Şekil 6.10 Float testlerde önceden getirme uygulanmadan AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı 52

Şekil 6.11 Float testlerde önceden getirme uygulanarak AVFC hesabında blok bazında APEC ve DPEC değerlerinin dağılımı 52

Şekil 6.12 64 byte blok boyutuna (her bir sözcük 4 byte) sahip veri önbelleğinde önceden getirme yapılmış durum için kirli bloklardaki değiştirilmiş sözcüklerin sayısının integer testler için birikimsel dağılımı. Aksis değiştirilen sözcüklerin sayısını ifade etmektedir... 53

Şekil 6.13 64 byte blok boyutuna (her bir sözcük 4 byte) sahip veri önbelleğinde önceden getirme yapılmış durum için kirli bloklardaki değiştirilmiş sözcüklerin sayısının float testler için birikimsel dağılımı. Aksis değiştirilen sözcüklerin sayısını ifade etmektedir... 54

YAŞAM ÖYKÜSÜ

Adı Soyadı: Olcay KABAL
Doğum Yeri ve Yılı: Selçuk / İZMİR - 19.09.1982
Adres: Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü,
Terzioğlu Yerleşkesi, 17100 ÇANAKKALE
E-Posta: okabal@yahoo.com

Eğitim Durumu

2004-2007: Yüksek Lisans / Çanakkale Onsekiz Mart Üniversitesi, Fen Bilimleri Enstitüsü, Bilgisayar Mühendisliği Anabilim Dalı
2000-2004: Lisans / Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü
1996-2000: Selçuk Y.D.A. (İng.) Lisesi, Selçuk / İZMİR
1988-1996: İsabey İlköğretim Okulu, Selçuk / İZMİR

Stajlar

- Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü, 2002.
- Türk Hava Yolları A.O. EBİ (Elektronik Bilgi İşlem) Bşk., İstanbul, 2003.

Kurslar

- ÇOMÜ YADEM Almanca Dil Kursu (C) (2001 Ekim – 2002 Ocak)

Projeler

Çanakkale Onsekiz Mart Üniversitesi Bilgisayar Mühendisliği Bölümü tarafından 2005 yılında yapılan "Çanakkale Belediyesi e-Belediye Sistemi Analizi ve Teknik Şartname Hazırlanması" projesinde Teknik Eleman olarak görev aldım.

Çanakkale Onsekiz Mart Üniversitesi Bilimsel Araştırma Projeleri Komisyonu tarafından desteklenen "IPv4 Ağlarının IPv6 Ağlarına Entegrasyonu" BAP 2006-20 nolu projede Araştırıcı olarak yer aldım.

Mesleki Deneyim

- 2004 - Çanakkale Onsekiz Mart Üniversitesi Bilgisayar Mühendisliği Bölümü Araştırma Görevlisi.

Çalışma ve İlgi Alanları

- Bilgisayar Mimarisi,
- Yazılım Mühendisliği,
- İşletim Sistemleri,
- Bilgisayar Ağları.

Aldığı Ödül ve Dereceler

- Çanakkale Onsekiz Mart Üniversitesi 2004 Yılı Üniversite Birinciliği,
- Çanakkale Onsekiz Mart Üniversitesi IX. Bahar Şenliklerinde (2004) Mühendislik – Mimarlık Fakültesi Basketbol takımı İkinciliği (oyuncu olarak),
- Çanakkale Onsekiz Mart Üniversitesi X. Bahar Şenliklerinde (2005) Mühendislik – Mimarlık Fakültesi Basketbol takımı Üçüncülüğü (idareci olarak),
- 26 Nisan 2000'de yapılan 5. İzmir Liselerarası Bilgi ve Kültür Yarışması finalinde İkinci olan Selçuk Lisesi yarışma ekibinde yer aldım.