ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ÖNBELLEKLERDE KULLANILAN ENERJİNİN OPTİMİZASYONU VE ENERJİ KAYBININ AZALTILMASI

Ayhan ZORLUBAŞ

Yrd. Doç. Dr. İsmail KADAYIF

Ağustos, 2006
ÇANAKKALE

ÖN BELLEKLERDE KULLANILAN ENERJİNİN OPTİMİZASYONU VE ENERJİ KAYBININ AZALTILMASI

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

Hazırlayan
Ayhan ZORLUBAŞ

Danışman
Yrd.Doç.Dr. İsmail KADAYIF

Ağustos, 2006
ÇANAKKALE

Çanakkale Onsekiz Mart Üniversitesi Fen Bilimleri Enstitüsü Müdürlüğüne,

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

Başkan : Yrd. Doç. Dr. İsmail KADAYIF

Üye : Doç. Dr. Mehmed Ali SALAHLI

Üye : Yrd. Doç. Dr. İbrahim TÜRKYILMAZ

Üye :

Üye :

Kod No:

Yukarıdaki imzaların adı geçen öğretim üyelerine ait olduğunu onaylarım.

Enstitü Müdürü

TEŞEKKÜR

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

Manevi desteğini benden hiçbir zaman esirgemeyen aileme ve eşim Derya ERGÜN ZORLUBAŞ'a;

Bana sürekli olarak destek olan çalışma arkadaşlarım Ör.Gör. Bora UĞURLU ve Okt. Mehmet ULUTAŞ'a;

Deneyimlerini ve bilgilerini benden esirgemeyen Bil.Müh. Selçuk KOYUNCU'ya sonsuz teşekkürlerimi sunarım.

Ayhan ZORLUBAŞ
2006

SİMGELER VE KISALTMALAR

ACC (Asymmetric Cell Cache): Asimetrik hücreli önbellek

ALU (Arithmetic Logic Unit): Aritmetik mantık birimi

BG: Bütün güç durumlarının birleşimi

C: Sığa

CPU (Central Processing Unit): Merkezi İşlem Birimi

D: Yarı iletken transistörün iletim zamanı

D-Cache (Data Cache): Veri önbelleği

DK: Durum koruyucu

DRAM (Dynamic RAM): Dinamik RAM

DY: Durum yok edici

ENIAC (Electronic Numeral Integrator and Computers): Elektronik sayısal toplayıcı ve bilgisayarlar

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

FP: Floating pointer

IAS (Institute for Advanced Study Machine): Gelişmiş çalışmalar makinesi enstitüsü

IBR (Instruction Buffer Register): Komut tampon yazmacı

I-Cache (Instruction Cache): Komut Önbelleği

ITRS (International Technology Roadmap for Semiconductors): Yarı iletkenler için uluslar arası teknoloji yol haritası

K: Sızıntı faktörü

K-M: Kestirimci-Melez

L1 (Level 1): Birinci seviye

L2 (Level 2): İkinci seviye

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

LSQ (Load Store Queue): Yükleme kaydetme kuyruğu

mJ: Millijoule

MOS (Metal Oxcide Semiconductor): Metal oksit yarı iletken

nJ: Nanojoule

nm: Nanometre

NS: Nanosaniye

OBL (One Block Lookahead): Bir bloğu önceden getirme

PC: Program Counter

P_d: Dinamik güç

PJ: Petajoule

RAM (Random Access Memory): Rasgele erişimli bellek

RPT (Reference Prediction Table): Referans tahmin tablosu

S-DK: Spekülatif durum koruyucu

S-DK-U: Spekülatif - durum koruyucu – uyuşuk

SRAM (Static RAM): Statik RAM

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

U-DY: Uyuşuk - Durum yok edici

V: Volt

V_{dd}: Besleme gerilimi

VLSI (Very Large Scale Integration): Geniş Ölçekte Tümleşme

V_{th}: Transistör eşik gerilimi

W: Güç

α: Etkinlik faktörü

ÖN BELLEKLERDE KULLANILAN ENERJİNİN OPTİMİZASYONU VE ENERJİ KAYBININ AZALTILMASI

ÖZET

Daha önceden yapılmış olan çalışmaların çoğunda önbellekler için performans ve enerji optimizasyonları konularına bir arada bakılmamıştır. Genel olarak, performansa dayalı teknikler enerji tüketimine etki etmiş, enerjiye dayalı tekniklerde program çalışma süresini arttırmaktadır. Gömülü sistemlerde çoklu tekniklerin bir arada kullanılmasıyla önbelleklerin toplam enerji ve performans davranışları açık bir araştırma konusudur. Yapılan bu tez çalışmasında, bu konular üzerine odaklanılmış; performans ve enerji optimizasyonlarının birbirlerini nasıl etkilediği gösterilmiştir. Daha sonra önceden getirme işlemine duyarlı önbellek bloklarının enerjilerinin kesilmesine yönelik üç ayrı optimizasyon yöntemi önerilmiştir. Özellikle, bu optimizasyon yaklaşımları önceden getirilmiş bloklara, normal yoldan önbelleğe getirilen bloklardan farklı davranmıştır. Önerilen yöntemlerin, SPEC2000'den rasgele seçilen beş tane uygulama üzerinde yapılan testlerinde, sızıntı enerjiyi önemli oranda azalttığı gösterilmiştir. Yapılan testlerin sonuçlarına bakıldığında, sağlanan enerji tasarrufunun sistem performansını çok az miktarda etkilediği görülmüştür.

Anahtar Sözcükler: Önbellekler, Önceden Getirme İşlemi, Sızıntı Enerji

ENERGY OPTIMIZATION IN CACHE MEMORY AND REDUCING CACHE LEAKAGE POWER

ABSTRACT

In general, performance-oriented techniques influence energy consumption and energy-oriented techniques increase program execution cycles. The overall energy and performance behavior of caches in embedded systems when multiple techniques co-exist remains an open research problem. In this study, it's focused on these subjects and demonstrated how performance and energy optimizations affect each other. Then we proposed three optimization schemes that turn off cache lines in a prefetching-sensitive manner. Specifically, these schemes treat prefetched cache lines differently from the lines brought to the cache in a normal way. It has been indicated that proposed methods decrease leakage energy considerably in performed tests on five randomly-selected codes from the SPEC2000 simulator. When the results of the performed tests evaluated, it has been observed that the performance overheads introduced by these schemes are negligible.

Keywords: Caches, Prefetching, Leakage Energy

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

Ayhan ZORLUBAŞ, tarafından **Yrd.Doç.Dr. İsmail KADAYIF** yönetiminde hazırlanan **"Ön Belleklerde Kullanılan Enerjinin Optimizasyonu Ve Enerji Kaybının Azaltılması**" başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

.....

Yönetici

.....

Jüri Üyesi

.....

Jüri Üyesi

Müdür
Fen Bilimleri Enstitüsü

TEŞEKKÜR

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

Manevi desteğini benden hiçbir zaman esirgemeyen aileme ve eşim Derya ERGÜN ZORLUBAŞ'a;

Bana sürekli olarak destek olan çalışma arkadaşlarım Ör.Gör. Bora UĞURLU ve Okt. Mehmet ULUTAŞ'a;

Deneyimlerini ve bilgilerini benden esirgemeyen Bil.Müh. Selçuk KOYUNCU'ya sonsuz teşekkürlerimi sunarım.

Ayhan ZORLUBAŞ
2006

SİMGELER VE KISALTMALAR

ACC (Asymmetric Cell Cache): Asimetrik hücreli önbellek

ALU (Arithmetic Logic Unit): Aritmetik mantık birimi

BG: Bütün güç durumlarının birleşimi

C: Sığa

CPU (Central Processing Unit): Merkezi İşlem Birimi

D: Yarı iletken transistörün iletim zamanı

D-Cache (Data Cache): Veri önbelleği

DK: Durum koruyucu

DRAM (Dynamic RAM): Dinamik RAM

DY: Durum yok edici

ENIAC (Electronic Numeral Integrator and Computers): Elektronik sayısal toplayıcı ve bilgisayarlar

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

FP: Floating pointer

IAS (Institute for Advanced Study Machine): Gelişmiş çalışmalar makinesi enstitüsü

IBR (Instruction Buffer Register): Komut tampon yazmacı

I-Cache (Instruction Cache): Komut Önbelleği

ITRS (International Technology Roadmap for Semiconductors): Yarı iletkenler için uluslar arası teknoloji yol haritası

K: Sızıntı faktörü

K-M: Kestirimci-Melez

L1 (Level 1): Birinci seviye

L2 (Level 2): İkinci seviye

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

LSQ (Load Store Queue): Yükleme kaydetme kuyruğu

mJ: Millijoule

MOS (Metal Oxcide Semiconductor): Metal oksit yarı iletken

nJ: Nanojoule

nm: Nanometre

NS: Nanosaniye

OBL (One Block Lookahead): Bir bloğu önceden getirme

PC: Program Counter

P_d: Dinamik güç

PJ: Petajoule

RAM (Random Access Memory): Rasgele erişimli bellek

RPT (Reference Prediction Table): Referans tahmin tablosu

S-DK: Spekülatif durum koruyucu

S-DK-U: Spekülatif - durum koruyucu – uyuşuk

SRAM (Static RAM): Statik RAM

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

U-DY: Uyuşuk - Durum yok edici

V: Volt

V_{dd}: Besleme gerilimi

VLSI (Very Large Scale Integration): Geniş Ölçekte Tümleşme

V_{th}: Transistör eşik gerilimi

W: Güç

α: Etkinlik faktörü

ÖN BELLEKLERDE KULLANILAN ENERJİNİN OPTİMİZASYONU VE ENERJİ KAYBININ AZALTILMASI

ÖZET

Daha önceden yapılmış olan çalışmaların çoğunda önbellekler için performans ve enerji optimizasyonları konularına bir arada bakılmamıştır. Genel olarak, performansa dayalı teknikler enerji tüketimine etki etmiş, enerjiye dayalı tekniklerde program çalışma süresini arttırmaktadır. Gömülü sistemlerde çoklu tekniklerin bir arada kullanılmasıyla önbelleklerin toplam enerji ve performans davranışları açık bir araştırma konusudur. Yapılan bu tez çalışmasında, bu konular üzerine odaklanılmış; performans ve enerji optimizasyonlarının birbirlerini nasıl etkilediği gösterilmiştir. Daha sonra önceden getirme işlemine duyarlı önbellek bloklarının enerjilerinin kesilmesine yönelik üç ayrı optimizasyon yöntemi önerilmiştir. Özellikle, bu optimizasyon yaklaşımları önceden getirilmiş bloklara, normal yoldan önbelleğe getirilen bloklardan farklı davranmıştır. Önerilen yöntemlerin, SPEC2000'den rasgele seçilen beş tane uygulama üzerinde yapılan testlerinde, sızıntı enerjiyi önemli oranda azalttığı gösterilmiştir. Yapılan testlerin sonuçlarına bakıldığında, sağlanan enerji tasarrufunun sistem performansını çok az miktarda etkilediği görülmüştür.

Anahtar Sözcükler: Önbellekler, Önceden Getirme İşlemi, Sızıntı Enerji

ENERGY OPTIMIZATION IN CACHE MEMORY AND REDUCING CACHE LEAKAGE POWER

ABSTRACT

In general, performance-oriented techniques influence energy consumption and energy-oriented techniques increase program execution cycles. The overall energy and performance behavior of caches in embedded systems when multiple techniques co-exist remains an open research problem. In this study, it's focused on these subjects and demonstrated how performance and energy optimizations affect each other. Then we proposed three optimization schemes that turn off cache lines in a prefetching-sensitive manner. Specifically, these schemes treat prefetched cache lines differently from the lines brought to the cache in a normal way. It has been indicated that proposed methods decreases leakage energy considerably in performed tests on five randomly-selected codes from the SPEC2000 simulator. When the results of the performed tests evaluated, it has been observed that the performance overheads introduced by these schemes are negligible.

Keywords: Caches, Prefetching, Leakage Energy

İÇERİK

Sayfa

BÖLÜM 1

GİRİŞ

Günümüzde Moore kanunlarına dayanarak işlemcilerdeki transistör sayısı her geçen yıl katlanarak artmaktadır. Teknolojik ilerlemeler sonucu transistörlerin kanal genişliği daha da küçülmektedir. Bu teknolojik gelişmeler sonucu devamlı olarak işlemci içerisindeki transistör sayısı ve performansı artmaktadır. Ancak aynı zamanda mikroişlemciler ve bellek performansı giderek asimetrik olarak gelişmiştir. Bu asimetrik gelişme performans boşluğunun da giderek artmasına neden olmuştur. Mikroişlemci ve bellek teknolojisindeki gelişmeler aynı miktarda olmadığından dolayı mikroişlemci bir istekte bulunduğu zaman bellekten bu bilginin gelmesini beklemektedir. Bu da, performans açısından kısıtlayıcı bir darboğaz oluşmasına neden olmaktadır. Bu problemle baş etmek için önbellek ismini verdiğimiz yapılar kullanılmaya başlanmıştır. Önbellekler, statik RAM'lerden yapıldığından dolayı çok hızlı çalışmaktadır. Bu hız sayesinde mikroişlemciler ile bellekler arasındaki performans boşluğunun önüne geçilmeye çalışılmıştır.

Mikroişlemci mimarisindeki gelişmelere paralel olarak, işlemcilerde kullanılan güçte giderek artmaktadır. Yapılan araştırmalara göre, işlemcilerde kullanılan enerjinin %50-60'nı önbelleklerin kullandığı tespit edilmiştir (Montenaro ve diğ., 1996). Dolayısıyla önbellekler üzerinde gerçekleştirilecek olan güç optimizasyonları harcanan toplam enerjinin azaltılmasını sağlayacaktır.

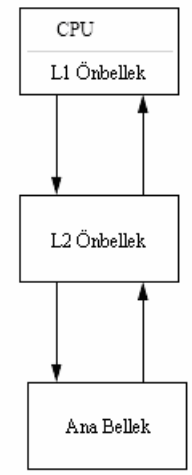
Bu tezde yapılan çalışmayla önbellekler üzerinde iki ana amaç gerçekleştirilmeye çalışılacaktır. İlki, belirgin bir performans iyileştirmesi (önceden getirme işlemi yardımıyla) ve güç iyileştirmesi (önbellek bloklarının enerjilerinin kesilmesiyle) üzerine odaklanmak ve bu iki optimizasyonun performans ve güç açısından birbirlerini nasıl etkilediklerini vurgulamaktır. İkincisi ise, donanıma dayalı üç yeni önceden getirme tekniği gerçekleştirilmesi amaçlanmaktadır. Elde edilen sonuçlar, ileride mikroişlemcilerin performanstan fazla ödün vermeksizin daha az enerjiyle çalışabileceğini göstermektedir.

BÖLÜM 2

ÖNBELLEKLERE GENEL BAKIŞ

Bilgisayarın ana belleğindeki veri ve komutların bir bölümünü geçici olarak depolamak amacıyla kullanılan, diğer belleklere göre çok daha kısa sürelerde erişilebilen küçük kapasiteli belleklere önbellek (cache) denilmektedir (Haraszti, 2002). Önbellekler işlemci tarafından sık sık erişilen veri ve komutları saklamak amacıyla kullanılır. Cache kelimesi, anlamı "saklamak" olan Fransızca "cachet" kelimesinden gelmektedir. İşlemci ve ana bellek arasındaki boş alanda bir köprü görevi gördüğü için çok önemlidir (Stacpoole ve Jamil, 2000).

Önbellekler SRAM olarak adlandırdığımız statik RAM'lerden yapılmışlardır. Ana belleğin yapımında kullanılan dinamik RAM'lerle kıyaslandığında, statik RAM'ler çok daha hızlı çalışmaktadır; fakat buna karşın daha pahalı ve çok daha fazla enerji tüketmektedir. Şekil 2.1'de görüldüğü gibi tipik bir bellek hiyerarşisi iki seviyeden oluşmaktadır. İlk seviye, işlemciyle aynı çip alanı üzerinde bulunan dahili önbellek (Level 1 - L1), ikinci seviye ise ayrık statik RAM çiplerinden ve bir önbellek denetleyicisinden oluşan harici önbellek'tir (Level 2 – L2). Dahili önbellek harici önbelleğe göre daha büyük transfer hızına sahiptir, ama boyut olarak daha küçüktür. Araştırmalar çoklu önbellek seviyesinin modern bilgisayar sistemlerinde uygun performansı sağladığını göstermektedir (Su, 1995).



Şekil 2. 1 Bilgisayar sistemlerindeki bellek hiyerarşisi (Su, 1995)

2.1 Önbelleklerin Tarihçesi

Tarihte yapılmış ilk genel amaçlı bilgisayar olan ENIAC'da önbellek kullanılmıyordu. Sadece veri ve komutları saklamak amacıyla bir ana bellek bulunuyordu. Ama sonradan ENIAC projesinin de danışmanı olan John Von Neumann tarafından 1946 yılında Princeton Enstitüsünde IAS isminde bir bilgisayarın yapım çalışmalarına başlanıldı. Bu bilgisayarda da önbellek bulunmuyordu; fakat ana bellekten getirilen komutların geçici olarak tutulması amacıyla IBR (Instruction buffer register) olarak adlandırılan komut arabellek yazmacı kullanılıyordu (Stallings, 2003). Önbellek olarak adlandırabileceğimiz ilk yapı Harvard Üniversitesinde Howard H. Aiken yönetiminde yapılan MARK III (Eylül 1949) ve MARK IV (1952) isimli bilgisayarlarda kullanılmıştır. Bu iki bilgisayarda komutlar ve veriler için ayrı bellekler yer almıştır. Harvard mimarisi olarak adlandırılan bu mimari çoğu makineye ayrı ayrı iki bellek kullanılması gerektiğini göstermiştir (Abd-El-Barr ve El-Rewini, 2005).

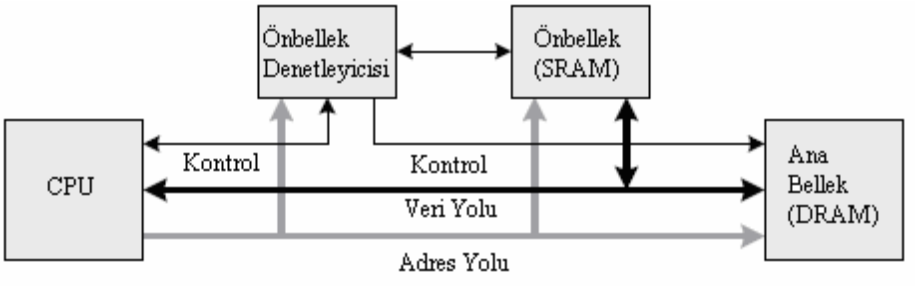
2.2 Önbelleklerin Yapısı ve Çalışma Şekli

Tümleşik devre teknolojisinin ilk uygulaması işlemcinin yapımında kullanılmasıdır. Ama daha sonradan bu teknolojinin bellek üretiminde de kullanılabileceği görülmüştür. 1950 ve 1960'larda çoğu bilgisayar belleği ince ferromanyetik (mıknatıssal) malzemeden halkalar kullanılarak yapılmıştır. Çekirdek (core) ismi verilen bu halkalar mıknatıslanarak bir veya sıfır değerlerini alıyorlardı. Manyetik çekirdek bellek oldukça hızlıydı; ana bellekte kayıtlı bir biti okuması saniyenin milyonda biri sürede gerçekleşiyordu. Ama oldukça pahalı olması, çok alan kaplıyor ve yıkıcı okuma yapıyor olması dezavantajlarıydı. Yani ana bellekten bir veriyi okuduğu anda, o verinin silinmesine neden oluyordu. Sonra 1974 yılında, Fairchild öncekilere göre göreceli olarak daha verimli yarı iletken bellekleri üretti. Bu bellek tek bir çekirdek halka boyutundaydı ve aynı anda ana bellekten 256 bit veri okuyabiliyordu. Bu önbellek yapısı öncekilere göre yıkıcı okuma yapmıyordu ve çok daha fazla hızlıydı. Ama fiyat olarak oldukça pahalıydı. Ancak sonradan yarı iletken teknolojisindeki gelişmeler fiyatının düşmesine neden oldu (Stallings, 2003).

Mikroişlemciler ve bellek performansı giderek asimetrik olarak gelişmiştir. Bu asimetrik gelişme performans boşluğunun da giderek artmasına neden olmuştur. Günümüzde ise mikroişlemciler binlerce megahertz hıza ulaşmaktadır, fakat dinamik RAM'lerin erişim zamanı ancak 40ns'ye kadar düşebilmiştir. Mikroişlemci ve bellek teknolojisindeki gelişmeler aynı miktarda olmadığından dolayı mikroişlemci bir istekte bulunduğu zaman bellekten bu bilginin gelmesini beklemektedir. Bu da performans açısından kısıtlayıcı bir darboğaz oluşmasına neden olmaktadır. Bu problem kısmen statik RAM kullanılarak çözülmüştür. Ama statik RAM kullanılsa da adres çözümlemesinde hala beklemeler yaşanmaktadır. Ancak statik RAM kullanmak bütün bellekler için uygun olmamaktadır. Çünkü statik RAM'lar daha pahalı, daha fazla aygıt gereksinimi, çip alanı ve bağlantı kablosuna gereksinim duymaktadır. Bütün bu gereksinimler maliyetin artmasına ve sistemin güvenilirliğinin azalmasına sebep olmaktadır. Bu sebeple ana belleğin yapısında dinamik RAM, önbelleklerde ise statik RAM kullanılmaya başlanmıştır.

Eğer mikroişlemci üzerinde çalışan bir program ana bellek üzerinde her zaman aynı konuma erişmeye çalışsaydı, bellek bant genişliğinin verimli kullanılmamasına neden olurdu. Neyse ki; yazılımlar, yeterli miktarda kısıtlanmış komutlar kümesine ve veriye, bir sonraki bellek konumuna erişme olasılığını arttırmak ve tamamen rasgele erişimlerin olasılığını azaltmak için verilen zaman içerisinde veriye erişmeye çalışır. Bu özelliğe lokalite (locality) denilmektedir. Komutlar bellekte kayıtlı oldukları sırada yürütülmeye çalışırlar.

Önbellekler ana bellek gecikme problemlerinin üstesinden gelinmesini sağlamaktadır. Daha önceden belirttiğimiz gibi önbellekler, mikroişlemcinin sık sık eriştiği ya da gelecekte erişme ihtimali olan verileri geçici olarak tutan, oldukça küçük ve hızlı belleklerdir. Basitliği ve hızlı erişim zamanından dolayı statik RAM'lardan yapılmaktadır. Şekil 2.2'de görüldüğü gibi bir önbellek, mikroişlemci ve ana bellek arasında durmaktadır ve iki basit elemandan oluşmaktadır: önbellek (cache memory) ve önbellek denetleyicisi.



Şekil 2. 2 Bilgisayardaki önbellek mimarisi (Balch, 2003)

Önbellekler, komut önbelleği (I-cache – Instruction Cache) ve veri önbelleği (D-cache – Data Cache) olmak üzere iki kısımdan oluşur. Komut önbelleği mikroişlemci tarafından istenilen veya istenilme ihtimali olan komutları saklamak için, veri önbelleği ise verileri saklamak için kullanılır. Hem komut hem de veri önbelleğinin boyutları günümüzde genellikle 16-128 KB arasında değişmektedir.

Önbellek denetleyicisi, mikroişlemci tarafından başlatılan bütün bellek hareketlerini izler ve önbellekten veya direk olarak ana bellekten getirilen verinin okunup okunmayacağına, önbelleğe veya ana belleğe yazılıp yazılmayacağına karar verir. Ana belleğe olan hareketler önbelleğe göre oldukça yavaştır, bu yüzden önbellek denetleyicisi direk ana belleğe yapılan hareketlerin sayısını azaltmaya çalışır.

Mikroişlemci tarafından istenen bir verinin önbellek içerisinde bulunmasına isabet (cache hit), istenen verinin önbellek içerisinde bulunamamasına da ıska (cache miss) denilmektedir. Lokalite önbellek denetleyicisinin isabet olması olasılığının artmasını sağlamaktadır. Yüzde yüz isabet oranı olma olasılığı imkansızdır, çünkü denetleyicinin gelecekte kesinlikle kullanılacak veriyi tahmin etmesi çok zordur. Bu da, ıska olmasına neden olur. Lokalite, zamansal (temporal) ve uzaysal (spatial) lokalite olmak üzere ikiye ayrılır. Komutların ve verilerin zamansal ve uzaysal lokalite özellikleri önbellek denetleyicisinin isabet oranını arttırmasına yardım eder (Balch, 2003).

- Zamansal Lokalite (Temporal Locality): Son zamanlarda erişilen veriye yakın gelecekte tekrardan erişilmeye çalışılmasına zamansal lokalite ismi

verilir. Çoğu program için kullanılacak verilerin önbelleğe alınması işlemi yüksek performans alınmasını sağlayan temel özelliklerden biridir. Daha yüksek zamansal lokalite bir kez erişildiğinde kısa bir süre içerisinde daha fazla aynı önbellek bloğuna erişilebilme olasılığını belirtir (Gomathisankaran ve Somani, 2003).

- Uzaysal Lokalite (Spatial Locality): Eğer erişilen bir bellek adresinin yakınındaki adreslere erişilme olasılığı çok yüksekse, buna bellek erişiminin uzaysal lokalite özelliği denir. Önbellek blok boyutu genellikle bu özellik kullanılarak seçilir. Ancak önbellek blok boyutunun çok fazla arttırılması önbelleklerin etkisinin azalmasına neden olur. Çünkü önbellek bloğunun getirilmesi zamanını (fetch time) arttırır. Blok ana bellekten önbelleğe getirilirken veri yolu erişimlerini geciktirir, bu da ekstra zaman kaybına sebep olur. Çoğu uygulama için en etkili blok boyutu 32 byte olarak kullanılır. Bununla birlikte uygulamalarda blok boyutu olarak 32 byte kullanıldığından beri daha fazla uzaysal lokalite özelliğine sahip olmuşlardır (Kumar ve Wilkerson, 1998; Gomathisankaran ve Somani, 2003; Hennessy ve Patterson, 2003).

2.3 Önbelleğe Veri Yazma Yöntemleri

Bilgisayar sistemleri için günümüzde mikroişlemci tasarımları yapılırken, minimum enerji, daha fazla performans ve verimli bellek bant genişliği kullanımı özellikleri aranmaktadır. Özellikle önbelleklerin kullanımında dikkat edilmesi gereken hususlar vardır. İşlemciyle önbellekler ve önbelleklerle ana bellek arasındaki veri yolunun verimli bir şekilde kullanılması performansı arttıran en önemli etkenlerden biridir. Bu nedenle, önbelleklere veri yazılırken kullanılmak amacıyla çeşitli yöntemler geliştirilmiştir: geriye-yazma yöntemi (write back), tampon yöntemi (write through).

- Geriye-yazma Yöntemi: Bu yöntem, mümkün olduğunca L1 önbelleği içerisindeki kirli blokları muhafaza etmeye çalışır. Ancak bu kirli blokların

üzerine başka bir blok yazılmaya çalışıldığında, kirli bloklar L2 önbelleğe yazılırlar. Böylece L2 önbelleğe erişim sayısı mümkün olduğunca az sayıda tutulmuş olur ve veri yolu çok fazla meşgul edilmiş olmaz. Günümüz işlemcilerinden Intel Pentium Mobile işlemcilerin L1 önbelleğinde geriye-yazma yöntemi kullanılmaktadır.

- Tampon Yöntemi: L1 önbelleği içerisindeki herhangi bir blok değiştirildiği anda L2 önbelleğe geri yazılmaktadır. Bu yöntem L1 önbelleğini sürekli temiz durumda (clean mode) tutmaktadır. Ama önemli ölçüde L2 önbelleğe erişim sayısını arttırmaktadır. Geriye-yazma yöntemine göre L2 önbelleğe çok daha fazla erişim yapılır. Intel Itanium 2 işlemcilerdeki L1 önbellekte tampon yöntemi kullanılmaktadır (Stacpoole ve Jamil, 2000; Li ve diğ., 2004).

2.4 Önbellek Organizasyonları

Önbelleğin fonksiyonel olabilmesi için onun kullanışlı veriler depolaması gerekmektedir. Eğer işlemcinin istediği bir veri önbellek içerisinde bulunmuyorsa, şuan bulunan veriler işlemci için faydasız veriler konumundadır. Veri veya komutlara erişildiğinde, işlemci ilk olarak ana bellekte bulunan bir adres üretir. Eğer veri önbelleğe kopyalanmışsa, önbellek içerisindeki verinin adresi ana bellekte bulunduğu adresle aynı değildir. Peki, işlemci ana bellekten önbelleğe kopyalanan bu verinin önbellekteki adresinin neresi olduğunu nasıl anlayacak? İşlemci bir organizasyon yaklaşımı kullanarak, verinin ana bellekte bulunduğu adresi önbellek içerisinde bir adrese dönüştürür.

Bu adres dönüşümünü, ana bellek adresi içerisindeki bitlere özel anlamlar vererek yapar. İlk olarak bitleri belirgin gruplara böleriz, bu gruplara alanlar (fields) denilmektedir. Organizasyon yaklaşımına göre, iki veya üç alana sahip oluruz. Bu yaklaşımlar, verinin bir kopyasının ne zaman önbelleğe kopyalandığını, nereye yerleştirildiğini belirler.

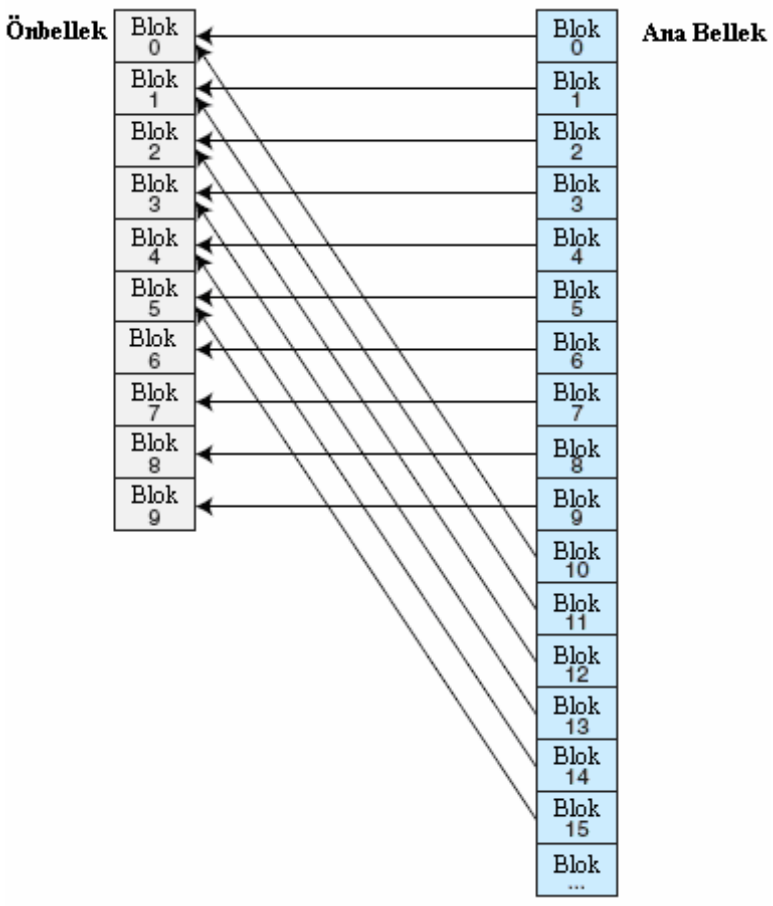
Ana bellek ve önbellek aynı büyüklükte bloklardan oluşmaktadır. İşlemci tarafından bir ana bellek adresi üretildiğinde, ilk önce işlemcinin istediği sözcüğün (word) önbelleğin içerisinde olup olmadığına bakılır. İstenilen sözcük önbellekte bulunamadığında, istenilen sözcüğün bulunduğu bütün ana bellek bloğu önbelleğe yüklenir. Böyle bir mantıkla çalışmak daha önce verilen lokalite ilkesinden dolayı başarılı olmaktadır. Çünkü istenilen sözcüğün yakınındaki sözcüklerin daha sonra istenilme olasılığı yüksektir.

Ana bellek adresi içerisindeki ilk alan bize, eğer veri önbellekte ise bulunduğu konumu, önbellekte değilse nerede bulunduğunu gösterir. Her önbellek bloğunda geçerli bit (valid bit) alanı bulunmaktadır. Geçerli bit'in değeri sıfır veya bloğun etiket kısmı adresin etiketi ile uyuşmuyorsa önbellek bloğu geçerli veri barındırmamaktadır. Bu durumda ıska oluşmuş demektir ve ana belleğe erişmek zorunda kalırız. Eğer geçerli bit'in değeri bir ve etiketler uyuşuyorsa önbellek bloğu geçerli veri barındırmaktadır. Bu durumda isabet olmuş olur. Ana bellek adresinin ilk alanın gösterdiği önbellek bloğunun geçerli bir blok olup olmadığını kontrol ederiz. Sonra adresimizin etiket alanı (tag field) ile önbellek bloğunun etiket alanını karşılaştırırız. Eğer etiket alanları aynı ise, o zaman istenilen önbellek bloğunu bulmuş oluruz. Aynı işlemi ana bellek adresinin farklı bir kısmını kullanarak da yapabiliriz. Bu alana sözcük alanı (word field) denilmektedir. Bütün önbellek organizasyon yaklaşımları için sözcük alanı gerekmektedir. Kullanılan üç çeşit önbellek organizasyon yaklaşımı vardır:

2.4.1 Doğrudan Eşlenmiş Önbellek (Direct Mapped Cache)

Doğrudan eşlenmiş önbellek yaklaşımı önbelleği modüler bir yaklaşım kullanarak yapılandırır. Çünkü önbellek blok sayısından daha fazla ana bellek bloğu bulunmaktadır. Böyle olunca ana bellek blokları, önbellekte bulunan boş alanlar için rekabet etmektedir. Doğrudan eşlenmiş önbellek yaklaşımını kullanarak ana bellekte bulunan X bloğunu önbellekte bulunan Y bloğuna yerleştirelim. Önbellek içerisinde toplam N tane blok olduğunu farz edelim. Örneğin önbellekte 10 tane blok bulunuyorsa; ana belleğin 0. bloğu önbelleğin 0. bloğuna, ana belleğin 1. bloğu

önbelleğin 1. bloğuna, …, ana belleğin 9. bloğu önbelleğin 9. bloğuna, ana belleğin 10. bloğu önbelleğin 0. bloğuna yerleştirilir. Bu işlem aşağıdaki Şekil 2.3'te gösterilmektedir. Şekil 2.3'de de görüldüğü gibi ana belleğin hem 0. bloğu hem de 10. bloğu (20., 30., …) önbelleğin 0. bloğuna yerleştirilmektedir.



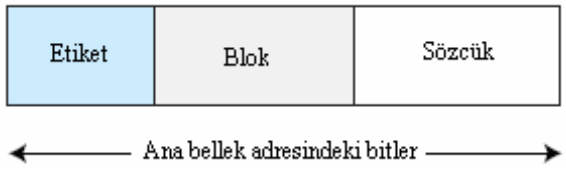
Şekil 2. 3 Ana bellek bloklarının doğrudan eşlenmiş önbellek yaklaşımı kullanılarak önbellek bloklarına yerleştirilmesi (Null ve Lobur, 2003)

Şekil 2.3'te görüldüğü gibi ana bellek bloğunun hem 0. hem de 10. bloğu önbelleğin 0. bloğuna yerleştiriliyor. Her blok önbelleğe kopyalanır ve önceden atanmış etiket alanı kullanılarak tanımlanır. Aşağıdaki Şekil 2.4'te görüldüğü gibi önbelleğe yakından bakacak olursak; önbelleğe birden fazla bloğun yerleştirildiğini görürüz.

Blok	Etiket	Veri	Geçerli Bit
0	00000000	Sözcük A, B, C,...	1
1	11110101	Sözcük L, M, N,...	1
2	--------		0
3	--------		0

Şekil 2. 4 Önbelleğin iç yapısı (Null ve Lobur, 2003)

Şekil 2.4'te görüldüğü gibi iki tane geçerli önbellek bloğumuz bulunuyor. 0. blok ana bellekten getirilen etiketi "00000000" olan birden fazla sözcük içermektedir. 1. blokta da etiketi "11110101" olan sözcükler bulunmaktadır. Kalan iki önbellek bloğu ise geçersiz önbellek bloğudur. Doğrudan eşlenmiş önbellek yaklaşımını gerçekleştirmek için, ikili ana bellek adresini Şekil 2.5'de görüldüğü gibi böleriz.



Şekil 2. 5 Doğrudan eşlenmiş önbellek yaklaşımı kullanılarak ana bellek adresinin biçimlendirilmesi (Null ve Lobur, 2003)

Her alanın boyutu ana bellek ve önbellekteki fiziksel özelliklere bağlıdır. Sözcük alanı (bazen sapma alanı (offset field) olarak da adlandırılır) bulunulan blokta bir sözcüğü anahtar olarak tanımlar ve bu yüzden uygun bit sayısına sahip olmalıdır. Başka bir deyişle bir önbellek bloğunun hangi kısmına verinin yazılacağını sözcük bitleri belirler. Blok alanı, ana bellekte bulunan blokların önbellekte hangi bloğa yazıldığını tutmak için kullanılır. Kalan bitlerde etiket alanını gösterir. Bütün bu alanların toplam bit sayısı ana bellek adresinin bit sayısına eşit olur.

Örnek olarak bir 32 bitlik işlemcide 5 bit sözcük alanını, 10 bit blok alanını, 17 bit de etiket alanını gösterebilir.

2.4.2 Tam Çağrışımlı Önbellek (Fully Associative Cache)

Doğrudan eşlenmiş önbellek yaklaşımını kullanan önbellekler diğer organizasyon yaklaşımlarını kullanan önbellekler kadar pahalı değildir. Çünkü herhangi bir arama işlemine gereksinim duymaz. Her ana bellek bloğu önbellek içerisinde konumlanabileceği bir adrese sahiptir. Bir ana bellek adresi önbellek adresine dönüştürüldüğünde, işlemci blok alanı içerisindeki bitleri basitçe inceleyerek ana bellek bloğunun önbelleğin neresinde olduğunu bilir.

Her ana bellek bloğu için benzersiz bir konum belirlemek yerine, bir ana bellek bloğunun önbellek içerisinde herhangi bir yere yerleşmesine müsaade edebiliriz. Tabii bu bloğu bulmak için tek yol bütün önbelleği aramaktır. Bu işlem için çağrışımlı bellekten (associative memory) yapılan bütün önbelleği paralel olarak arayabiliriz. Yapılması gereken istenilen adresin etiket alanını önbellekteki bütün blokların etiket alanlarıyla karşılaştırıp, istenilen bloğun önbelleğin neresinde olduğunu buluruz. İlişkili bellek, paralel arama yapmak için fiziksel olarak özel donanıma ihtiyaç duyar. Tabii bu da pahalı olmasına neden olur.

Tam çağrışımlı önbellek yaklaşımında ana bellek adresi iki parçaya bölünür: Etiket ve sözcük alanı (Mano, 1993).

2.4.3 Küme Çağrışımlı Önbellek (Set Associative Cache)

Hızı ve karmaşıklığından dolayı ilişkili bellekler çok pahalıdır. Her ne kadar doğrudan eşlenmiş önbellek yaklaşımını kullanan önbellekler pahalı olmasa da çok kısıtlayıcı çalışmaktadır. Küme çağrışımlı önbellek yaklaşımı, doğrudan eşlenmiş ile tam çağrışımlı önbellek yaklaşımlarının birleşiminden oluşur. Doğrudan eşlenmiş önbellek yaklaşımında olduğu gibi bu yöntemde de ana bellekten getirilen blok belirli bir önbellek bloğuna yerleştirilir. Ama en önemli fark, küme çağrışımlı önbellek yaklaşımında tek bir blok yerine birden fazla bloğu birkaç önbellek bloğu satırına (set) yerleştirebilmektedir. Önbellek içindeki bütün satırlar aynı boyutlu olmak zorundadır. Bu boyut çeşitli önbelleklerde farklı boyutlarda olabilir. Örneğin,

2-yollu küme çağrışımlı önbellekte Şekil 2.6'da görüldüğü gibi her satır için iki tane önbellek bloğu bulunmaktadır.

Küme	Etiket	Kümenin 0. Bloğu	Geçerli Bit	Etiket	Kümenin 1. Bloğu	Geçerli Bit
0	00000000	Sözcük A, B, C,...	1	--------		0
1	11110101	Sözcük L, M, N,...	1	--------		0
2	--------		0	10111011	P, Q, R, . . .	1
3	--------		0	11111100	T, U, V, . . .	1

Şekil 2. 6 2-yollu küme çağrışımlı önbellek (Null ve Lobur, 2003)

Yukarıdaki şekilde de görüldüğü gibi 2-yollu küme çağrışımlı önbelleğin 0. ve 1. satırında ilk bloklar geçerli bloklardır ve "A,B,C,L,M,N" sözcükleri bulunmaktadır. Ancak ikinci blokları geçersiz bloklardır. Bu yaklaşımı çağrışımlı bellekte birden fazla (N-yollu) doğrudan eşlenmiş önbellek yaklaşımının kullanılması olarak düşünebiliriz.

Küme çağrışımlı önbellek yaklaşımında ana bellek adresi üç parçaya bölünür: etiket alanı, satır alanı (set field) ve sözcük alanı. Etiket ve sözcük alanlarının ne amaçla kullanıldığını yukarıda bahsetmiştik. Ek olarak satır alanı ise ana bellek bloğunun önbellek içerisinde hangi satırda olduğunu gösterir (Murdocca ve Heuring, 1999; Haraszti, 2002; Null ve Lobur, 2003).

2.5 Yer Değiştirme Algoritmaları

Yeni bir blok önbelleğe yerleştirilmeye çalışıldığında, uygun bir boş blok tanımlanmalıdır. Eğer kullanılmayan bloklar varsa, geçerli bit değeri sıfır olan bloklardan birine ana bellekten getirilen blok yerleştirilir. Ama önbellekteki bütün blokların geçerli bit değeri bir ise önbellekten bir blok çıkarılıp yerine ana bellekten getirilen blok yerleştirilir ve o zaman ıska olayı meydana gelir. Önbellek dolu olduğunda çıkarılacak olan bloğun tespiti için yer değiştirme algoritmaları kullanılır. En çok kullanılan yer değiştirme algoritmaları: Rasgele (random), FIFO (First in first out – İlk giren ilk çıkar), LRU (Least recently used – son zamanlarda en az kullanılan). Yer değiştirme algoritmaları performans açısından çok önemlidir.

- Rasgele algoritması: Basitçe rasgele bir blok seçilir ve çıkarılır.

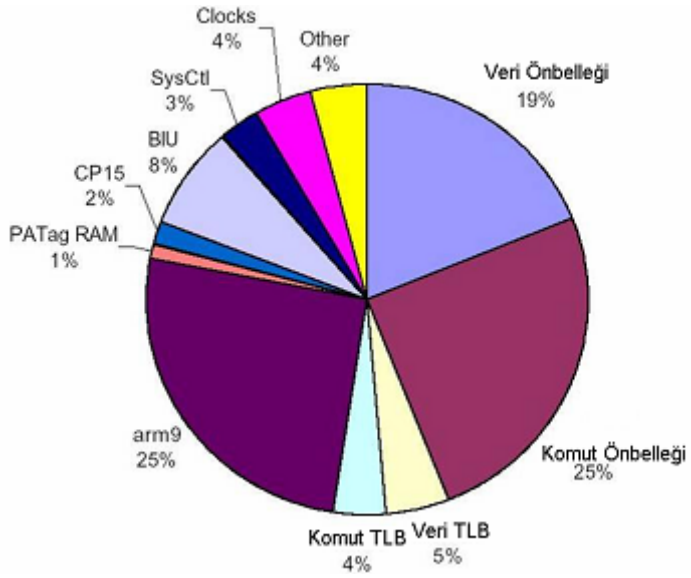
- FIFO algoritması: Gerçekleşme sıralarına göre bütün bellek erişimleri N boyutlu bir dizide tutulurlar. Önbellek dolduğu zaman bu diziye ilk giren blok çıkarılacak blok olarak seçilir. FIFO algoritmasında önbellekte bulunan her bir sözcük için ekstra bitler kullanılır.

- LRU algoritması: Her bir blok için, bu bloğa erişildiğinde güncellenen bir zaman mührü (time stamp) eklenir. Önbellekten bir blok çıkarılması gerektiği durumlarda her bloğun zaman mührüne bakılarak işlemci tarafından son zamanlarda en az kullanılan blok çıkarılır. Bu algoritmanın önbelleklerde uygulanabilmesi için özel donanım ve ekstra bitler kullanılır (Mano, 1993; Murdocca ve Heuring, 1999; Stacpoole ve Jamil, 2000).

BÖLÜM 3

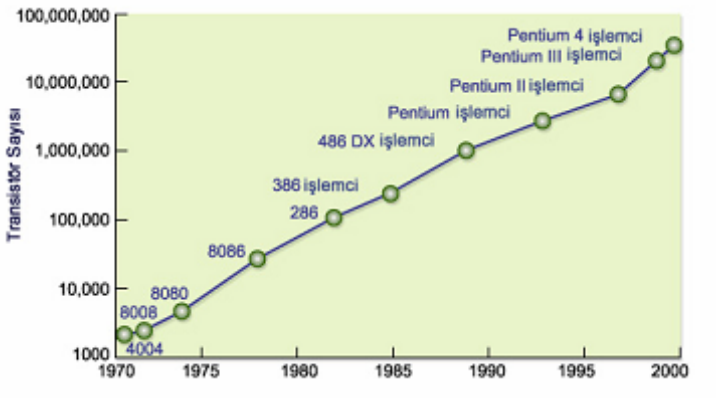
ENERJİ OPTİMİZASYONU

Bilgisayar sistemlerinde güç tüketiminin optimizasyonu günümüzde önemli araştırma alanlarından biri olmuştur. Bunun nedenlerini, taşınabilir bilgisayarlarda pil ömrünün uzatılması, diğer gömülü sistemlerde ise veri güvenilirliğinin sağlanması, işlemcinin soğutulma mekanizmasının (cooling) geliştirilmesi, işlemci içerisinde kullanılan malzeme (ambalajlama) kalitesinin arttırılması ve harcanan gücün azaltılması gibi konular oluşturmaktadır. Taşınabilir bilgisayarların pillerinin sınırlı enerjiyle yüklü olmasından dolayı, bu enerjinin en verimli biçimde kullanılması ve kullanım süresinin maksimum düzeye çıkarılması gerekmektedir. Taşınabilir sistemlerin pil kullanımlarında yapılan güç optimizasyonları, genellikle performansı olumsuz yönde etkilemektedir. Günümüzde bilgisayar sistemlerinde yer alan işlemcilerdeki güç tüketiminin yaklaşık %50 - %60'nı önbellekler harcamaktadır (Montenaro ve diğ., 1996; Gordon-Ross ve diğ., 2004). Dolayısıyla önbellekler üzerinde gerçekleştirilen güç optimizasyonu, taşınabilir sistemlerin pillerinin güç sarfiyatı üzerine doğrudan etkimektedir.



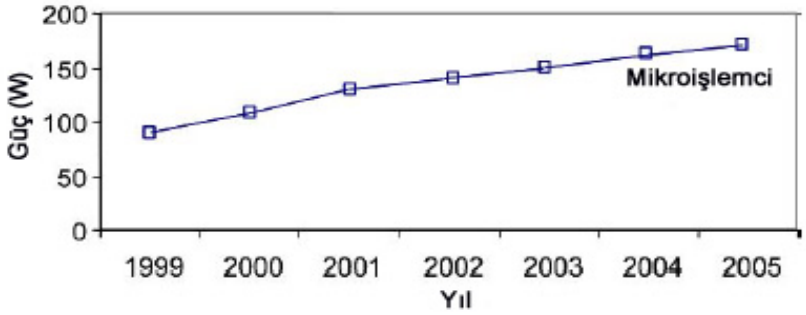
Şekil 3. 1 Önbelleklerin güç tüketimi (Gordon-Ross ve diğ., 2004)

Günümüzde Moore kanunlarına dayanarak işlemcilerdeki transistör sayısı her geçen yıl katlanarak artmaktadır. Teknolojik ilerlemeler sonucu transistörlerin kanal genişliği daha da küçülmektedir. Bu teknolojik gelişmeler sonucu devamlı olarak işlemci içerisindeki transistör sayısı ve performansı artmaktadır. Örneğin Intel şirketinin verilerine göre, 1993 yılında üretilen Pentium Pro işlemcilerde 3.1 milyon transistör, 1997 yılında üretilen Pentium II işlemcisinde 7.5 milyon transistör, 1999 yılında üretilen Pentium III işlemcisinde 9.5 milyon transistör, 2000 yılında üretilen Pentium IV işlemcilerde de 42 milyon transistör bulunmaktadır. Şekil 3.2'de görüldüğü gibi, örnek verilen bu işlemcilerdeki transistör sayıları, işlemcilerin saat hızları arttıkça farklılık göstermektedir(IMTCC, 2006).



Şekil 3. 2 Moore Kanununa göre transistör sayısı artış grafiği (IMTCC, 2006)

Aynı mimariye sahip işlemcilerde bile transistör sayısının fark etmesinin nedeni, işlemcilerde kullanılan önbelleklerden dolayı farklılaşmaktadır. İşlemciler içerisindeki önbellekler toplam transistör sayısının önemli bir kısmını oluşturmaktadır. İşlemcilerdeki transistör sayısının artmasıyla ortaya çıkan en önemli problemlerden biri ısınmadır. Eğer işlemciler uygun soğutma sistemleriyle soğutulmazlarsa ortaya çıkan yüksek ısıdan dolayı oluşacak yumuşak hatalar nedeniyle verilerde bozulmalar meydana gelmekte ve en kötü ihtimalle işlemcinin yanmasına neden olabilmektedir. Ayrıca transistör sayısının her geçen gün artmasıyla harcanan güç tüketimi de giderek artmaktadır (Şekil 3.3). Bundan dolayı, önbellekler, işlemcinin toplam güç tüketimi, sistem performansının belirlenmesinde önemli rol oynamaktadır.



Şekil 3. 3 Yıllara göre işlemcilerdeki güç tüketim artışı (Jeong, 2004)

Şuana kadar önbellekler üzerinde hem güç tüketiminin azaltılması hem de performansın arttırılması konusunda birçok çalışma yapılmıştır. Ancak yapılan bu çalışmalarda genellikle performans-enerji tüketim ilişkileri araştırılmamıştır. Bu çalışmada performansa dayalı önceden blok getirme işleminin enerji tüketimine olan etkileri araştırılmıştır.

Güç tüketimi iki grupta kategorize edilmektedir: Dinamik güç tüketimi (Dynamic Power) ve sızıntı (Leakage Power) veya statik güç tüketimi (Static Power).

3.1 Dinamik Güç

Dinamik güç tüketimi, düşünülen donanıma erişildiğinde devre anahtarında bit geçişleri (1'den 0'a veya 0'dan 1'e) olduğunda karşılaşılan güç tüketimidir. Anahtarlar sığanın elektrik yüküyle dolmasını veya boşalmasını sağlar, bu sayede akımların devrenin içerisine dolmasını ve ısı dağılımıyla gücün tüketilmesini sağlar. Devrelerin dinamik güç tüketimi aşağıdaki formülle hesaplanır (Sakurai ve Newton, 1990).

$$P_D = \frac{1}{2} \alpha C V_{dd}^2 \qquad (3.1)$$

Bu formülde V_{dd} besleme gerilimini, C erişilen kapının etkin sığasını göstermektedir. Etkinlik faktörü α, her birim zamanda sinyal geçişlerinin ortalamasını gösterir. Dinamik güç tüketiminin azaltılması formülün sağ tarafında bulunan her bir terimin değerinin azaltılmasıyla sağlanır.

Dinamik güç besleme geriliminin karesiyle orantılı olduğundan, besleme geriliminin değerini azaltmak dinamik güç tüketiminin azaltılması için en etkili yaklaşım olacaktır. Yani aygıta olan erişim sayısını azaltarak veya erişim başına güç maliyetini düşürerek dinamik güç tüketimi kabul edilebilir bir seviyede tutulabilir. Güç tüketimi ve güç yoğunluğunu yönetilebilir bir seviyede tutarak besleme gerilimini düşürebiliriz.

Yukarıda verilen formül 3.1'de besleme geriliminin ne kadar azaltılabileceği çok açık değildir. Formül 3.1'deki besleme gerilimi ve metal-oksit yarı iletken kapının (MOS – Metal Oxide Semiconductor) hızı arasındaki ilişki formül 3.2'de gösterilmiştir.

$$D \approx \frac{V_{dd}}{(V_{dd} - V_{th})^{\alpha}} \qquad (3.2)$$

Burada, D tek bir yarı iletken transistörün iletim zamanını göstermektedir. V_{th} transistörün eşik gerilim değeridir. α ise sabit bir değer olup, değeri 1.3'dür (Sakurai ve Newton, 1990). Transistörün iletim zamanı besleme gerilimiyle ters orantılıdır. Eğer V_{th} değişkeninin değeri değişmezse, besleme gerilimi transistörün performansının bozulmasına neden olur. V_{dd} değişkenine göre yeterli yükseklikte gerilim $(V_{dd}-V_{th})$ sağlamak için V_{th} değişkeninin değerini değiştirmek uygun olacaktır. Ancak, V_{th} değişkeninin değerini arttırmak sızıntı güç tüketiminin üssel olarak artmasına neden olur (Hung, 2004).

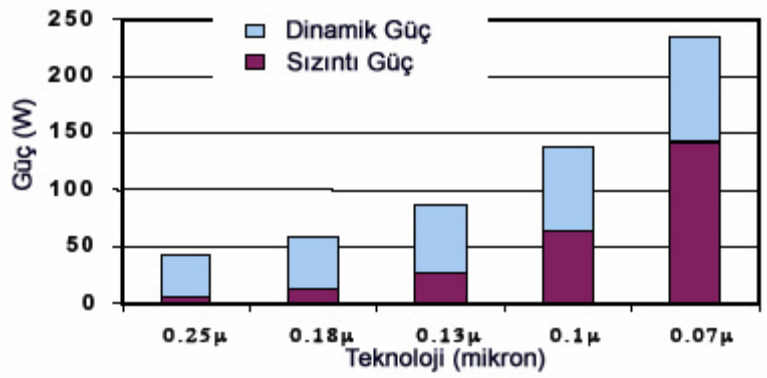
Bugüne kadar bilgisayardaki dinamik güç tüketimi üzerine birçok çalışma yapılmıştır. Bunlardan, önbelleklerin güç tüketimi için (Ghose ve Kamble, 1999; Inoue ve diğ., 1999), anabellekler için (Catthoor ve diğ., 1998), işlemci kontrol ünitesi için (Zyuban ve Kogge, 1998) dallanma yönünün önceden tahmininin güç tüketimi ile ilişkisini (Parikh ve diğ., 2002), komut-veri TLB'lerde (Translation Lookaside Buffer) dinamik güç harcanmasının azaltılması ile ilgili çalışmalar (Kadayıf ve diğ., 2002; Kadayıf ve diğ., 2005).

Yapılan bu çalışmalar sonucunda dinamik güç tüketiminin azaltılması için aşağıdaki üç yol kullanılmaktadır:

- Bir görevi gerçekleştirmek için gerekli olan zamanı azaltmak,
- Gerilimi azaltmak,
- Her bir makine çevriminde daha az sayıda transistör arasında geçiş yapacak şekilde mikro-mimari geliştirmek (Brorsson , 2001).

3.2 Sızıntı Güç

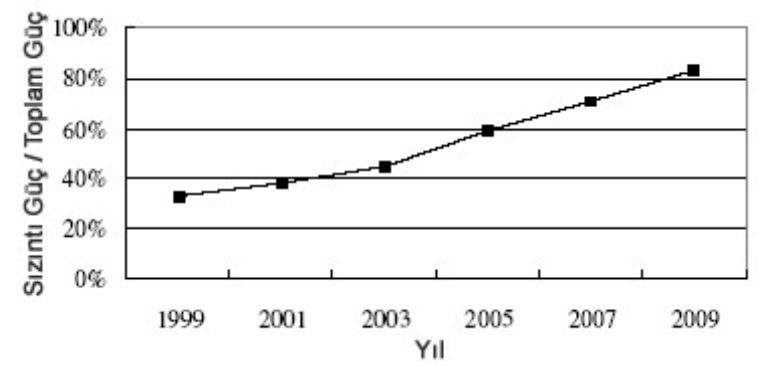
Statik güç, devrelerdeki transistörler içerisinde var olan sızıntı akımlar nedeniyle oluşan bir güç tüketimidir. Günümüze kadar olan süre içerisinde çoğu zaman yapılan çalışmalarda toplam güç tüketiminin azaltılması için dinamik güç tüketiminin düşürülmesine odaklanılmıştır. Ancak, teknoloji ilerledikçe transistör boyutları çok daha küçülmeye başladığından, transistörler daha fazla sızıntı güç ortaya çıkarmaya başladılar. Sızıntı akımının büyüklüğü her yeni teknolojik kuşak için hızla artmaktadır. Toplam sızıntı güç tüketiminin de her işlemci kuşağı için beş kat artacağı tahmin edilmektedir (Şekil 3.4)(Borka, 1999).



Şekil 3. 4 Teknolojiye göre dinamik ve sızıntı güç oranları (Krishnamurthy ve diğ., 2002)

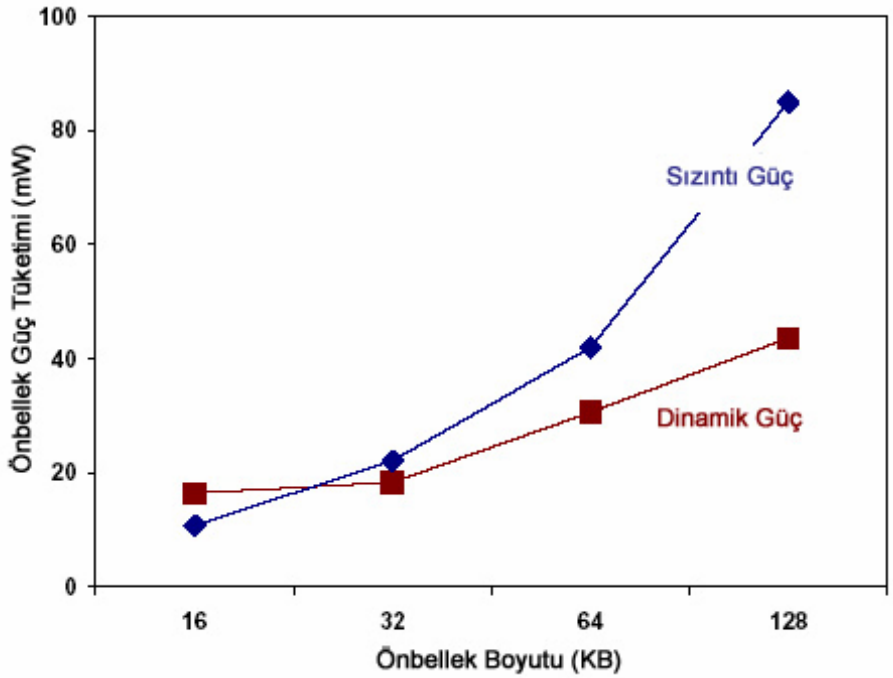
Çoğu gelişmiş işlemcilerde kullanılan tasarımların birçoğunda statik güç tüketiminin oranı hemen hemen dinamik güç tüketimiyle aynıdır. Aygıtlardaki besleme geriliminin azaltılmasıyla transistörlerdeki akım sızdırmaları daha kolaylaşmaktadır. Bu da statik güç tüketimine neden olmaktadır. Bu yüzden dinamik

güç tüketiminin azaltılmasına ve performans arttırılmasına yönelik yapılan çalışmalar statik güç tüketiminin toplam güç tüketimindeki oranını arttırmaktadır. Örneğin ITRS (International Technology Roadmap for Semiconductors) verilerine göre, 1999 yılında statik güç tüketimi tüm gücün %35'i iken, bu oran 2003'te %42'ye, 2005'te ise yaklaşık %60'a çıkmıştır. 2009 yılında statik gücün payının %80'in üzerinde olacağı tahmin edilmektedir (ITRS, 2005) (Şekil 3.5). Sonuç olarak, statik güç tüketiminin azaltılmasına yönelik yapılan çalışmaların önemi giderek artmaktadır. Statik güç tüketiminin gelecekte VLSI (Very large-scale integration) tasarım faktörüyle sınırlandırılabileceği tahmin edilmektedir (Hung, 2004).



Şekil 3. 5 Yıllara göre sızıntı gücün toplam güce oranı (ITRS, 2005)

Modern bilgisayarlarda mikroişlemcilerdeki transistörlerin çok büyük bir kısmı önbellekler için kullanıldığından, önbelleklerde harcanan statik güç işlemcinin toplam güç tüketiminin önemli bir kısmını oluşturmaktadır. Daha önceden de bahsettiğimiz gibi modern bilgisayarların çoğunda L1 ve L2 önbellekler kullanılmaktadır. L1 ve L2 önbelleklerin boyutları içlerinde barındırdıkları transistör sayısına göre değişmektedir. Transistör sayıları değiştiği içinde L1 ve L2 önbelleklerdeki statik güç tüketiminin farklı boyutlarda olması beklenmektedir. Yapılan çalışmalara göre 0.13µm teknolojisi için L1 önbelleğinin toplam güç tüketiminin %30'nu, L2 önbelleğinin ise %70-%80'ni statik güç tüketimi oluşturmaktadır (Zhang ve diğ., 2002). Şekil 3.6'ya bakacak olursak farklı boyutlardaki önbellekler için farklı miktarlarda statik güç tüketimi olduğu görülmektedir.



Şekil 3. 6 Önbellek boyutuna göre dinamik ve sızıntı güç tüketim miktarları (Kim, 2004)

Önbelleklerde statik güç tüketiminin azaltılması konusunda birçok çalışma yapılmıştır. Bu çalışmalardan biri, önbellek bozulması (cache decay) diye bilinen yöntemdir. Bu yöntemde, önbellek bloğunun ölü periyoda girdiğinde besleme geriliminin kesilmesi mantığına dayanmaktadır. Kullanılmayan önbellek bloklarının besleme gerilimi kesilerek güç tasarrufuna gidilmektedir. Bir veri bloğu ana bellekten önbelleğe getirildikten sonra, onun üzerinde bir dizi işlemler gerçekleştirilir. Daha sonra bu bloğa belirli bir süre erişim olmazsa, blok gereksiz yere statik güç harcayacaktır. Bunu önlemek için, bloğu Aktif Durum (AK) modundan Durum Yokedici (DY) moda sokarak statik güç tüketmesi önlenmiş olur (Kaxiras ve diğ., 2001).

Bloğun DY modunda herhangi bir statik güç tüketmesi söz konusu değildir. Yalnız burada dikkat edilmesi gereken nokta; eğer ölü periyoda giren blok daha önceden değiştirilmişse, ilgili blok DY moduna geçmeden önce kendisinden bir önce seviyedeki bellek birimine yazılması gerekmektedir. Aksi halde DY moduna sokulan

bloktaki veri kaybolacak, bu da bilgisayarın kararsız bir duruma girmesine neden olacaktır.

Diğer bir dikkat edilmesi gereken nokta ise bloğun ölü periyoda ne zaman girdiğinin tahminidir. Eğer blok çok erken bir zamanda DY moduna sokulursa, bu bloktaki veriye kısa bir süre sonra ihtiyaç duyulduğunda tekrardan bir sonraki bellek birimine erişilerek aynı verinin getirilmesi gerekmektedir. Bu da ekstradan makine çevrimi gereksiniminden dolayı performansın düşmesine neden olacaktır. Öte yandan, blok çok geç DY moduna sokulursa da statik güç tasarrufu azalacaktır (Lai ve diğ., 2001).

Statik güç tüketiminin azaltılmasına yönelik yapılan bir diğer çalışma ise sayıklayan önbellek (drowsy cache) diye adlandırılan yöntemdir. Bu yöntemde, yeterince uzun süre erişilmeyen blokların voltaj gerilimleri dinamik olarak (programın çalışması sırasında) düşürülerek statik güç tüketiminde azaltma yoluna gidilir (Flautner ve diğ, 2002). Bir önceki yöntemden temel farkı, erişim yapılmayan blokların DY modu yerine Durum Koruyucu (DK) moduna geçirilmesidir. DK modundaki bloğun statik güç tüketimi AK modundaki güç tüketiminin yaklaşık %10'nu kadardır (Kim ve diğ., 2004b). DY modunda güç tüketimi sıfır olduğu için ilgili bloktaki veri korunmaz. Fakat ilgili blok DK moduna sokulursa içerisindeki veri korunur, yalnız bu bloklara erişim fazladan bir makine çevrimi gerektirir (Li ve diğ., 2003).

BÖLÜM 4

ÖNCEDEN GETİRME İŞLEMİ (PREFETCHING)

4.1 Önceden Getirme İşlemi Nedir?

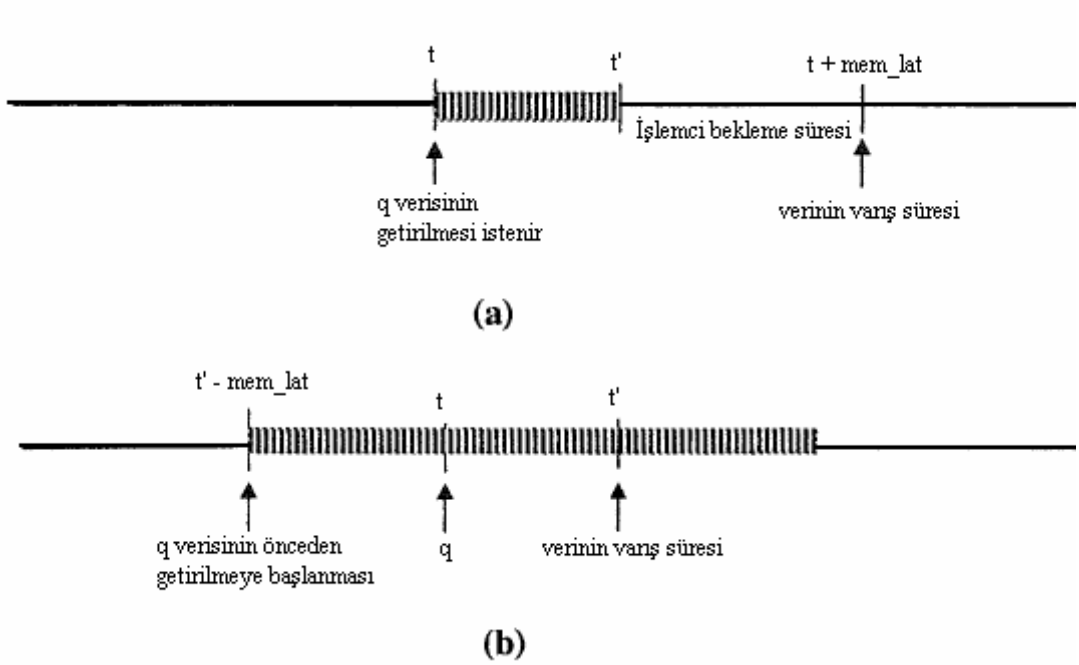
Ana belleğe uzun erişim süreleri modern mikroişlemcilerin performanslarını sınırlamaktadır. Sonuç olarak, önbellekte aranan bir veri bulunamadığında (miss) önemli ve umulmadık veri yolu tıkanmaları ve çok fazla işlemci devir sayısı harcanmaktadır. Bu problem, bellek erişim sürelerindeki gecikmelerin artmasıyla büyümeye devam edecektir. Bu sebepten dolayı, uzun bir süre gecikme süresi maskeleme teknikleri önemli bir tasarım gereksinimi olmaya devam edecektir ve bu probleme karşı verinin önceden getirilmesi (prefetching) yöntemi geliştirilmiştir.

Veriyi önceden alıp-getirme yöntemi, işlemci tarafından veriye gereksinim duyulmadan önce ana bellekten önbelleğe verinin getirilmesi işlemidir. Gecikmeyi azaltma (latency hiding) ana bellek erişimlerinin ve işlemcideki yürütme işlemlerinin paralel şekilde gerçekleştirilmesiyle başarılmıştır.

Önbelleğe verinin daha önceden getirilmesi ıska sayısının ve meydana gelen gecikmelerin süresini azaltmak için kullanılan bir tekniktir. Bu tekniğin başarılı olması gelecekte erişilecek olan ana bellek adreslerinin olabildiğince doğru tahmin edilmesine ve bu erişimlerin zamanında yapılmasına bağlıdır (Ro, 2004).

Aşağıdaki Şekil 4.1'de ana belleğe erişim sürelerindeki gecikmenin önceden getirme yöntemiyle nasıl azaltıldığı gösterilmektedir. Bu şekle göre, q verisinin t zamanında işlemci tarafından istenildiğini ve hem L1, hem de L2 önbellekte bulunmadığını farz edelim. Şekil 4.1(a)'da görüldüğü gibi q verisi ancak (t + mem_lat) sürede işlemci için hazır olmaktadır. Burada mem_lat değişkeni verinin ana bellekten alınması için geçen süreyi göstermektedir. Eğer bu esnada uygun bağımsız komutlar varsa, işlemci sadece bu komutların t' süresine kadar yürütülmesine izin verir. Verinin getirilmesi için geçen süre içerisinde işlemci

(t+mem_lat) süresine kadar beklemeye geçer. Önceden getirme işlemi, q verisini t zamanından önce bellekten isteyerek, işlemcinin bekleme süresinin azaltılmasını sağlar. Ayrıca q verisi Şekil 4.1(b)'de olduğu gibi (t' – mem_lat) sürede getirilmeye başlanmış olsaydı, işlemcinin bekleme süresi tamamıyla ortadan kaldırılmış olurdu.



Şekil 4. 1 Önceden getirme işleminin gösterimi (Yang, 2001)

Önceden getirme işlemi tahmine dayalıdır. Bu işlem gerçekleştirilirken yazmaçlara herhangi bir değer yüklenmez ve sanal adres hataları ve koruma ihlalleri için olağandışı durumlara sebep olmaz. Bu özellik programın çalıştırılmasının herhangi bir aşamasında önceden getirme işleminin yapılmasına izin verir. Böylece önceden getirme işlemi daha fazla esnek olarak gerçekleştirilmiş olur.

Önceden getirme işlemini gerçekleştirmek için aşağıdaki iki konuyu göz önünde bulundurmamız gerekmektedir.

- *Adres Tahmini:* Etkin bir adresteki komut, yüklendiği anda işlenmektedir. Bellek işlemleri gerçekleşmeden önce etkin bir önceden getirme işleminin yapılabilmesi için adresler üreten iyi bir adres tahmin planına ihtiyacımız vardır.

- *Önceden Getirme İşlemi Zamanlaması:* Önceden getirme işleminin başarılı olması için zamanlama çok kritik öneme sahiptir. Eğer önceden getirme işlemi çok erken gerçekleştirilirse, erken getirilen bloklar işlemci tarafından erişilmeden önce önbellekten çıkarılabilirler. Bu tarz işlemlere erken önceden getirme işlemi (early prefetch) adı verilir. Eğer önceden getirme işlemi çok geç zamanda gerçekleştirilirse, ana bellek erişim sürelerinde azaltılmaya gidilememiş olur (Yang, 2001).

4.2 Bellek Duvarı Problemi (Memory Wall Problem)

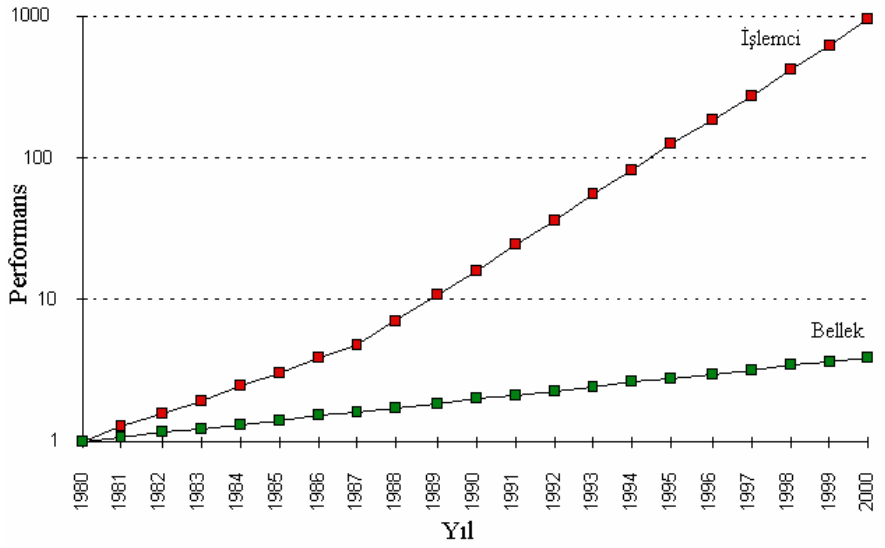
Geçtiğimiz 20-30 yılda bilgisayar sistemlerinin performansında olağanüstü gelişmeler meydana gelmiştir. Devre ve bilgisayar mimarisindeki yenilikler bu gelişmenin hızını arttırmıştır. Devre mimarisindeki gelişmeler, özellikle transistör boyutunun başarılı bir şekilde küçültülmesi, mikroişlemcilerin daha da hızlanmasını ve çip alanında birim başına düşen transistör sayısını artmasına neden olmuştur. Bilgisayar mimarisindeki gelişmeler açısından bakacak olursak, veri yolları kullanılarak komut seviyesinde paralellik başarılı bir şekilde uygulanarak bilgisayarların performansı arttırılmıştır.

Ancak, komut seviyesinde daha yüksek paralellik sağlamak giderek zorlaşmaktadır. Bundaki en önemli sebep ana belleğe erişim süresidir. 1980 yılından itibaren işlemci ve bellek hızları arasındaki fark önemli oranda artmaktadır. 1986 yılından itibaren her yıl işlemcilerin hızları bir önceki yıla oranla %55 artmaktadır (1980 yılından 1986 yılına kadar olan sürede her yıl %35 artış olmuştur), ana bellek hızı ise her yıl sadece %7 oranında artmaktadır (Şekil 4.2). Böylece işlemci ve ana bellek arasındaki hız boşluğu her iki yılda bir iki katına çıkmaktadır. Ortaya çıkan aradaki bu hız farkı Bellek Duvarı problemi (Memory Wall Problem) olarak bilinen bu problemin oluşmasına neden olmuştur (Solihin, 2002).

Modern mikroişlemciler bellek duvarı problemini hafifletmek için büyük boyutlu önbellekler kullanmaktadır. Son 10 yılda, önbellekler bu problem için büyük bellekleri başarılı bir şekilde adresleyebilmişlerdir. Ancak, bu işlemi yaparken verilerin uzaysal veya zamansal lokalitesine güvendiklerinden, herhangi bir

uygulama yeterli lokaliteye sahip değilse başarılı olamayacaklardır. Bu duruma günümüzün popüler uygulamaları olan düzensiz bellek erişim biçimlerine sahip multi-medya işleme, veritabanları veya bilimsel uygulamalarında karşılaşılmaktadır. Buna ek olarak, günümüz uygulamalarında işaretçi zinciri (pointer chasing), karmaşık kontrol-akışı (complex control-flow), tekrarlanan fonksiyon çağrıları (recursive function calls) nedeniyle erişim düzenliliği azalmaktadır. Ayrıca, uygulamaların çoğunda lokalite eksikliği nedeniyle fazla sayıda ıska meydana gelmektedir (Ro, 2004; Solihin, 2002; Cooksey, 2002).

Sık meydana gelen önbellek ıskalarını önlemek için çeşitli önceden getirme yöntemleri geliştirilmiştir. Bu yöntemlerden bazıları donanım tabanlı (hardware prefetching) bazıları ise yazılım tabanlı (software prefetching) önceden getirme teknikleridir. Donanımsal önceden getirme işlemi, dinamik olarak çalışma zamanı bellek erişim biçimlerini adapte eder ve tahmini olarak ihtiyaç olunan önbellek bloğunu getirir (Chen ve Baer, 1995). Buna karşın, yazılımsal önceden getirme işlemi, statik analize dayalı derleme zamanında program kodunun içerisine önceden getirme komutlarını ekler (Luk ve Mowry, 1996).



Şekil 4. 2 İşlemci – bellek performans farkı grafiği. Y ekseni logaritmik olarak artmaktadır (Cooksey, 2002).

4.3 Önceden Getirme İşleminin Türleri

4.3.1 Donanım Tabanlı Önceden Getirme İşlemi (Hardware Prefetching)

Donanım tabanlı önceden getirme işlemi, çalışma zamanında referans adrese göre ek verileri önceden alıp getirerek ıska oluşma oranını azaltmaya çalışır. Donanım desteği kullanılarak önceden getirme işlemi otomatik olarak başlatılır. Önceden getirme işlemi için kullanılan donanım dinamik olarak lokaliteye bakarak önceden getirilecek verilerin adreslerini tahmin eder. Bu teknik düzenli mesafelerde örneğin vektör verilerde önceden getirilecek veri için kullanışlı bir yöntemdir.

Derleyici teknolojisine bağlı değildir, fakat geliştirilmesi pahalı bir tekniktir. Ayrıca, potansiyel olarak daha az esnek ve uygun programın yapısal bilgisini derleyici için kullanmadığından daha az verimlidir. Donanıma dayalı önceden alıp getirme işleminin yöntemlerinin çoğunda geçmişteki bellek erişim davranışlarından yola çıkılarak gelecekteki bellek erişim biçimlerinin tahmin edilebileceğine inanılmaktadır (Crago, 1997; McIntosh, 1998; Lim, 1999).

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

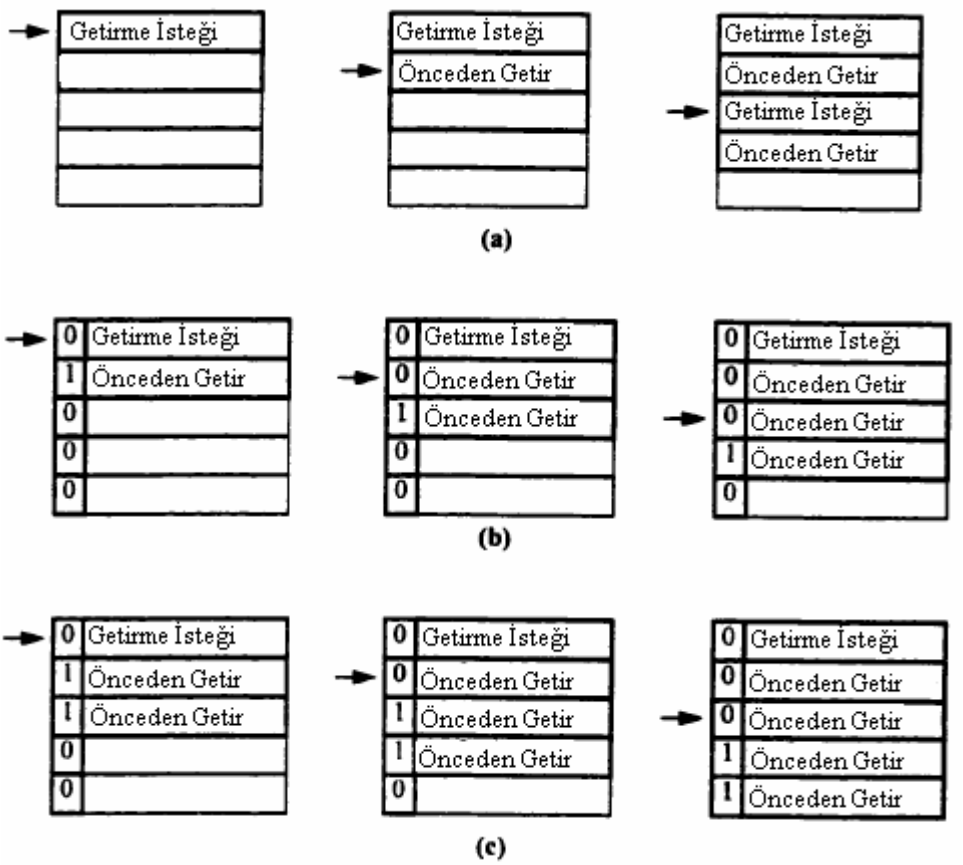
Önceden getirme işleminin en basit biçimi, önbelleğe verinin getirilmesi esnasında bir sonraki önbellek bloğunun da getirilmesi işlemidir. Sıralı önceden getirme (Sequential Prefetching) veya sonraki bir bloğu önceden getirme (one block lookahead - OBL) olarak da adlandırılmaktadır. Önceden getirme işleminin bu biçimi programların çok fazla uzaysal lokaliteye sahip olduğu durumlarda doğru çalışır. Sıralı önceden getirme işlemi büyük önbellek bloklarıyla ilişkili problemlerin çoğuyla karşılaşmaksızın uzaysal lokalitenin avantajlarını kullanabilir.

Bir sonraki bloğu önceden getirme yöntemi, herhangi bir b bloğuna erişildiğinde b+1 bloğunun önceden getirilmesi işlemini başlatırken, bu erişim tipine göre farklılıklar göstermektedir. Smith (1992)'e göre, ıska olduğunda önceden getirme (prefetch-on-miss) ve etiketlemeli önceden getirme (tagged prefetch) şeklinde iki çeşit yaklaşım kullanılmaktadır. Iska olduğunda önceden getirme

algoritması basitçe b bloğuna erişim olmaya çalışıldığında ıska olursa b+1 bloğunun önceden getirilmesi prensibine dayanmaktadır (Şekil 4.3(a)). Ancak b+1 bloğu daha önceden önbelleğe yerleştirilmişse önceden getirme işleminin başlatılmasına gerek yoktur.

Etiketlemeli önceden getirme algoritmasında ise her önbellek bloğuna, bir sonraki bloğun ne zaman önceden alınıp-getirileceğini gösteren bir işaret biti eklenmiştir (Şekil 4.3(b)). Bir blok önceden getirildiğinde, onun işaret biti sıfır yapılır. Ana bellekten önbelleğe blok getirme işlemi esnasında bloğa erişilirse ve biti sıfırsa, bir sonraki sıralı bloğun önceden alınıp-getirilme işlemi tetiklenmiş olur ve esas bloğun işaret bit değeri bir yapılır. Etiketlemeli önceden getirme işleminde her blok için fazladan bir bitlik alana ihtiyaç duyulmaktadır. Bu bit, o bloğun daha önceden erişilip erişilmediğine dair bilgi sahibi olmamızı sağlar. Sadece b bloğuna ilk seferde erişildiğinde b+1 bloğunun önceden getirilmesi mümkündür. Yapılan bu tez çalışmasında da etiketlemeli önceden getirme yöntemi kullanılacaktır.

Smith (1992)'e göre, etiketlemeli önceden getirme yönteminin hem komut hem de veri önbelleğindeki ıska oranını %50 ila %90 arasında azalttığını bulmuştur. Iska olduğunda önceden getirme yöntemi ise etiketlemeli önceden getirme yöntemine göre yarı yarıya daha az verimli sonuçlar vermiştir. (Gornish, 1995; Sair, 2003; VanderWiel, 1998).



Şekil 4. 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 (VanderWiel, 1998).

4.3.1.2 Markov Yöntemi

Verileri önceden getirmek için kullanılan bir diğer gelişmiş teknik ise Markov yöntemidir. Bu yöntemde, Markov tablosu adı verilen, gelecekte ana bellekte erişilmesi muhtemel adresleri veya tekrarlayan adres dizilerini tutmak amacıyla bir tablo kullanılmaktadır. Markov yönteminde mutlaka bellek erişim biçimlerinde mesafeye dayalı davranış gerekmemektedir, ama load (gösterilen adresteki veriyi okuma komutu) adreslerindeki ilişkiyi kullanmaktadır. Bu yöntem tahmine dayalı bir tekniktir. Bu tahmine dayalı donanımsal önceden getirme tekniği, düzenli bellek erişim biçimi gösteren bilimsel çalışmalar için etkileyici ve verimli bir yöntemdir. Ancak, tekrarlanmayan biçimde adres erişimlerine sahip bilimsel olmayan uygulamalarda tahmine dayalı önceden getirme teknikleri gelecekte ulaşılması

gereken adresleri tahmin edemez. Bu yüzden bu tarz uygulamalar için kullanışsızdır (Kim, 2004).

4.3.1.3 Stream Buffer Yöntemi

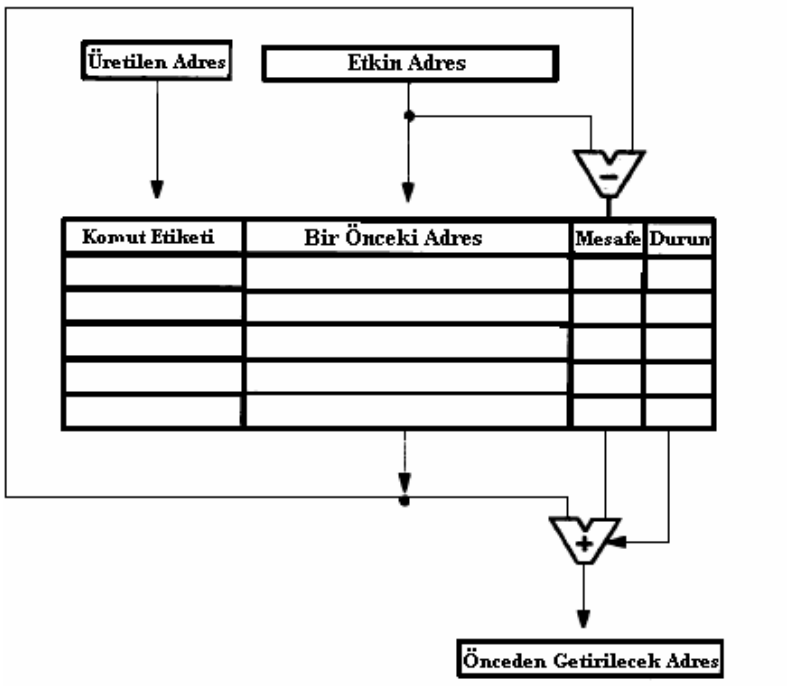
Stream buffer yöntemi, Jouppi (1990) tarafından önerilen bir yöntemdir. K tane ardı ardına sıralı bloğun önbelleğe getirilmeden önce stream buffer'a önceden getirilmesi işlemine dayanan bir yaklaşımdır. Stream buffer donanımı FIFO algoritmasına göre kullanılan bir tampon bellektir. Bu yaklaşıma göre ıska olduğunda, bulunamayan blok ana bellekten önbelleğe ve sonraki K tane sıralı blokta stream buffer içerisine getirilmektedir. Stream buffer kuyruğunun başındaki blok referans olarak gösterildiğinde, bu blok önbelleğe gönderilir ve kalan bloklar kuyruğun başına taşınırlar. Aynı zamanda getirilmesi istenilen bloğun ardındaki blokta stream buffer kuyruğunun sonuna eklenir. Bu yöntemde erken getirilen bloklar direk olarak önbelleğe yerleştirilmediği için önbellekte kirlilik olması engellenmiş olur. Ancak, önbellekte ıska olursa ve istenilen blok stream buffer kuyruğunun başında bulunamazsa, kuyruk boşaltılır. Bu nedenle, performans kazancı sağlamak istiyorsak, stream buffer'a önceden getirilmiş olan blokları getiriliş sırasına göre erişmeliyiz (VanderWiel, 1998).

4.3.1.4 Mesafeye Dayalı Önceden Getirme (Stride-based Prefetching)

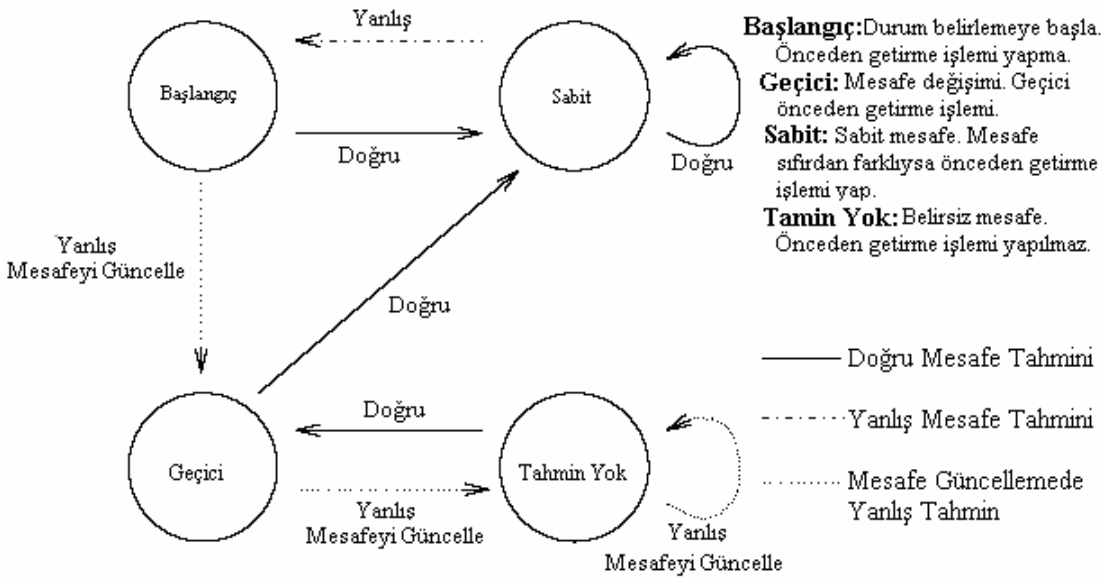
Bu yöntemde, load komutunun eriştiği son adres ve buna ek olarak bir önce erişilmiş olan adresle arasındaki fark kayıt altında tutulur. Son erişilen adres ile bir önceki adres arasındaki farka mesafe (stride) adı verilmektedir. Hesaplanan bu mesafe değeri sayesinde, işlemci tarafından üretilen adres üzerine mesafe değeri eklenerek load komutunun erişmesi muhtemel bir sonraki adres hesaplanmış olur (Sair, 2003).

Yalnız bu yaklaşımda bütün bellek erişim adreslerinin kayıt altında tutulması imkansızdır. Bütün erişim adresleri yerine sadece son zamanlarda sık sık erişilmiş olan bellek adresleri tutulmaktadır. Erişilmiş olan adresler arasındaki mesafe ve son zamanlarda erişilmiş olan adreslerin bilgileri Referans Tahmin Tablosu (Reference

Prediction Table - RPT) adı verilen ayrı bir önbellekte kayıt edilmektedir. Referans tahmin tablosunda komut etiketi, bir önce erişilmiş olan adres, mesafe ve komutun o anki durumunu gösteren alanlar bulunmaktadır (Şekil 4.4). Ayrıca Şekil 4.5'de de tablodaki kayıtların durum bilgisinin nasıl değiştirildiğini belirleyen diyagram gösterilmektedir.



Şekil 4. 4 Referans Tahmin Tablosunun Yapısı (VanderWiel ve Lilja, 2000)



Şekil 4. 5 RPT tablosundaki kayıtların durumlarının belirlenmesi (Metcalf, 1993)

Tablo içerisinde tutulan her kayıt işlemcinin ürettiği adrese (Program Counter - PC) yani komut etiketine göre dizilmiştir. Ana bellek komutu m_i ilk defa için işletildiğinde, RPT tablosundaki bu komutun kaydının durumu başlangıç (initial) olarak atanır. Bu durum, o komut için henüz önceden getirme işleminin başlamadığını göstermektedir. Eğer m_i komutu RPT tablosundaki kaydı silinmeden önce tekrar işletilirse, mesafe değeri RPT tablosunda kayıtlı bir önceki adres değeri ile şuan ki etkin adres değeri çıkarılarak hesaplanır (VanderWiel ve Lilja, 2000).

4.3.2 Yazılım Tabanlı Önceden Getirme İşlemi (Software Prefetching)

Yazılım tabanlı teknikler, önceden getirme işlemini başlatmak için kaynak kodun içerisine çeşitli komutlar eklerler. Bu komutlar önceden getirilecek bloğun adresini ve blok boyutunu belirlerler. Programcı veya derleyici tarafından otomatik olarak bu komutlar yerleştirilebilmektedir.

Blokları önceden getirmeyi sağlayan gerekli komutların eklenip eklenmediğini belirlemek için derleyicinin ıskaların ne zaman olacağını tahmin etmesi gerekmektedir. Derleyici ıska olacağını tahmin edemediği sürece kodun içerisine veriyi önceden getirmek için gerekli olan komutlar eklenemez. Eğer ıska olacağı tahmin edilebilirse, derleyici önceden getirme komutlarının nereye ekleneceğini belirler. Derleyici, önceden getirme komutlarının nereye eklenirse ana belleğe erişim süresinde azalma olacağını tahmin etmek zorundadır.

Çünkü derleyici ıska olacağını ve belleğe erişim süresindeki gecikmeleri tahmin etmelidir. Yazılım tabanlı önceden getirme işlemi sınırlı bir alanda çalıştığı için çok iyi sonuçlar vermektedir. Bu teknikte, bilimsel amaçlı programlarda diziler üzerindeki düzenli hesaplamalar için derleme zamanında içerik çözümleme fonksiyonları gerekmektedir. Pratikte yazılım tabanlı önceden getirme işlemlerini kullanmak çok zor olabilmektedir.

Yazılım tabanlı önceden getirme işleminin avantajı en az seviyede donanım gereksinimi duyması, önemli verilerin önceden getirilmesini ve ne zaman getirileceğinin belirlenmesi açısından daha fazla esneklik sağlamaktadır (Crago, 1997; Lim, 1999).

4.3.3 Donanıma Dayalı ve Yazılıma Dayalı Önceden Getirme İşlemleri Arasındaki Farklar

Yazılıma dayalı ve donanıma dayalı önceden getirme teknikleri arasındaki en önemli farklılık, yazılıma dayalı tekniklerin donanıma dayalı tekniklere göre daha az donanım desteğine gereksinim duymalarıdır. Bu yüzden, yazılıma dayalı teknikler daha ucuz geliştirilme imkanlarına sahiptir. Bir yazılıma dayalı teknik, önceden getirilmesi gereken her veri için bir komuta (prefetch instruction) ve bu komutun hangi adrese yerleştirileceğini hesaplaması gerekmektedir. Bu nedenle kodun boyutu hem dinamik hem de statik olarak artmaktadır. Ayrıca yazılıma dayalı tekniklerde yeniden derleme işlemi gerekmekte ve bu da genel derleme zamanını arttırmaktadır.

Bir diğer önemli farklılık, üretilen bir adrese erişim tipi yazılıma dayalı tekniklerde derleme zamanında belirlenmektedir. Bu da derleyicinin sık sık mesafe değerini belirlemesine neden olur. Donanıma dayalı teknikler ise her bir erişimin tipine göre daha karmaşık olmaktadır ve ayrıca erişim tipi ancak bir miktar yürütme işleminde sonra belirlenmektedir. Bu yüzden donanıma dayalı teknikler uygun sabit duruma geçmeden önce yazılıma dayalı tekniklere göre başlangıçta daha sık ıska olmaktadır. Donanıma dayalı teknikler bazı bilgileri çok çabuk ve verimli bir şekilde işlemesi gerekmekte; bu da önceden getirme işlemlerinde uygun kararların sırasıyla alınmasını sağlamaktadır (Gornish, 1995).

4.4 Ne Zaman, Nereye, Ne Önceden Getirilecek

- *Ne zaman önceden getirme işlemi başlamalı:* Önceden getirme işleminin ne zaman yapılacağı çok önemlidir. Eğer önceden getirme işlemi çok erken gerçekleştirilirse, verinin getirildiği bir üst seviyedeki bellekte bulunan yararlı bir verinin üzerine yazılabilir. Bu da yararlı verinin daha

sonra ihtiyaç olunduğunda tekrardan alt seviyedeki bellekten getirilmesine sebep olur ve ekstradan makine çevrimi gerektirir. Eğer önceden getirme işlemi çok geç gerçekleştirilirse, işlemcinin daha fazla beklemesine neden olur.

- *Önceden getirilen veriler nereye yerleştirilmeli:* Bellek hiyerarşisinde önceden getirilen verinin nereye yerleştirileceğinin kararı çok önemlidir. Performans açısından bir artış sağlanmak isteniyorsa; veri, bellek hiyerarşisine göre bir üst seviyedeki belleğe yerleştirilmelidir. Önceden getirme tekniklerinin çoğunda getirilen veri çeşitli türlerdeki önbelleklere yerleştirilebilmektedir. Geriye kalan az sayıdaki önceden getirme tekniklerinde de veriler tamponlar içerisine yerleştirilmektedir. Bu sayede önbellekler vaktinden önce boşaltılsa da veri kaybı yaşanmaz ve önbellek kirliliği yaşanmamış olur.

- *Ne önceden getirilmeli:* Önceden getirilecek veriler sözcükler, önbellek blokları, ana bellek blokları veya program veri nesneleri şeklinde getirilmektedir. Getirilen verinin miktarı önbellek ve ana bellek sistemi tarafından belirlenir. Tek-işlemciler'de (uniprocessors) önceden getirilecek verilerin miktarı için en uygun boyut önbellek bloğudur. Daha büyük boyutlu bellek blokları ise, çoklu işlemcilerde (multiprocessor) bulunan dağıtık bellekler arasındaki veri iletişiminin başlangıç maliyetini azaltmak için kullanılır (VanderWiel ve Lilja, 2000).

BÖLÜM 5

ÖNBELLEKLERDE SIZINTI GÜCÜN AZALTILMASI

Önbellekler hem performans hem de enerji açısından çok kritik bileşenlerdir; ayrıca mobil ve gömülü sistemlerde giderek kullanımı artmaktadır. Performans açısından, önbellekler yürütme işleminde kritik bir öneme sahiptir ve önbelleklerin isabet-ıska karakteristikleri genelde uygulamaların toplam performanslarını belirlemede önemli rol oynamaktadır. Enerji açısından, önbellekler çip üzerindeki enerji tüketiminin toplamda %50-60'ndan sorumludur (Montenaro ve diğ., 1996). Bu yüzden performans ve enerji karakteristiklerini optimize etmek çok önemlidir ve gelecekte daha fazla önem kazanacağı umulmaktadır.

Önbelleklerin öneminden dolayı, hem performans hem güç açısından bir önceki çalışmaların çoğunun belirgin hedefleri vardır. Performans iyileştirme teknikleri kabaca gecikme-azaltma teknikleri (kod/veri optimizasyonu (Gupta ve diğ., 1991; O'Boyle ve Knijnenburg, 1998)), karmaşık önbellek mekanizmaları (çalışma zamanında uyarlamalı önbellek hiyerarşisi yönetimi (Johnson ve Hwu, 1997), özel önbellek hiyerarşisi tasarımı (Cotterell ve Vahid, 2002), değişken önbellekler (Ranganathan ve diğ., 2000)) ve gecikme saklayıcı teknikler (önceden getirme işlemi (VanderWiel ve Lilja, 2000; Cooksey ve diğ., 2002)) olarak ayrılmaktadır. Enerji-koruma teknikleri ise wordline/bitline bölümleme (parçalama) (Ghose ve Kamble, 1999), blok tamponlama (block buffering) (Su ve Despain, 1995) ve önbellek bloklarının enerjisinin kesilmesi tekniklerinden oluşmaktadır. Bu teknikler birbirlerinden bağımsız bir şekilde geliştirilirken, hem performansın hem de gücün aynı sistemde birlikte var olmalarından dolayı birbirleriyle etkileşimlerine geçmişte çok fazla dikkat edilmemiştir. Örneğin, sızıntı enerji kullanımını azaltmak için kullanılan, önbellek bloklarının enerjisinin kesilmesiyle veri/komut önceden getirme işleminin birbirine etkisi tam olarak açık değildir. Birbirlerine karşı olan etkileşim çok önemlidir; çünkü gömülü sistemlerde bugün yüksek performans ve düşük güç kullanımı istenmektedir. Performans iyileştirmelerinin güce etkisi ve güç

iyileştirmelerinin performansa etkisi belirlenmeksizin bir gömülü sistemin optimizasyonu ve tasarımında gerekli değişiklikler gerçekleştirilemez.

İşlemci ve ana bellek arasındaki hız farkını kapatmak için orjinal olarak tasarlanmış önbellekler, sık sık erişilen verileri yavaş DRAM'de tutmak yerine ihtiyaçları daha iyi karşılayan hızlı SRAM'de tutmaktadır; böylece sistem gücü üzerinde önemli etkilere sahip olmuş olurlar. İşlemci teknolojisindeki yenilikler nedeniyle (daha küçük kapasitans, daha düşük depolama ve eşik voltajı) sızıntı enerji 0.10 mikron ve daha küçük teknolojiye sahip işlemcilerin toplam çip gücünün büyük bir kısmını oluşturmaktadır (Chandrakasan ve diğ., 2001). Sızıntı enerjinin büyük boyutlu yapılar için, örneğin önbellekler gibi, belirli etkileri vardır. İşlemcilerdeki transistör sayılarının artmasıyla; önbelleklerde bulunan transistör sayısı işlemcideki toplam transistör sayısının %90'dan fazlasını kaplamaktadır. Düşük güçteki sistemler önbelleklerin sızıntı enerjisini iyileştirmek için donanım ve yazılım tekniklerine güvenmektedir.

Bu tezde yapılan çalışmayla iki ana amaç gerçekleştirilmeye çalışılmaktadır. İlki, belirgin bir performans iyileştirmesi (önceden getirme işlemi yardımıyla) ve güç iyileştirmesi (önbellek bloklarının enerjilerinin kesilmesiyle) üzerine odaklanmak ve bu iki optimizasyonun performans ve güç açısından birbirlerini nasıl etkilediklerini vurgulamak. İkincisi ise, donanıma dayalı üç yeni önceden getirme tekniği gerçekleştirilmesi amaçlanmaktadır. Önerilen yaklaşımlar, önceden getirilen bloklar için farklı bozulma zaman aralıklarını kullanarak önceden getirilen bloklar üzerindeki bilgiyi kullanırlar. Başka bir deyişle, önerilen tekniklerde önceden getirilen bloklar ve normal bloklar load komutunun işletilmesi sırasında önbelleğe getirilirler. Bu önceden getirme işleminin enerji tüketimi üzerindeki muhtemel negatif etkisini azaltır. Daha fazla detaya inecek olursak, önceden getirme işlemi veri/komutları ana bellekten önbelleğe ihtiyaç olunmadan önce getirirler. Bu işlem özellikle ihtiyaç olunmayan verilerin önceden getirilme oranı yüksekse, dinamik enerji tüketimini etkileyebilir. Ama daha önemlisi bir önbellek bloğu düşük-sızıntı güç modunda getirilmeye çalışılırsa; önceden getirme işlemi o bloğu normal işlem moduna yani aktif moda geçmeye zorlar, böylece sızıntı güç tüketimi etkilenmiş olur.

Yapılan bu tez çalışmasında amaç iki seviyeli önbellek hiyerarşisinde bir takım testler kullanarak bu etkileşimleri incelemek ve farklı önceden getirme işlemine duyarlı önbellek enerjisini azaltma tekniklerini kullanarak bu etkileşimin muhtemel yararlarını ölçmektir. Bir simülatör programı ve SPEC2000 (SPEC) testlerini kullanarak performans ve enerji arasındaki etkileşimlerin önemini belirten test sonuçları grafiklerle çizilecek; ve önerilen optimizasyon tekniklerinin aynı sistem içerisinde önceden getirme işlemi yapılarak ve önbellek bloklarının enerjisi kesilerek güç-performans değişimlerinin nasıl geliştiği gösterilecektir.

5.1 Önceden Getirme İşlemi ve Önbellek Bloklarının Enerjisinin Kısılması

Önceden getirme işlemi modern mikro işlemcilerde ana belleğe erişim sürelerini kısaltmak için kullanılan, ihtiyaç olunmadan önce verinin/komutun işlemciye getirilmesini sağlayan bir tekniktir. Literatürde kullanılan birçok önceden getirme işlemi vardır. Bu teknikler çalışma şekillerine göre donanıma dayalı ve yazılıma dayalı teknikler olmak üzere iki gruba ayrılmaktadır. Bir önceki bölümde yaygın şekilde kullanılan donanıma ve yazılıma dayalı tekniklerden bahsetmiştik.

Bu tez çalışmasında, etiketlemeli önceden getirme işlemi (tagged prefetching) olarak adlandırılan donanıma dayalı bir önceden getirme tekniği kullanılacaktır. Bu teknik sonraki bir bloğu daha önceden önbelleğe getirme mantığına göre çalışmaktadır. Bir b bloğuna erişildiğinde, b+1 bloğunun önceden getirilme işleminin başlatılması şeklindedir. Yalnız (b+1). bloğun önbelleğe getirilmesi iki farklı duruma göre çalışmaktadır. Birincisi, b bloğuna erişildiğinde, eğer bir ıska olmuşsa. İkincisi, eğer b bloğu, önbelleğe önceden getirme işlemi ile getirildiyse ve b bloğuna ilk kez erişiliyorsa; (b+1). bloğun önceden getirilme işlemi tetiklenmiş olur. Bu yaklaşım, L1 ve L2 önbellek hiyerarşisinde hem verileri hem de komutları önceden getirmek için kullanılacaktır.

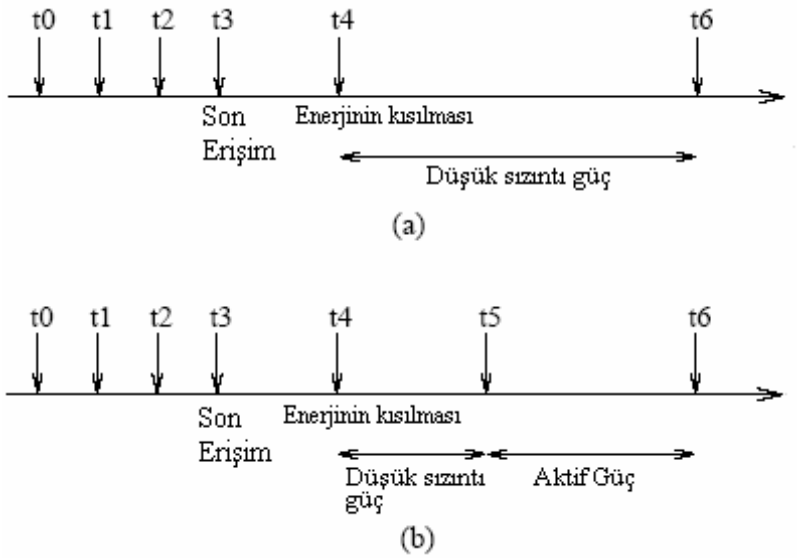
Önceden getirme işlemi yaparak performans geliştirilmeye çalışılırken, sızıntı kontrol mekanizmaları ile de enerji tüketimi azaltılmaya çalışacaktır. Sızıntı enerjiyi azaltmak için önemli bir gereksinim olan hem durum yok edici (State-destroying -

DY) hem de durum koruyucu (state-preserving - DK) sızıntı kontrol mekanizmaları kullanılmayan kaynakların (önbellek blokları) belirlenmesi için uygun mekanizmalardır. Yang ve diğ. (2001) önbellek boyutunu dinamik olarak değiştirerek önbelleğin sızıntı enerjisini azaltmaya çalışmışlardır (Yang ve diğ., 2001). Sızıntı enerji tüketimini azaltmak için bir diğer yol ise önbellek içerisindeki verileri düşük besleme geriliminde tutarak, önbelleğin kullanılmayan bölümlerine dinamik besleme voltajı uygulamaktır (Kim ve diğ., 2002a). Kim ve diğ. (2004) komut önbelleklerinde sızıntı enerjiyi azaltmak için veri saklama teknikleri üzerinde çalışmışlardır (Kim ve diğ., 2004). Azizi ve diğ. (2003) düşük sızıntı güce sahip asimetrik-hücreli önbellekler (asymmetric-cell caches - ACC) üzerinde çalışmışlardır. Yapılan bu çalışmada temel fikir, veri/komut önbelleklerindeki bitlerin çoğunun sıfır olması fikrine dayanmaktadır. ACC önbelleklerde herhangi bir hücre içerisindeki bit sıfır olduğunda, transistörlerdeki güç azaltılarak sızıntı enerji tüketimi düşürülmektedir (Azizi ve diğ., 2003). Diğer taraftan, kullanılan durum yok edici sızıntı kontrol mekanizmaları da bulunmaktadır. Örneğin, Kaxiras ve diğ. (2001) önbellekler için durum yok edici sızıntı enerji azaltma tekniği geliştirmişlerdir. Bu teknik, önbellek bozulması olarak adlandırılan, önbellek bloğunun ölü periyoda girdiğinde beslemesi kesilmek suretiyle güç tasarrufu yapılması prensibine dayanmaktadır (Kaxiras ve diğ., 2001). Li ve diğ. (2003) bloklardaki verilerin önbellek hiyerarşisinin farklı seviyelerinde bu blokların kopyalarının oluşturulması üzerine çeşitli teknikler geliştirmişlerdir. L1 önbelleğinde olup da, L2 önbelleğinde de bulunan bloklar için hem durum koruyucu hem de durum yok edici sızıntı kontrol mekanizmaları kullanmışlardır. S-DK-U (Spekülatif-Durum Koruyucu-Uyuşuk) olarak adlandırılan bu yöntemde sızıntı enerji azaltma çalışmalarındaki en iyi sonuçlar elde edilmiştir. Bu yönteme göre, herhangi bir veri L2'den L1 önbelleğe getirildiğinde, verinin bulunduğu L2 önbellekteki blok DK sızıntı kontrol moduna sokulur (Li ve diğ., 2003).

Yapılan bu tez çalışmasında, önbellek bozulması ve S-DK-U teknikleri birleştirilerek L1 ve L2 önbelleklerindeki sızıntı enerjinin azaltılmasına çalışılacaktır. Özelikle, L1 önbellek için önbellek bozulması tekniğini, L2 önbellek için ise hem önbellek bozulması hem de S-DK-U tekniği kullanılacaktır. Veri L2 önbellekten L1

önbelleğe getirildikten sonra, L2 önbellekteki blok hemen DK sızıntı kontrol moduna geçirilecektir. Buna ek olarak, eğer getirilen bloğa uzun bir süre erişilmezse, bloğun durumu DK modundan DY moduna sokulur. Eğer L2 önbellekteki bir bloğun bütün kopyaları DY modunda ise, o blok geçersiz sayılır ve LRU yer değiştirme algoritmasıyla çıkarılabilecek aday bloklar arasına konulur. Sadece L1 önbelleği için önbellek bozulması tekniği kullanıldığında, yürütme anının herhangi bir zamanında bu önbelleğin blokları DK moduna geçirilemez.

Önceden getirme işleminin ve önbellek bloklarının enerjisinin kısılması işlemleri birbirleriyle etkileşim içine girebilmektedir. Şekil 5.1(a)'da tipik bir veri önbellek bloğunun yaşam ömrü gösterilmektedir. t0 anında, yeni bir veri bloğa yerleştirilmektedir. Bu veriye sırasıyla t1, t2 anında erişilmekte ve sonradan t3 zamanında da son erişim yapılmaktadır. Bu bloğa bir süreliğine erişilmediğini ve önbellek bozulma periyodundan sonrada t4 anında düşük sızıntı güç moduna geçtiğini farz edelim. Yeni bir veri bu bloğa getirilinceye kadar, önbellek bloğu düşük sızıntı güç modunda tutulur ve t6 anında da yeni bir veri bu bloğa getirilir. Şimdi Şekil 5.1(b)'de t5 anında önbellek bloğuna bir önceden getirme işleminin yapıldığını varsayalım. Bu durumda, blok t5 anında aktif moda geçirilmek zorundadır. Getirilen veri t6 anında kullanılır. t5-t6 zaman aralığında önbellek bloğu aktif durumda tutulmakta ve tam sızıntı enerji tüketmektedir. Oysaki Şekil 5.1(a)'da aynı zaman aralığında bu blok düşük sızıntı güç modunda tutulmaktadır. Bu örnek önceden getirme işleminin önbellek sızıntısı üzerindeki muhtemel olan negatif etkisini göstermektedir. Önceden getirilen verinin sızıntı tüketimini arttıracağı durumlarda, bu bloğun hiçbir şekilde kullanılmaması mümkündür.



Şekil 5. 1 Önbellek bloğu için iki farklı erişim senaryosu. Yatay doğru, zaman çizgisini göstermektedir

5.2 Kullanılan Araçlar

Yapılan tez çalışmasında, önceden getirme işlemi ve sızıntı enerji azaltma tekniklerini gerçekleştirmek için SimpleScalar 3.0 (Burger ve Austin, 1997) simülatörü kullanılmıştır. SimpleScalar yazılımı, işlemciler üzerinde uygulama programlarının hızlı simulesini sağlayan bir simülatördür. Bu simülatör programı içerisinde biz sim-outorder bileşenini kullandık.

Çizelge 5.1'de yapılan çalışmada kullanılacak ana simülasyon parametreleri listelenmektedir. Mikroişlemcilerde 70nm teknolojisini kullanılacak ve L1-L2 önbelleklere erişmede harcanan dinamik enerji miktarı da CACTI 3.2 (CACTI) simülatör programı yardımıyla ölçülecektir. Ayrıca sızıntı faktörü (leakage factor) isminde bir k değişkeni kullanılacaktır. Bu k değişkeni, L1 önbelleğinin her çevriminde harcanan sızıntı enerji ile her erişimde tüketilen dinamik enerji arasındaki oranı ifade etmektedir. Daha büyük k değişkeni değerleri gelecekteki sızıntı enerji miktarının çipin güç tüketiminin büyük bir kısmını oluşturacağını ifade etmektedir. Yapacağımız çalışmada k=1 olarak farz edeceğiz. Ama sonradan farklı k değişkeni değerleriyle elde edilen sonuçlar gösterilecektir. Buna ek olarak L2 önbelleğinden L1

önbelleğine gönderilen bloğun sızıntı enerji miktarıyla L1 önbelleğindeki bu blokla sızıntı enerjisinin aynı olduğunu farz ediyoruz.

Çalışmalarımızda SPEC2000 performans ölçüm testlerinden rasgele seçilmiş 5 tanesi kullanılacaktır. Herhangi bir testi simule etmek çok uzun süre aldığından, ilk 300 milyon komutu hızlı çalıştırıp sonraki 200 milyon komutu simule edeceğiz. Çizelge 5.2'de yapılacak testlerin önemli özellikleri gösterilmektedir. Bu çizelgedeki 3. ve 4. sütunlar L1 komut önbelleğinin dinamik ve sızıntı enerji tüketimlerini, 5. ve 6. sütunlar L1 veri önbelleğine ilişkin sonuçları, 7. ve 8. sütunlar ise L2 önbelleğiyle ilgili enerji sonuçlarını, son sütun ise sızıntı enerjinin L1-L2 önbellek hiyerarşisindeki toplam enerji tüketimine katkısını göstermektedir. L2 önbelleğin daha büyük boyutlu olması ve bu önbelleğe daha az erişilmesi nedeniyle L1 önbellekle karşılaştırıldığında; L1-L2 önbelleklerindeki sızıntı enerjinin dinamik enerjiye baskın olduğu görülmektedir. Ayrıca 4. ve 6. sütunlardaki değerler her test için aynı çıkmaktadır. Bunun nedeni L1 komut önbelleği ile L1 veri önbelleğinin boyutlarının birbirine eşit olmasından kaynaklanmaktadır.

Çiz. 5. 1 Temel konfigürasyon

İşlemci çekirdeği	
Fonksiyonel birimler	4 integer ve 4 FP ALU 1 integer çarpıcı/bölücü 1 FP çarpıcı/bölücü
LSQ boyutu	64 komut
LRU boyutu	64 komut
Fetch/Decode/Issue/Commit genişliği	4 komut/çevrim süresi
Fetch kuyruk boyutu	4 komut
Önbellek ve ana bellek yapısı	
L1 komut/veri önbelleği	32KB, 32 byte blok boyutu 2-yollu, 1 çevrim gecikme süresi
L2 önbellek	1MB birleştirilmiş, 128 byte blok boyutu 2-yollu, 10 çevrim gecikme süresi
Veri/Komut TLB	128 kayıt, tam çağrışımlı 30 çevrim ıska gecikme süresi
Ana bellek	100 çevrim gecikme süresi
Enerji yönetimi	
Teknoloji	0,07 micron
Besleme Gerilimi	1,0V
Her L1 önbelleğine erişim için dinamik enerji tüketimi	0,186nJ
Her L1 önbellek bloğunun sızıntı enerjisi veya her aktif çevrimdeki L2 önbellek alt-blok sızıntı enerjisi tüketimi	0,182pJ
Her bekleme çevriminde her L2 önbellek alt-bloğu için sızıntı enerji tüketimi (durum koruyucu)	0,018pJ
Her L1 önbellek bloğu için sızıntı enerji tüketimi veya her bekleme çevrim süresinde L2 önbellek alt-bloğu için sızıntı enerji tüketimi (durum yokedici)	0pJ

Çiz. 5. 2 Çalışmada kullanılan testler ve önemli özellikleri

Testler	Çalışma Süresi (milyon cycle)	il1 Enerji (mJ)		dl1 Enerji (mJ)		l2 Enerji (mJ)		Top. sız. enerji katkısı (%)
		Dinamik	Sızıntı	Dinamik	Sızıntı	Dinamik	Sızıntı	
172.mgrid	106,25	37,34	19,76	14,34	19,76	5,31	632,40	%92,18
173.applu	131,98	37,43	24,54	15,34	24,54	12,16	758,28	%92,78
191.fma3d	82,81	43,25	15,40	7,13	15,40	1,01	492,80	%91,06
175.vpr	159,63	52,23	29,69	15,13	29,69	5,28	950,08	%93,29
256.bzip2	115,70	37,25	21,52	15,74	21,52	5,63	688,64	%92,58

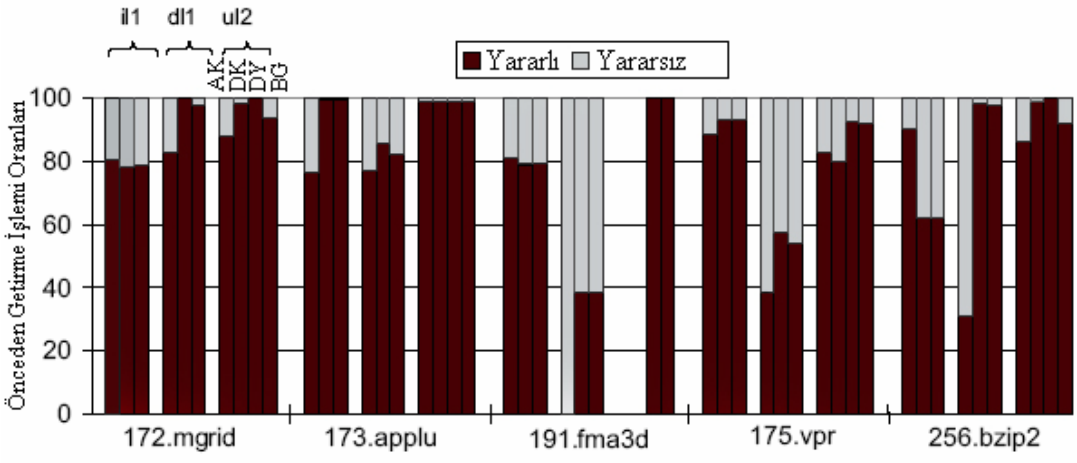
5.3 Önceden Getirme İşlemi – Blok Enerjisinin Kısılması Etkileşimi

5.3.1 Önceden Getirme İşlemini ve Önbellek Bloklarını Tanımlamak

Hem performans hem de sızıntı enerji açısından, önceden getirilen blokların başka bloklarla yer değiştirmeden önce kullanılması çok önemlidir. Bu yüzden, önceden getirme işlemini iki gruba böleriz: yararlı (useful) ve yararlı-olmayan (non-useful) önceden getirme işlemi. Eğer önceden getirilmiş bir blok erişilmeden önce önbellekten atılırsa, yapılmış olan önceden getirme işleminin gereksiz olduğu anlamına gelir ve yararsız önceden getirme işlemi olarak adlandırılır. Yararlı-olmayan önceden getirme işlemleri veri yolunu meşgul ettiklerinden dolayı performansta bozulmalara ve fazladan ıska oluşmasına neden olurlar. Ayrıca bu işlemler gereksiz önbellek erişimlerinden dolayı dinamik enerji tüketimini ve eğer sızıntı koruma modundaki bir blok önceden getirilirse de sızıntı enerji tüketimini arttırırlar.

Şekil 5.2'de L1 komut önbelleği, L1 veri önbelleği ve L2 önbelleği blokları için farklı güç tüketimlerinde yararlı ve yararsız önceden getirme işlemlerinin katkıları gösterilmektedir. Yapılan bu tez çalışmasında önbellek blokları üç çeşit güç tüketim durumundan birinde olacaklardır: aktif mod (AK) (tam sızıntı enerji tüketimi olur), durum koruyucu mod (DK) (aktif durumda tüketilen enerjinin %10'nu kadar sızıntı enerji tüketilir), durum yok edici mod (DY) (hiç enerji tüketimi olmaz). Yapılan bu çalışmada L1 önbellek blokları herhangi bir zamanda AK ve DY güç durumlarından birinde, L2 önbellek blokları ise üç güç durumundan birinde olabilirler. Şekil

5.2'deki testlerin her birinde üç grup dikey çubuklar bulunmaktadır. 1. ve 2. grup çubuklar L1 komut ve veri önbelleğine, 3. grup ise L2 önbelleğine karşılık gelmektedir. L1 önbellekleri için, ilk iki çubuk AK ve DK durumdaki önbellekleri, üçüncü çubuk ise bütün güç durumları için yararlı ve yararsız önceden getirme işlemlerinin katkılarını; L2 önbellek için olan dört çubukta AK, DK, DY ve BG (Bütün güç durumlarının birleşimi) durumlarındaki yararlı ve yararsız önceden getirme işlemlerinin katkılarını göstermektedir.

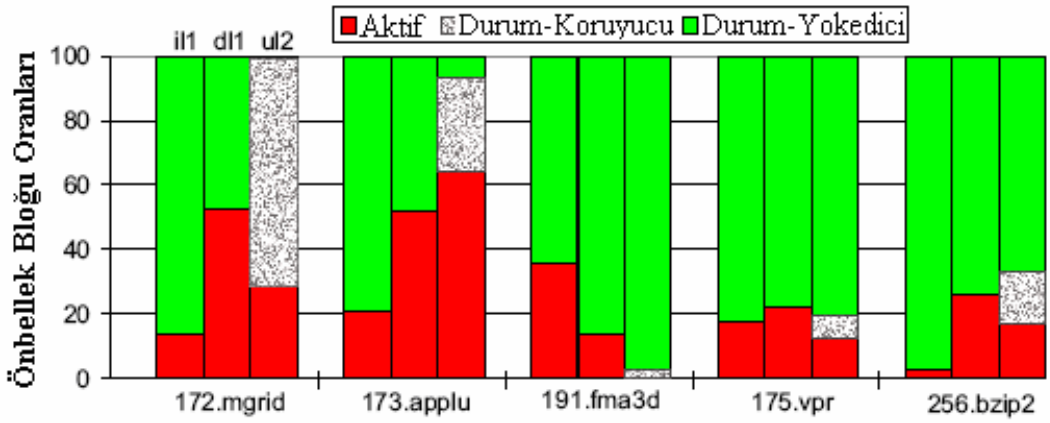


Şekil 5. 2 Yararlı ve yararsız önceden getirme işlemi oranları

Şekilden görüldüğü gibi önceden getirilmiş olan blokların önemli bir kısmı işe yarar bloklardır. Ortalama olarak yararlı önceden getirme işleminin oranı L1 komut önbelleği için %82,47, L1 veri önbelleği için %74,04 ve L2 önbelleği için ise %95,23'dür. Bu sonuçlar önceden getirme işleminin büyük bir kısmının yararlı olduğunu göstermektedir. Bu işlemler toplam sızıntı enerji tüketiminde artışa neden olmaktadır; gerçekleştirilme sayılarının azaltılmasıyla yapılacak olan bir optimizasyon pratikte enerji tüketimi için çok yararlı olabilir.

Şekil 5.3'te bizim mimarimizde farklı önbellekler için AK, DK ve DY güç durumlarında olan önbellek bloklarının oranı gösterilmektedir. Yaptığımız çalışmalarda blok/alt-blokları DY moda sokmak için Kaxiras ve diğ.(2001)'lerinin kullandıkları eşik değerlerini kullandık. Örneğin, L1 önbellek için 10 bin çevrim ve L2 önbellek için ise bir milyon çevrim. Bu eşik değerlerine bozulma süresi (decay interval) adı verilmektedir (Kaxiras ve diğ., 2001). Şekil 5.3'teki her bir test için, 1.

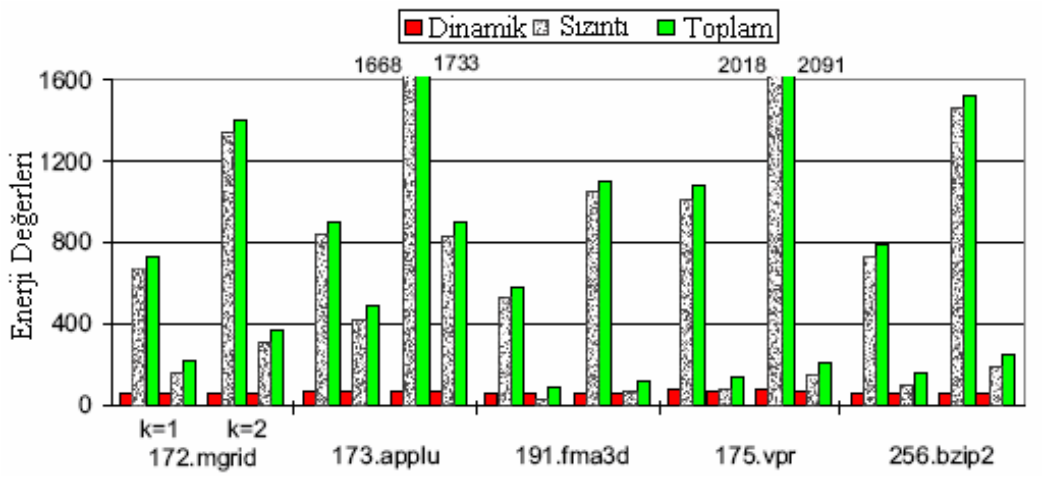
ve 2. çubuklar L1 komut ve L1 veri önbellekleriyle, 3. çubuk ise L2 önbelleğine karşılık gelmektedir. Bu şekilde iki önemli noktaya dikkat etmemiz gerekiyor. Birincisi, L1 komut önbelleği blokları DY modunda L1 veri önbelleği bloklarından daha fazla çevrim harcamaktadır. Bu, komut lokalitesinin veri lokalitesinden daha fazla belirgin olduğunun sonucudur. İkinci nokta, L2 veri önbellek blokları çalışma sürelerinin çoğunda düşük sızıntı güç modunda (DK-DY) durmaktadır. Ortalama olarak, L2 önbellek blokları yürütme zamanının %24,32'sini AK durumda geçirmektedir. Bu da sızıntı enerjinin önemli bir kısmının azalmasını sağlamaktadır.



Şekil 5. 3 Farklı durumlarda gerçekleştirilen önceden getirme işlemi oranları

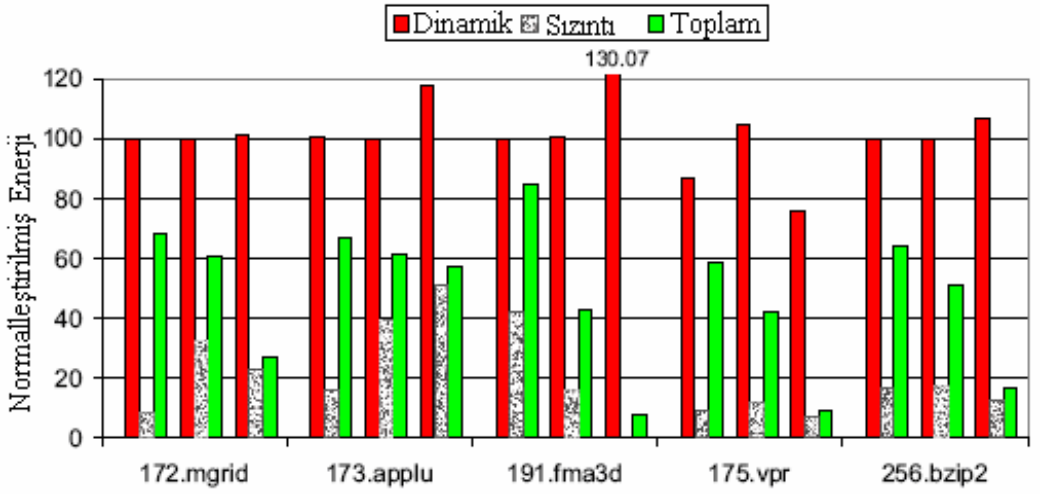
5.3.2 Enerji Sonuçları

Bu bölümde, iki farklı k değeri için önbellek hiyerarşimize ait enerji değerleri verilmektedir. Hatırlanacağı gibi büyük bir k değeri gelecekteki sızıntı tüketiminin daha fazla önemli olduğu düşünülen eğilimleri elde etmek için kullanılabilir. Şekil 5.4'te iki farklı k değeri için hem orjinal hem de optimize edilmiş testler için enerji değerleri gösterilmektedir. Her bir test iki grup çubuktan oluşmaktadır; birinci grup çubuklar k=1 için (birincisi), ikinci grup çubuklar ise k=2 için (ikincisi) enerji değerleriyle ilişkilidir. Her bir k- grubundaki ilk üç çubuk orjinal kod için sırasıyla dinamik, sızıntı ve toplam enerjinin kesin değerlerini vermektedir; oysa son üç çubuk optimize edilmiş kod (önceden getirme işlemi uygulanan kod) için aynı değerleri vermektedir.



Şekil 5. 4 Farklı k değerleri için enerji tüketimleri

Bu şekle bakarak şu gözlemleri yapabiliriz. İlk olarak, önceden getirme işleminin ve sızıntı kontrolünün neden olduğu dinamik enerji oldukça küçük olduğundan ihmal edilebilir. Bunun nedeni, optimizasyonlar sonucu ortaya çıkan önbellek erişim sayısındaki küçük artıştır. İkinci olarak, sızıntı enerji orjinal kodlar için dinamik enerjiyi bastırırken, optimizasyonlar sızıntı enerjiyi önemli derecede azaltır ve optimize edilmiş kodlar için dinamik enerji bütün enerji tüketiminin önemli bir bölümü haline gelir. Bu temelde iki farklı nedenden oluşmaktadır. Birincisi, sızıntı mekanizmaları oldukça etkilidir ve daha sonra tekrar ele alacağımız gibi, önceden getirme işlemi önbellek transistörlerinin daha kısa zaman aralıklarında sızıntı yapmasını sağlayarak yürütme performansını arttırır. Son gözlem göstermektedir ki, k değeri arttığı zaman, optimizasyonları uyguladığımızda toplam enerjideki tasarruf da artmaktadır. Örneğin, optimizasyonları uyguladıktan sonra, 256.bzip2 ve 172.mgrid testleri için sırasıyla k=1 değerinde toplam enerjide %81,12 ve %71,43, k=2 değeri için ise %84,43 ve %74,94 tasarruf elde edilmektedir.



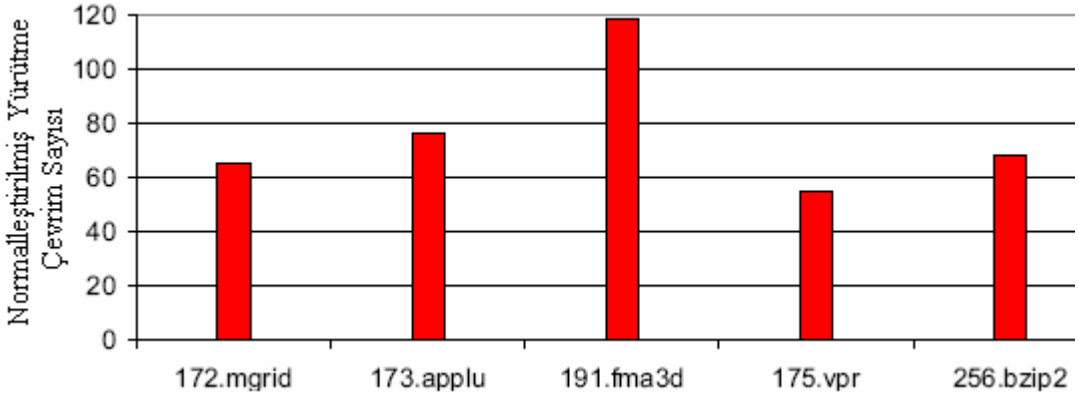
Şekil 5. 5 k=1 olduğunda normalleştirilmiş enerji değerleri

Şekil 5.5'te, orjinal kodların enerji tüketimine göre optimize edilmiş kodların normalleştirilmiş enerji tüketimleri k=1 için verilmiştir. Şekilde her bir testte, birinci ve ikinci grup çubuklar sırasıyla L1 komut önbelleği ve L1 veri önbelleği, son gruptaki çubuklar ise L2 önbellek içindir. Her bir gruptaki çubuklar, soldan sağa doğru dinamik enerji, sızıntı enerji ve toplam enerjiye karşılık gelmektedir. Bu şekilden görüleceği üzere, bütün testler için, sızıntı kontrol ve önceden getirme işlemi mekanizması ile birlikte gösterilen dinamik enerji ek yükü o kadar küçüktür ki L1 önbellekleri için göz ardı edilebilir. Fakat bu durum L2 önbelleği için geçerli değildir. Örneğin, optimizasyonlarımız 173.applu ve 191.fma3d testleri için L2 önbelleğinin dinamik enerjisini %17,83 ve %30,07 oranında arttırmaktadır. İlginç olarak, 175.vpr testi için L1 komut ve L2 önbelleğindeki dinamik enerji tüketimi göz önüne alındığında optimizasyonlardan fayda sağlanmaktadır. Bunun nedeni, bu test için olan önceden getirme işlemi söz konusu olan önbellek için ıska sayısını azaltmaktadır. Bu sebeple, toplam önbellek erişim sayısı azalmaktadır. Sızıntı sonuçlarına baktığımızda, her bir test için bütün önbelleklerde dikkate değer bir enerji tasarrufu olduğunu görmekteyiz. Ortalama tasarruf değerleri sırasıyla, L1 komut, L1 veri ve L2 önbellekleri için %81,38, %76,19 ve %81,11'dir. Şekil 5.5'teki önemli bir gözlem ise, sızıntı enerji tüketimi L2 önbelleğindeki dinamik enerji tüketimini önbellek bloklarının enerjisinin kısılmasından sonra kontrol altına

almaktadır. Bu, önbellek için orjinal kodlarda sızıntı enerji tüketiminin çok büyük olmasına atfedilebilir.

5.3.3 Çalışma Süresi

Önceden getirme işlemi, blokları önbelleğe önceden getirerek potansiyel olarak performansı arttırırken; sızıntı kontrol mekanizmaları genellikle önbellek bloğuna erişilmeden önce DK-DY sızıntı modundan tam güç moduna geçildiğinde oluşan ek enerji yükünden dolayı performansı geriletir. Yapılan bu tez çalışmasında, bir önbellek bloğunu aktif moda getirmek için tek bir ekstra çevrim süresinin gerektiği farz edilmiştir. Şekil 5.6 optimize edilmiş kodlar için normalleştirilmiş çalışma süresini göstermektedir. Ortalama olarak, optimize edilmiş kodlarla %24,35 oranında performans artışı sağlanmaktadır. Önceden getirme işlemi bütün testlerin performansını önemli derecede arttırmaktadır (191.fma3d dahil olmak üzere). Bu grafikteki çevrim değerleri, önceden getirme işlemi ve sızıntı kontrolünün her ikisi tarafından optimize edilmiş testler içindir. Burada, optimize edilmiş testler için çalışma sürelerini ayrı olarak göstermeme karşın, yapılan çalışma göstermektedir ki önceden getirme işlemi 191.fma3d testinde performansı %15,18 arttırırken, sızıntı kontrol mekanizması performansı %33,40 oranında geriletmektedir. Genel olarak, önceden getirme işlemi ve sızıntı kontrol mekanizması beraber kullanıldığında testler için %18,22 oranında performans azalması söz konusudur. Bu sonucun anlamı, önceden getirme işlemi sızıntı optimizasyonları ile birlikte kullanıldığında yararlı olmayacaktır ve bu yüzden önceden getirme işlemi sızıntı kontrol mekanizması ile birlikte ya kontrollü ya da koordineli bir şekilde kullanılmalıdır.



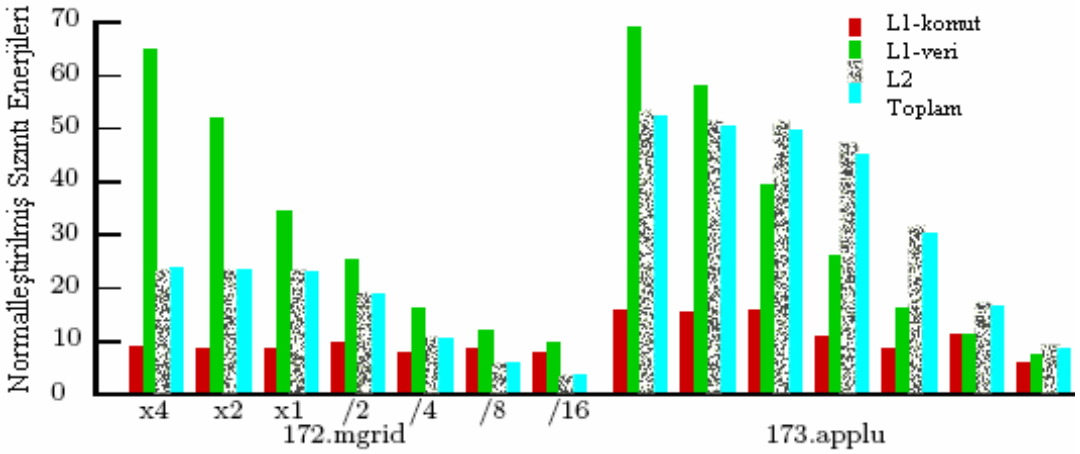
Şekil 5. 6 Optimize edilmiş kodlar için çalışma süreleri

5.4 Duyarlılık Analizleri

Şimdiye kadar, DY sızıntı modunda, bir bloğu L1 komut/veri önbelleğine yerleştirmek için 10K'lık çevrim eşik değerleri ve L2 önbelleğine yerleştirmek için 1M'lık çevrim değerleri kullanılmıştır. Bu tez çalışmasında, üzerinde çalışılan optimizasyonların performanslarını ölçmek için, optimizasyonların farklı eşik değerlerine olan duyarlıkları ölçülmüştür.

Şekil 5.7 farklı eşik değerleri için L1-L2 önbellek hiyerarşisindeki farklı önbellek bileşenlerinin sızıntı enerji değerlerini göstermektedir. Bütün sızıntı değerleri, orjinal koda karşılık gelen sızıntı enerjilerine göre normalleştirilmiştir. Kalan testlerin sonuçları benzer eğilimler gösterdiğinden, burada sadece iki testin sonucu verilmektedir. Her test; her biri, bir çift eşik değerine karşılık gelen yedi grup çubuğa sahiptir. Üçüncü grup, L1 önbelleği için 10K'lık çevrim ve L2 için 1M'lık çevrim eşik değerlerini kullanan temel konfigürasyonumuzun enerji değerlerine karşılık gelmektedir. Birinci grup, temel konfigürasyonun eşik değerlerinin dört katına sahip iken (L1 önbelleği için 40K, L2 önbelleği için 4M çevrim), dördüncü grup temel konfigürasyonun eşik değerlerinin yarısına sahiptir (L1 önbelleği için 5K çevrim, L2 önbelleği için 512K çevrim). Ayrıca, her bir grup soldan sağa doğru L1 komut önbelleği, L1 veri önbelleği, L2 önbelleği ve tümü için sızıntı enerji

değerlerini gösteren dört çubuğa sahiptir. Yapılan bu çalışmada, sızıntı faktörü k=1 olarak kullanılmıştır.

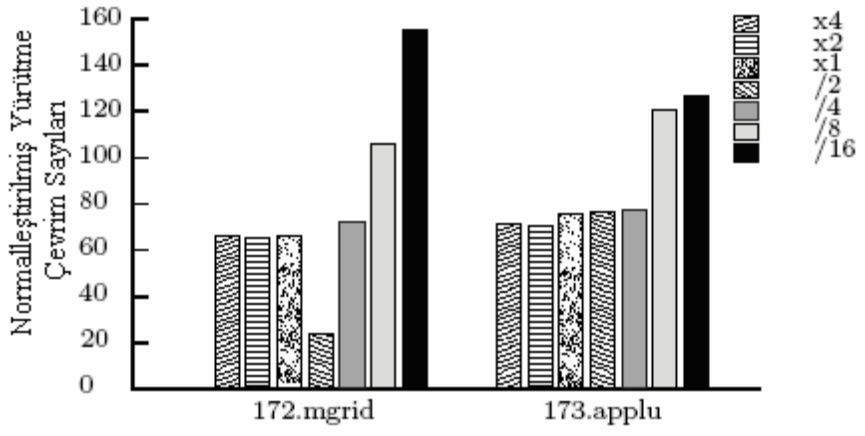


Şekil 5. 7 Farklı eşik değerleri için normalleştirilmiş sızıntı enerjileri (k=1)

Şekil 5.7'den görüldüğü gibi, eşik değerleri düştükçe, erişilmeyen önbellek bloklarının sızıntı enerjisini azaltmak için erkenden bu blokları DY moda geçirerek, L1 veri önbelleği ve L2 önbelleği için olan sızıntı enerjilerindeki enerji tasarrufu arttırılmaktadır. Bu yaklaşım, genel olarak L1 komut önbelleği için şu sebeplerden dolayı geçerli değildir. Çok iyi komut lokalitesinden dolayı, L1 komut önbellek bloklarının sadece küçük bir bölümü AK modundadır ve L1 veri önbellek bloklarıyla karşılaştırıldığında, L1 komut önbellek blokları DY sızıntı modundadır. Sonuç olarak, bloklar düşük eşik değerlerinden fayda sağlayamaz. Tersine, düşük eşik değerleri çalışma süresini arttıracağından, L1 komut blokları AK modunda daha çok çevrim harcayabilirler. Bu da sızıntı enerji tüketiminde artışa neden olur. Örneğin, 172.mgrid için, ortalama olarak, performans artışı yaklaşık %28'dir ve temel konfigürasyonun eşik değerlerini dörtte bir (/4) oranında azalttığımızda, AK modundaki L1 komut önbellek blokları çalışma süresinin %10'u oranında kalmaktadır. Diğer yandan eşik değerini sekizde bir oranında (/8) azalttığımızda, performans %6 oranında geriler ve L1 komut önbellek blokları, AK modunda yürütme zamanının %8'inde kalır. Sonuç olarak, L1 komut önbellek için sızıntı enerji tasarrufu /4'de %92 ve /8'de %90 civarındadır. Üstelik L2 önbellek sızıntı

enerjisi L1 veri/komut önbellek sızıntı enerjisini bastırdığından, toplam sızıntı enerjisi birikimlerindeki eğilim, farklı eşik değerleri için L2 önbellek sızıntı enerjisi birikimlerindeki eğilime benzerdir.

Blokları ve alt blokları DY moda erken yerleştirmek sızıntı enerjisi tasarrufu anlamında faydalıdır. Bu işlem, blokların ve alt blokların DY modunda erişilme şansını arttıracağı için performans azalmasına neden olabilir ve komut/verilerin önbellek hiyerarşisinde bir sonraki seviyeye getirilmesi için fazladan çevrim oluşmasına sebep olur. Farklı eşik değerleri ile normalleştirilmiş çalışma süreleri Şekil 5.8'de iki farklı test için verilmiştir. Testteki her bir çubuk farklı bir çift eşik değerine karşılık gelmektedir. Şekil 5.7 ve Şekil 5.8'de görülebileceği gibi, temel durumun (x1) eşik değerlerinin ötesinde eşik değerlerini arttırdığımızda, çok az bir performans artışı ve genel olarak önbellekler için sızıntı enerji birikiminde ufak bir azalma meydana gelmektedir. Fakat bu durum L1 veri önbelleği için söz konusu değildir. Bu önbellek için olan sızıntı enerjisi birikimleri x4 için %31 ve x1 için %60 civarındadır. Unutulmamalıdır ki, eşik değerlerini azalttığımızda toplam sızıntı enerjisi artarken, L1 veri önbelleğindeki artan önbellek ıskaları yüzünden performanstaki ters etkileri nedeniyle düşük eşik değerlerini rasgele kullanamayız. 172.mgrid testi için, çalışma süresi /8 için %6 ve /16 için %55 oranında artmaktadır.



Şekil 5. 8 Farklı eşik değerleri için çalışma süreleri

5.5 Önceden Getirilen Bloklar İçin Önbellek Bloklarının Enerjisini Özelleştirmek

Yapılan çalışmanın bu kısmına kadar, bütün önbellek blokları aynı şekilde işlem gördü. Özellikle, önceden getirilen önbellek bloğu bozulma aralığı süresince erişilmezse DY güç moduna yerleştirilmektedir. Bu bölümde, önceden getirilmiş önbellek bloklar için kullanılan üç farklı optimizasyon stratejisi anlatılacaktır. Birinci strateji, önceden getirme işlemi gerçekleştikten sonra önceden getirilmiş önbellek blokları hemen DK moduna yerleştirir. Yapılan bu tez çalışmasının geri kalan kısmında bu yaklaşım S-DK (Spekülatif-Durum Koruyucu) yaklaşımı olarak adlandırılacaktır.

İkinci strateji, önceden getirilmiş önbellek bloklarını DY güç moduna yerleştiren yeni bir bozulma periyodu kullanmaktadır. Bu yaklaşım, U-DY (Uyuşuk-Durum Yok edici) olarak isimlendirilmektedir. Kısaca açıklarsak, U-DY modundaki önceden getirilmiş önbellek blokları için bozulma periyodu normal önbellek bloklarına göre çok daha kısadır. Son strateji, önceki önceden getirme işleminin karakteristiklerine dayanan davranıştır.

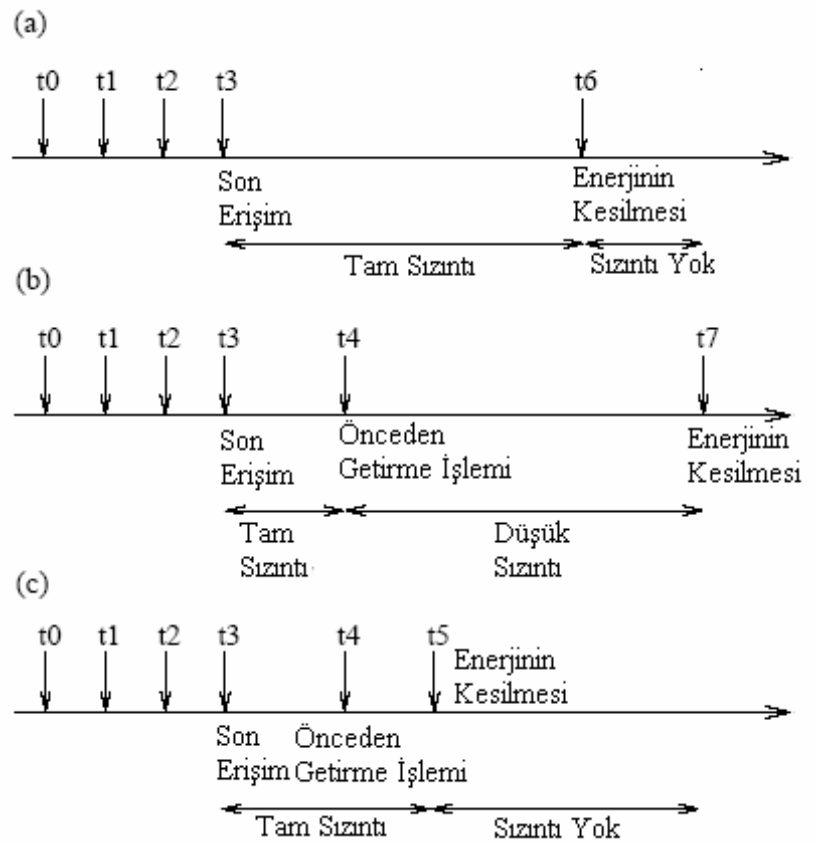
Üçüncü stratejiyi K-M (Kestirimci-Melez) olarak isimlendirebiliriz. Söz konusu olan önceden getirilmiş bloğun yararlı olup olmadığını belirlemeye çalışan kestirimci, sezgisel bir yaklaşımdır. Önceden getirme işleminden sonra bloğun güç durumu AK veya DK olabildiği için önceden getirilmiş blokların güç durumuna karar vermektedir. İlginç bir nokta olarak, birçok durumda, bu yaklaşımları kullanarak sızıntı enerjisinin azaltma miktarı yararlı-olmayan önceden getirme işlemiyle oluşturulan ek sızıntı enerji miktarından fazladır. Bunun nedeni; bu yaklaşımlar sadece yararlı-olmayan önceden getirme işlemlerinden dolayı oluşan ek sızıntı enerji tüketimini azaltmayacak, ama normal önbellek bloklarını bozulma periyodu esnasında DK/DY güç moduna erkenden sokabileceklerdir, bu da fazladan sızıntı birikimlerinin oluşmasına neden olur.

5.5.1 S-DK Yaklaşımı

Bu yöntem, önceden getirme işlemi tamamlandıktan hemen sonra getirilmiş olan blokları DK sızıntı kontrol modunda yerleştirir. Bu yaklaşımda, eğer bloklar gereksiz yere önceden getirilmişse (hiçbir erişim olmadığında blok önbellekten çıkarılır), söz konusu blok başka bir blokla yer değiştirinceye kadar (veya bozulma periyodu boyunca) AK modunda tam sızıntı enerji tüketimi yapması yerine DK sızıntı kontrol moduna sokulur. Aynı zamanda, bu strateji ile bazı yararlı-olmayan önceden getirme işlemleri kötü olmayıp; tersine, sızıntı enerjisi tasarrufuna katkıda bulunabilirler.

Şekil 5.9(a) tipik bir önbellek bloğunun yaşam süresini göstermektedir. t0 anında, söz konusu bloğa yeni bir veri getirilir ve t3 zamanında veriye son erişim oluşur. Bu veriye t1 ve t2 zamanlarında sırasıyla erişilir ve t3 zamanında da son erişim yapılır. Bozulma periyodu boyunca önbellek bloğuna erişilmediği düşünülürse, t6 zamanından sonra hiç sızıntı enerjisi tüketmeyen DY sızıntı kontrol moduna geçiş yapılır.

Şekil 5.9(b) S-DK stratejisi kullanıldığında önbellek bloklarını göstermektedir. t4 zamanında önceden getirme işlemiyle verinin bloğa getirildiğini ve aynı zamanda bloğun DK moda geçirildiğini farz edelim. Sonuç olarak, blok t3-t6 aralığı yerine t3,t4 periyodu boyunca sadece tam sızıntı gücü tüketecek ve t4-t7 periyodu boyunca düşük sızıntı güç harcanacak, bu da sızıntı enerji tasarrufu yapılmasını sağlayacaktır. Eğer, blok içerisindeki orjinal veri program tarafından sonradan referans edilmemişse, bu önceden getirme işlemi, tek başına kullanışlı olmamasına rağmen güç tasarrufuna dönüştürülebilir.



Şekil 5. 9 (a) Önceden getirme işlemi yapılmıyor. t3-t6 periyodunda tam sızıntı güç tüketiliyor. (b) S-DK yaklaşımı, t4 zamanında önceden getirme işlemi yapılıyor ve t3-t4 periyodunda tam sızıntı güç tüketiliyor. (c) U-DY yaklaşımı, önceden getirilmiş blok t5 zamanında DY sızıntı kontrol moduna geçirilir, t3-t5 periyodunda tam sızıntı güç tüketilir. t5 zamanından sonra sızıntı güç tüketimi yoktur.

Bu yöntemin dezavantajları, sadece DK sızıntı kontrol modunda, önceden getirilmiş yararlı-olmayan blokları değil, aynı zamanda önceden getirilmiş yararlı önbellek bloklarıda DK moduna konur. Bu yüzden, önceden getirilmiş yararlı bloklara erişildiğinde, fazladan bir makine çevrimine ihtiyaç duyulur. Bu yöntemin neden olduğu performans gerilemesi daha sonra ölçülecektir.

5.5.2 U-DY Yaklaşımı

Bu yaklaşımda, önbelleğe önceden getirme yöntemi ile getirilen bloklar için yeni bir bozulma periyodu tanımlanmaktadır. Lokalitenin ilkelerinden dolayı, önceden getirilmiş yararlı blokların önbelleğe getirildikten sonra kısa bir zaman içerisinde erişilebileceği farz edilecektir. Eğer önceden getirilmiş bloklar bu küçük

bozulma süresinde hiçbir şekilde erişilmezse, DY sızıntı kontrol moduna yerleştirilir. Bu yöntemi daha iyi açıklamak için Şekil 5.9(c)'yi göz önüne alalım. Blok önbelleğe t4 zamanında önceden getirme işlemi ile getirilmektedir. Eğer bu blok t4-t5 süresince erişilmeyecekse (t5-t4 yeni önbellek bozulma süresine eşittir), t5 zamanında DY sızıntı kontrol moduna yerleştirilir. Sonuç olarak, önceden getirilmiş yararlı-olmayan bloklar, t4-t7 zaman aralığı yerine t4-t5 zaman aralığında tam sızıntı güç tüketecektir (t7-t4 normal önbellek bloklarının bozulma periyoduna eşittir). Bu, önceden getirilmiş yararlı-olmayan bloklar için fazlalık güçlerin azalmasına sebep olur. S-DK yaklaşımında olduğu gibi, AK sızıntı modundaki önbellek bloklarında yararlı-olmayan önceden getirme işlemleri sızıntı enerji tasarrufuna katkıda bulunacaktır. Bunun nedeni, yukarıda S-DK yaklaşımında açıklandığı gibi benzer nedenden dolayıdır.

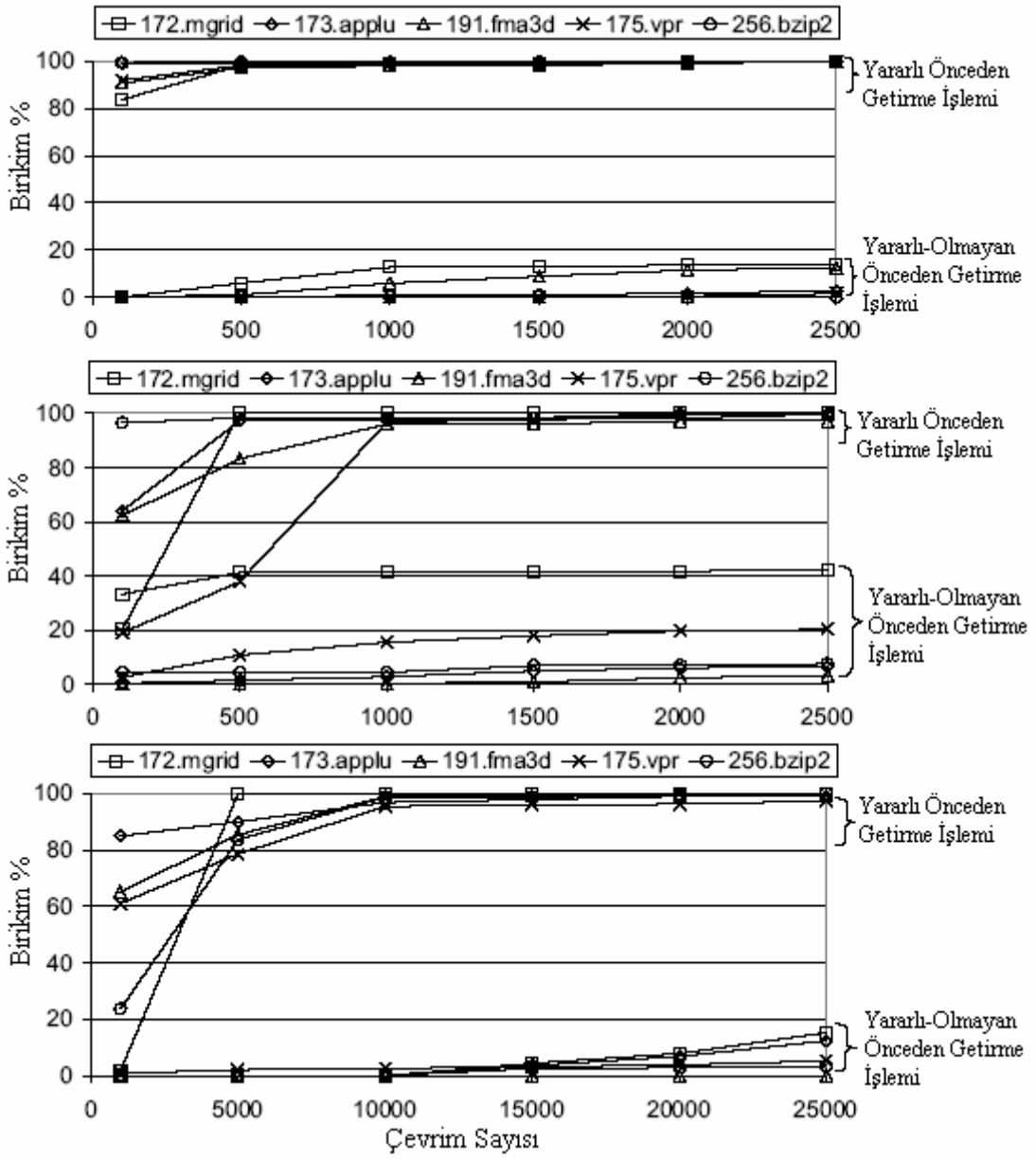
5.5.2.1 Önceden Getirilmiş Bloklar için Bozulma Zamanının Belirlenmesi

U-DY yaklaşımında, önceden getirilmiş önbellek blokları için uygun bozulma süresini seçmek çok önemlidir. Eğer bozulma periyodunu çok kısa seçersek, önceden getirilmiş yararlı bloklar erişilmeden önce erkenden kapanır. Bu da, performans azalması ve enerji kaybı ile sonuçlanır. Diğer yandan, bozulma periyodu çok uzun seçilirse, önceden getirilmiş yararlı-olmayan blok, gereksiz olarak uzun bir süre için AK sızıntı kontrol modunda olacaktır. Bu da, sızıntı enerji ek yükünü arttıracaktır.

Önceden getirilmiş L1 komut, L1 veri ve L2 blokları için erişim aralıklarının birikimli dağılımı Şekil 5.10'da gösterilmektedir. Üç grafiğin tümünde, alttaki çizgiler önceden getirilmiş yararlı-olmayan bloklar için erişim aralıklarını ifade ederken, üstteki çizgiler önceden getirilmiş yararlı bloklar için erişim aralıklarını ifade etmektedir. Bu şekilden, şu gözlemleri yapmamız mümkündür. İlk olarak, genelde, L1-L2 hiyerarşisindeki önceden getirilmiş yararlı bloklar, getirildikten sonra çok kısa bir süre içerisinde erişilebilmektedir. Örneğin, ortalama olarak, önceden getirilmiş yararlı L1 komut (L1 veri) önbelleği bloklarının %98,84 (%97,65)'ü, önceden getirme işlemi ile getirildikten sonra 1K'lık çevrim aralığında erişilebilmektedir. Buna ek olarak, L2 önbelleğine getirildikten sonra 10K'lık çevrim

aralığı içinde önceden getirilmiş yararlı blokların %97,95'ine erişilmektedir. Bu sonuçlar, uzaysal lokalitenin ilkeleriyle uyumludur. İkinci olarak, önceden getirilmiş yararlı-olmayan bloklar önbellekten çıkarılmadan önce dikkate değer bir zaman harcamaktadır. Örneğin, ortalama olarak sadece önceden getirilmiş yararlı-olmayan L1 komut önbelleği bloklarının %4,79'u ve önceden getirilmiş yararlı-olmayan L1 veri önbelleği bloklarının %12,76'sı önbelleğe getirildikten sonra 1K'lık çevrim aralığı içerisinde yer değiştirmektedir. Diğer yandan, L2 önbelleği için, önceden getirilmiş yararlı-olmayan önbellek bloklarının %2,62'si, getirildikten sonra 175.vpr testi içinde 10K'lık çevrim aralığında yer değiştirmektedir. Testlerin geri kalanı için, bu süre içinde önceden getirilmiş yararlı-olmayan L2 önbellek bloklarının hiçbiri önbellek içinden çıkarılmamıştır.

Bu iki gözleme dayanarak, önceden getirilmiş yararlı-olmayan bloklardan kaynaklanan sızıntı enerji ek yükünü indirgemek için performanstaki azalma göz ardı edilip, AK sızıntı kontrol modunda daha az zaman harcamasına izin verilerek önceden getirilmiş blokların bozulma aralığı azaltılır. Analizlerin geri kalanında, önceden getirilmiş blokların bozulma süresi L1 komut ve veri önbellekleri için 1K'lık çevrim, L2 önbelleği için 10K'lık çevrim seçilmiştir. Diğer bir deyişle, önceden getirilmiş yararlı-olmayan L1 komut (veri) / L2 önbellek blokları 1K/10K'lık çalışma süreleri sonrasında kapatılır. Sonuç olarak, önceden getirilmiş yararlı-olmayan L1 komut (veri) blokları için 10K'lık çalışma süresi ve L2 önbelleği bloklarının kapatılması için 1M'lık çalışma süresi beklenmek zorunda değildir. Bu da bize açıkça toplam sızıntı enerjisi tüketiminin azaltılmasında yardımcı olacaktır.



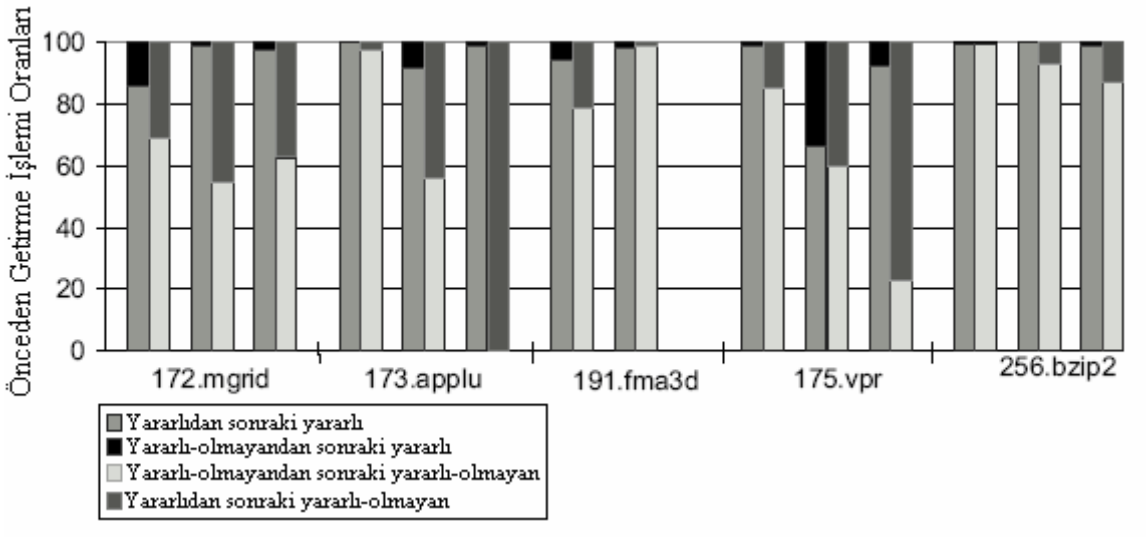
Şekil 5. 10 Önceden getirilmiş önbellek bloklarının erişim aralıkları dağılımı. En üstteki ve ortadaki şekil L1 komut ve veri önbelleği için, en alttaki şekil ise L2 önbelleği için. Her bir şekildeki üstteki çizgiler önceden getirilmiş yararlı blokların erişim aralıklarını, alttaki çizgiler ise önceden getirilmiş yararlı-olmayan blokların erişim aralıklarını göstermektedir.

5.5.3 K-M Yaklaşımı

Bu yaklaşımın detaylarına girmeden önce, bu yöntemin ardındaki fikri sağlamlaştırmak için elde ettiğimiz sonuçlara bakalım. İlk olarak, önceden getirme işlemini yararlı ve yararlı-olmayan önceden getirme işlemi olarak kategorize ettik. Eğer aynı blok içerisine getirilen yararlı bir önceden getirme işleminden sonraki

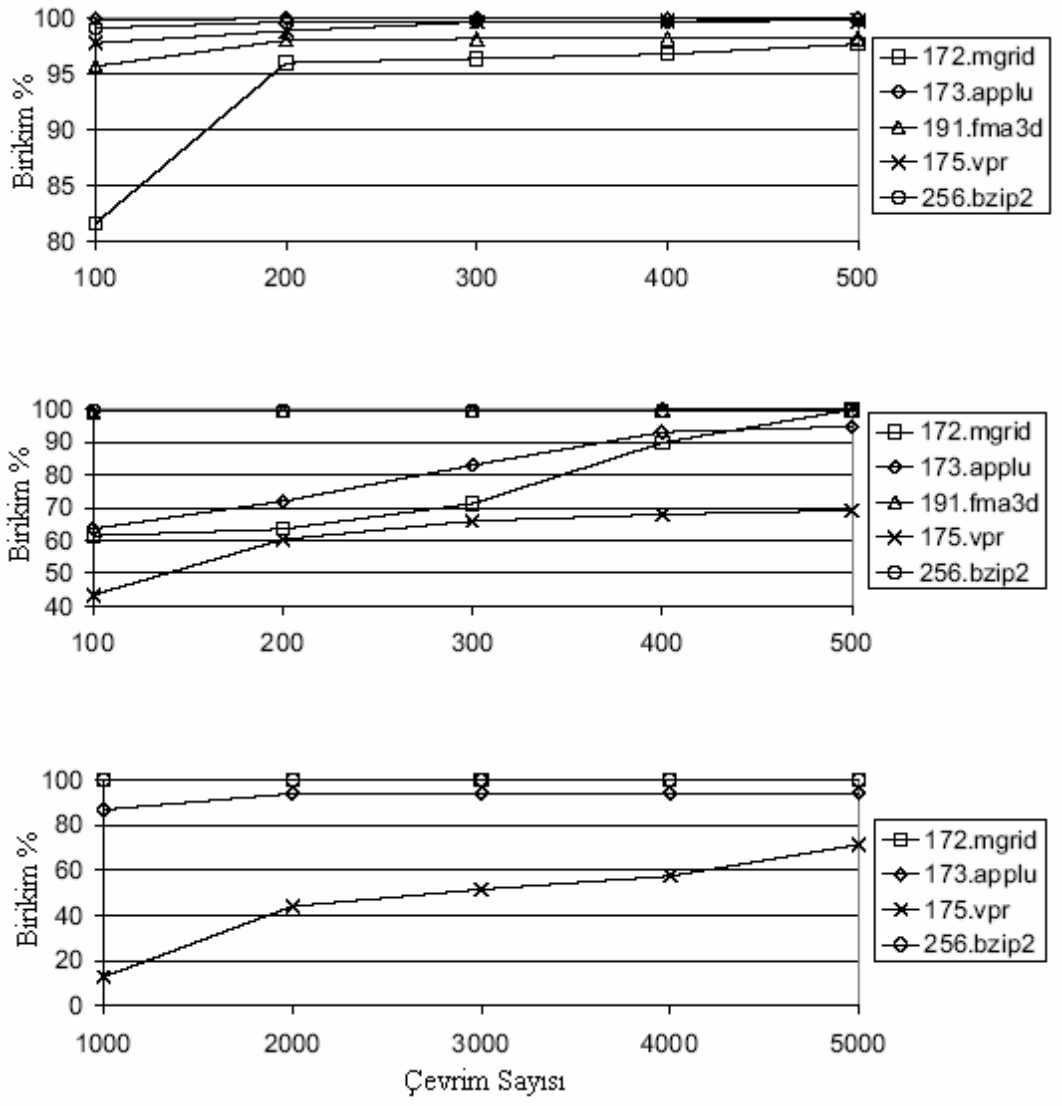
getirme işlemi yararlı ise, buna yararlıdan sonraki yararlı (useful after useful) önceden getirme işlemi olarak adlandırılır. Eğer yararlı-olmayan önceden getirme işleminden sonraki getirme işlemi yararlı ise, yararsızdan sonraki yararlı önceden getirme işlemi (useful after non-useful prefetch) ismi verilir. Benzer olarak, yararlı-olmayan önceden getirme işleminden sonraki bir yararlı-olmayan önceden getirme işlemi ise yararsızdan sonraki yararsız önceden getirme işlemi (non-useful after non-useful prefetch) denilir.

Şekil 5.11'de 4 kategori içerisinde yararlı ve yararlı-olmayan önceden getirme işlemleri gösterilmektedir. Bu şekildeki her bir test için üç grup çubuk bulunmaktadır; birinci ve ikinci grup çubuklar L1 komut ve veri önbelleklerini, üçüncü grup çubuklar ise L2 önbelleğini temsil etmektedir. Şekilden kolayca görülebildiği gibi, bütün önbellekler için eğer önceden getirme işlemi yararlı ise, aynı bloğa bir sonraki önceden getirme işlemi de muhtemelen yararlı olacaktır. Ayrıca, L2 önbelleği için 173.applu ve 175.vpr testleri hariç, eğer veri bloğa önceden getirme yöntemi ile getirilmişse ve erişilmeden önce yer değiştirmişse, aynı bloğa önceden getirme yöntemi ile getirilen veri erişilmeden önce muhtemelen çıkarılacaktır. İkinci sonuç olarak, aynı bloğa getirilen bir önceki verinin sadece erişim süresine (Access interval) bakılarak önceden getirilen veriye ne zaman erişildiğinin zamanı tahmin edilmeye çalışılmıştır. Önceden getirme işlemi ile verinin önbelleğe getirildiği zaman ve erişilme zamanı arasındaki farka erişim süresi denilmektedir.



Şekil 5. 11 Yararlıdan sonraki yararlı ve yararlı-olmayandan sonraki yararlı durumlarında yapılan yararlı önceden getirme işlemlerinin oranı. Yararlı-olmayandan sonraki yararlı-olmayan ve yararlıdan sonraki yararlı-olmayan durumlarında yapılan yararlı-olmayan önceden getirme işlemlerinin oranı

Şekil 5.12'de aynı blok içerisine yapılan ardışık iki yararlı önceden getirme işlemi için erişim süreleri arasındaki farlılıkların dağılımı gösterilmektedir. Bu şekilden çıkarılacak ilk gözlem, tasarlanan mimarideki üç önbellek için, önceden getirilmiş bir bloğun erişim süresi belli bir değer içerisinde (aynı blok içerisine bir önceki önceden getirme işleminin erişim süresine dayanarak) tahmin edilebilmektedir. Örneğin, 173.applu testi için önceden getirme işlemiyle L1 veri önbelleğine getirilen verilerin %71,94'üne, aynı blok içine bir önceki getirme işleminin erişim süresinden 200 çevrim süresi kadar sapılarak erişilmiştir (önceden getirme işleminden sonra).



Şekil 5. 12 Aynı blok içerisine yapılan ardışık iki önceden getirme işleminin erişim aralıkları arasındaki farklılıkların dağılımı. En üstteki ve ortadaki şekil L1 komut ve veri önbellekleri için, en alttaki şekil ise L2 önbelleği için.

Daha önceden de bahsedildiği gibi, K-M yaklaşımı hem yararlı/yararlı-olmayan önceden getirme işleminin sonucunu ve hem de aynı blok içerisine bir önceki önceden getirme işlemine dayanarak önceden getirilen veriye ne zaman erişilebileceğini tahmin etmektedir. Aşağıda bu yaklaşımın yalancı-kod (pseudo-code) formatındaki algoritması verilmektedir. Her önceden getirilmiş blok için, K-M yöntemi aynı bloğa yapılan bir önceki önceden getirme işleminin yararlı olup olmadığını kontrol eder. Eğer bir önceki önceden getirme işlemi yararlı-olmayan önceden getirme işlemi ise, blok DK sızıntı kontrol moduna geçirilir. Aksi takdirde,

önceden getirme işlemi yararlı önceden getirme işlemi olarak farz edilir ve bloğun sızıntı modu aynı blok içerisine yapılan bir önceki yararlı önceden getirme işleminin davranışı göz önüne alınarak belirlenir. Eğer bir önceki önceden getirme işleminin erişim süresi eşik değerinden küçük ise, önceden getirilen veriye önceden getirme işlemi tamamlandıktan sonra çok kısa bir süre içerisinde erişileceği varsayılır. İhtiyaç olunduğunda hazır olması için AK sızıntı moduna geçirilir (eşik değeri olarak L1 komut ve veri önbellekleri için 200 makine çevrimi, L2 önbelleği için 2000 makine çevrimi kullanılmıştır). Eğer bu durum gerçekleşmezse, önceden getirilmiş blok bir periyot (aynı blok içerisine daha önce getirilmiş olan verinin erişim süresinden daha küçük) için DK sızıntı kontrol moduna geçirilir.

If aynı blok içerisine yapılan bir önceki önceden getirme işlemi yararlı değilse **then**
    Bloğu DK moda geçir
Else
    Eğer aynı bloğa yapılan bir önceki önceden getirme işleminin erişim aralığı<eşik değeri **then**
        Bloğu AK moda geçir
    Else
        bloğu DK moda geçir
        bir önceki önceden getirilmiş verinin erişim aralığı-eşik değeri

        Eğer blok hala DK modunda mı? **Then**
            Bloğu AK moda geçir
        End if
    End if
End if

Eğer önceden getirilmiş bloğun bozulma süresi esnasında bloğa erişilmiyorsa **then**
    Bloğu DY moda geçir
End if

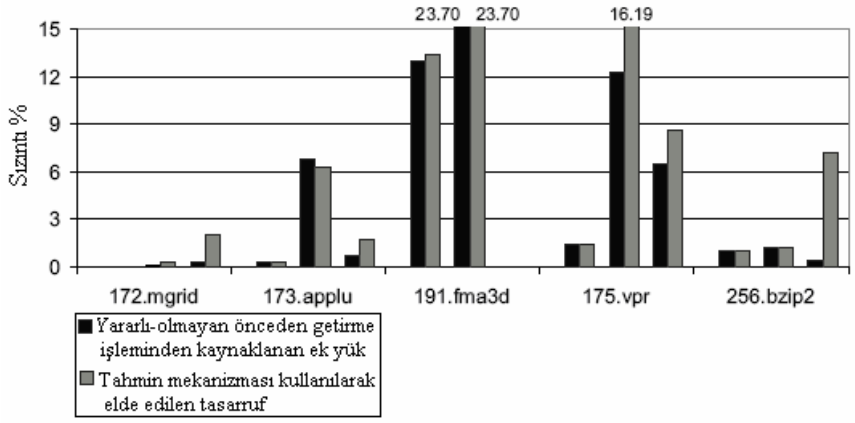
Eşik değeri parametresini ayarlamak hem gücü hem de performansı etkilediği için çok kritik öneme sahiptir. Eğer eşik değeri çok küçük olursa, önceden getirilmiş bloklara ihtiyaç olunduğunda erken bir sürede enerji verilemez; bu da performans azalmasıyla sonuçlanır. Benzer olarak, eğer eşik değeri çok büyük olursa, bloklara bu sefer çok erken enerji verilir; bu da sızıntı enerji tasarrufunda azalmaya neden olur. Erişim süresinden sonra, eğer blok hala DK modunda ise, bloğa erişim süresi içerisinde erişilemez; eğer AK sızıntı modunda ise, ihtiyaç olunduğunda hazır olacaktır. Eğer önceden getirilmiş blok, getirildikten sonra (bir önceki önceden getirme işleminin sonucu ne olursa olsun) bozulma periyodu süresince erişilemezse, sızıntı enerji tasarrufu yapmak için U-DY modunda olduğu gibi DY moduna geçirilir.

Yapılan testlerde, önceden getirilen bloklar için bozulma periyodu olarak U-DY yönteminde kullanılan değerler kabul edildi (L1 önbelleği için 1K çevrim ve L2 önbelleği için 10K çevrim). Bu noktada, Şekil 5.11 temel alınarak şu önerilebilir, yararlı-olmayan önceden getirme işlemlerinin çoğu, sadece aynı blok içerisine yapılan bir önceki önceden getirme işlemi yararlı olduğunda önceden getirme işlemine izin verilerek elenebilir.

5.5.4 Sızıntı Enerji Tasarrufları

Bu bölümde, yararlı-olmayan önceden getirme işlemlerinden kaynaklanan sızıntı enerji ek yüküne ilişkin sonuçlar, bir tahmin mekanizması (oracle predictor) kullanıldığında maksimum sızıntı enerji tasarrufu ve yukarıda anlatılan U-DY, S-DK ve K-M yaklaşımları kullanıldığında elde edilen sızıntı enerji tasarruflarının sonuçları verilecektir. Maksimum sızıntı tasarrufu için, bir önceden getirme işleminin yararlı olup olmadığını doğru tahmin edebilen bir tahmin mekanizmasına sahip olduğumuz varsayılacaktır. Önceden getirme işlemi yararlı-olmayan türden olduğunda, ilgili bloğun enerjisi önceden getirme işlemi tamamladıktan hemen sonra enerji tasarrufu yapmak için kesilir.

Yararlı-olmayan önceden getirme işlemlerinden kaynaklanan sızıntı enerji ek yükünün ve optimize edilmiş kodlar için L1-L2 önbelleklerindeki tahmin mekanizmasıyla elde edilen sızıntı enerji tasarruflarının oranı Şekil 5.13'te gösterilmektedir. Her bir test için birinci, ikinci ve üçüncü grup çubuklar sırasıyla L1 komut, L1 veri ve L2 önbellekleriyle ilişkilidir. Her grup içerisindeki ilk çubuk yararlı-olmayan önceden getirme işleminden kaynaklanan ek yükleri ve ikinci çubuk ise tahmin mekanizması kullanıldığında elde edilen sızıntı enerji tasarrufunu göstermektedir. Bu şekilde, genel olarak herhangi bir testte bazı bileşenler için sızıntı enerji tasarrufunun, yararlı-olmayan önceden getirme işleminden kaynaklanan ek yükten daha fazla olduğu görülmektedir.

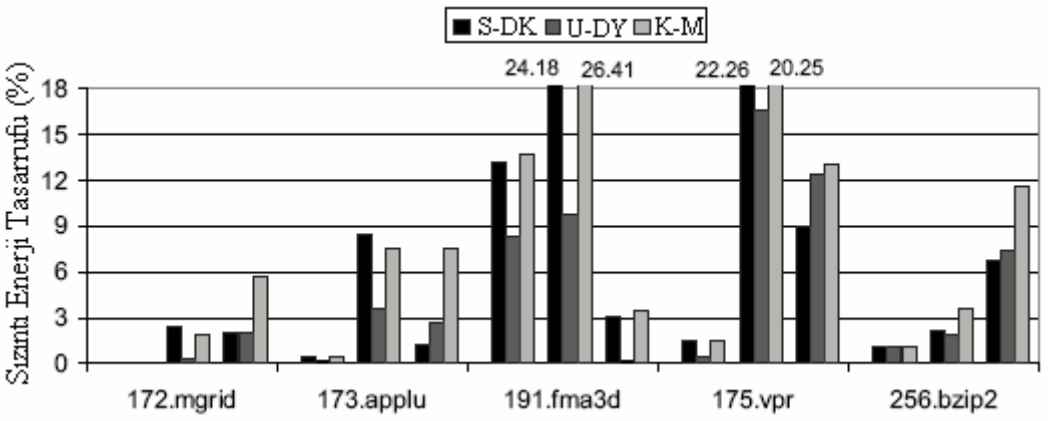


Şekil 5. 13 Yararlı-olmayan önceden getirme işleminden kaynaklanan sızıntı enerji ek yükleri ve tahmin mekanizması kullanılarak elde edilen sızıntı enerji tasarrufları

Ayrıca, 191.fma3d testi için hem sızıntı ek yükü hem de tasarrufu L1 veri önbelleğinde %23,70 oranındadır. Diğer taraftan aynı testte, L2 önbelleğindeki bütün önceden getirme işlemleri yararlı olduğunda bu önbellek herhangi bir ek yüke maruz kalmaz.

Şekil 5.14'te U-DY, S-DK ve K-M yaklaşımları uygulandığında L1-L2 önbelleklerindeki sızıntı enerji tasarrufları gösterilmektedir. Önceden getirme işlemi ve sızıntı kontrol mekanizmalarıyla optimize edilmiş kodların sızıntı enerji tüketimlerine göre tasarruf oranları verilmektedir. Her bir test için, üç grup çubuk bulunmaktadır; birinci ve ikinci gruplar L1 komut ve L1 veri önbelleklerindeki,

üçüncü grup ise L2 önbelleğindeki sızıntı tasarrufları içindir. Her grup içerisinde, ilk çubuk S-DK yaklaşımı kullanıldığında, ikinci ve üçüncü çubuklar ise U-DY ve K-M yaklaşımları kullanıldığında elde edilen sızıntı enerji tasarruflarını göstermektedir.



Şekil 5. 14 S-DK, U-DY ve K-M yaklaşımları kullanıldığında elde edilen sızıntı enerji tasarrufları.

Bu şekilden şu gözlemler çıkarılabilmektedir. İlk olarak, S-DK yaklaşımı L1 önbelleği sızıntı tasarruflarına göre U-DY yaklaşımından daha iyi sonuçlar vermektedir. S-DK yaklaşımı, önceden getirilmiş blokları getirme işlemi tamamlandıktan hemen sonra DK sızıntı kontrol moduna geçirmesinden ve L1 önbellek bloklarının enerjisinin kesilmeden önce en fazla 10K çevrim harcamasından dolayı daha başarılı bir yaklaşımdır. Diğer taraftan, U-DY yaklaşımı önceden getirilmiş blokları L1 önbelleğinde DY moduna geçirmek için 1K çevrim süresi beklemektedir; bu da daha fazla sızıntı enerji tüketimiyle sonuçlanır.

İkinci olarak, L1 önbelleklerindeki durumun tam tersi olarak, U-DY yaklaşımı L2 önbelleğindeki sızıntı enerji tasarrufunda S-DK yaklaşımından daha iyi iş çıkarmaktadır. Bu durum, U-DY yaklaşımının önceden getirme işlemi tamamlandıktan sonra oldukça kısa bir sürede (10K çevrim) önceden getirilmiş blokları DY sızıntı kontrol moduna geçirmesinden kaynaklanmaktadır. S-DK yaklaşımı önceden getirilmiş blokları eğer erişilmemişse, nispeten daha uzun bir süre (1M çevrim) DK sızıntı kontrol modunda tutmaktadır. Bu durum, S-DK yaklaşımının U-DY yaklaşımından L2 önbelleğindeki sızıntı enerji tasarruflarına göre daha az etkili yapmaktadır. Şekilde de görüldüğü gibi, bütün testler için (sadece 191.fma3d testi hariç) U-DY yaklaşımının L2 önbelleğindeki enerji tasarrufuna ilişkin davranışı

S-DK yaklaşımınkinden daha iyidir. Bu testte gözlemlenen farklı davranış L2 önbelleğinde hiç yararlı-olmayan önceden getirme işleminin olmamasından dolayı U-DY yaklaşımının avantajını kullanamamasından kaynaklanmaktadır.

Bu şekil üzerindeki bir diğer gözlem ise, K-M yaklaşımının bütün önbellek mimarileri için sızıntı enerji tasarrufunda U-DY yaklaşımından daha iyi performans sergilemesidir. Ayrıca, K-M yaklaşımı L2 önbelleğinde sızıntı enerji tasarrufu yapmada S-DK yaklaşımından daha iyi olduğundan, L1 önbellekleri söz konusu olduğunda davranışları kıyaslanabilmektedir.

Dördüncü olarak, U-DY yaklaşımının performansı yararlı-olmayan önceden getirme işlemlerinin sayısına bağlıdır. Eğer yararlı-olmayan önceden getirme işlemlerinin sayısı L2 önbelleğindeki 191.fma3d testinde olduğu gibi düşükse, performans çok düşük olacaktır. Ancak, önceden getirilmiş yararlı bloklar DK sızıntı güç modunda tutulduğundan K-M yaklaşımı için bu durum gerçekleşmez. 191.fma3d testinde, K-M yaklaşımının performansı S-DK yaklaşımından biraz daha iyi olduğundan dolayı U-DY yaklaşımında hiç sızıntı enerji tasarrufu bulunmamaktadır.

Beşinci olarak, Şekil 5.13'te verilen sızıntı enerji ek yükleri/tasarrufları ve bu üç yaklaşım yoluyla elde edilen sızıntı enerji tasarrufları karşılaştırıldığında; açıkça görülmektedir ki, genellikle bu stratejiler önceden getirilmiş yararlı-olmayan bloklar nedeniyle karşılaşılan sızıntı enerji ek yükleri ve tahmin mekanizması sayesinde elde edilen sızıntı enerji tasarruflarından daha fazla sızıntı enerji tasarrufu yapmaktadır. Örneğin 173.applu testinde L1 veri önbelleğine önceden getirilmiş yararlı-olmayan bloklar S-DK yaklaşımında sızıntı enerjinin %6,77 artmasına neden olduğundan, L1 veri önbelleğindeki sızıntı enerji tasarrufu %8,43'tür. Bir diğer örnek 175.vpr testinde önceden getirilmiş yararlı-olmayan bloklar nedeniyle L2 önbelleğinde sızıntı enerji ek yükü %6,50 ve U-DY yaklaşımı kullanıldığında sızıntı enerji tasarrufu %12,43'tür. Bunun anlamı, yararlı-olmayan önceden getirme işlemlerinden dolayı oluşan sızıntı enerji ek yüklerini telafi etmek için, önceden getirilmiş yararlı-olmayan blokların aslında diğer sızıntı enerji tasarrufları için kullanılabileceğidir.

5.5.5 Performans Ek Yükü

U-DY yaklaşımı uygun zaman dilimini bekledikten sonra düşük güç modunda önceden getirilmiş blokları yerleştirdiğinden, performans düşüşü ihmal edilebilir. Aslında, ortalama performans düşüşü U-DY yaklaşımında %0,3'ten daha azdır. Bu, uygun bir şekilde seçilen önceden getirilmiş blokların yeni bozulma periyodunun (L1 veri/komut önbelleği için 1K çevrim, L2 önbelleği için 10K çevrim) göz önüne alınmasıyla açıklanabilir; böylece bütün önceden getirilmiş yararlı bloklar uygun bozulma periyodunda erişilebilmektedir. Örneğin, önceden getirilmiş yararlı verilerin %97,65'ine uygun bozulma periyodunda erişilebilmektedir ve hiçbir önceden getirilmiş yararlı blok erişilmeden önce DY sızıntı güç moduna geçirilmemektedir. Aynı argüman hem L1 veri önbelleğinde hem de L2 önbelleğine önceden getirilmiş yararlı blokları seçmeye çalışmaktadır.

S-DK yaklaşımı sadece önceden getirilmiş yararlı-olmayan blokları değil aynı zamanda önceden getirilmiş yararlı blokları da, önceden getirme işleminden sonra DK moduna geçirir. Önceden getirilen bloğa erişildiğinde, bu blok veri işlemciye gönderilmeden önce AK güç moduna geçirilmesi gerekmektedir. Yapılan testlerde, bu geçişin 1 çevrim süresi aldığı farz edilmiştir. Bu ek yükün sebebi, S-DK yaklaşımının performans düşüşlerine göre U-DY yaklaşımı kadar başarılı olamamasıdır. Örneğin, S-DK yaklaşımı kullanıldığında 173.applu testi için %1,3, 191.fma3d testi için ise %1,8 performans düşüşü olmaktadır. Testlerin geri kalanında S-DK yaklaşımından kaynaklanan performans düşüşü %0,8'den daha azdır. Bu performans ek yükleri U-DY yaklaşımından kaynaklanan ek yüklerden fazla olduğunda, bu performans düşüşleri fazla değildir ve enerjide yapılan büyük tasarruflar sayesinde tolerans gösterilebilir.

K-M yaklaşımı, önbellek bloklarını ihtiyaç duyulduğunda hazır hale getirmek için erkenden DK sızıntı güç moduna geçirmeye çalışır. Bu yüzden, bu yöntemin performansı S-DK yaklaşımından daha iyidir. Diğer taraftan, önceden getirilmiş blokların erişim aralıklarının tahminindeki hatadan dolayı, önbellek bloklarının bazıları ihtiyaç duyulduğunda DK moduna sokulabilir; bu da performansta azalmaya

neden olur. K-M yaklaşımında ortalama olarak performans düşüşü %0,7'dir. Önerilen üç yaklaşımın herhangi birinde, başarılan yüksek enerji tasarruflarıyla kıyaslandığında çok yüksek performans düşüşleri olmamaktadır.

BÖLÜM 6

SONUÇ VE TARTIŞMA

Yapılan bu tez çalışmasında iki önemli konuya değinilmiştir. İlki, önceden getirme işlemi ve önbellek bloklarının enerjisinin kesilmesi arasındaki güç-performans etkileşimlerinin detaylı ölçümü sunulmuştur. İkinci olarak, bu ölçümlere dayanarak, üç farklı önbellek bloğu enerji azaltma yöntemi incelenmiştir. Bu önceden getirme işlemlerine duyarlı yaklaşımların ardındaki ana amaç, önceden getirme işleminden elde edilen muhtemel performans yararlarından vazgeçmeksizin sızıntı enerji tasarrufunu arttırmaktır.

S-DK yaklaşımı, önceden getirilmiş önbellek bloklarını önceden getirme işlemi gerçekleştirildikten hemen sonra DK sızıntı kontrol moduna geçirir. U-DY yaklaşımı, önceden getirilmiş blokları DY sızıntı moduna geçirmek için yeni bir bozulma periyodu kullanır. Daha da açmak gerekirse, eğer blok kısa bir bozulma periyodu içerisinde erişilmezse önceden getiriliş önbellek bloğu DY moda geçirilir. K-M yaklaşımı bir veriyi önceden bloğa getirmek için, aynı blok içerisine yapılan bir önceki önceden getirme işleminin karakteristiğine göre davranır. Bu yaklaşım blok içerisine yapılan önceden getirme işleminin yararlı olup olmadığını tahmin eder. Tahminine dayanarak önceden getirilmiş bloğu ya AK yada DK güç moduna geçirir. SimpleScalar simülasyonu kullanılarak SPEC2000 uygulamaları için, bu üç yaklaşımın çok küçük performans düşüşleriyle önemli sızıntı enerji tasarrufu yapabildikleri gözlenmiştir.

KAYNAKLAR

Abd-El-Barr, M., El-Rewini, H. 2005. *Fundamentals Of Computer Organization And Architecture*. A John Wiley and Sons, Inc Publication. 1-11.

Azizi, N., Najm, F.N., Moshovos, A. 2003. Low-Leakage asymmetric-cell SRAM. *IEEE Transaction On Very Large Scale Integration Systems,* vol. 11.(4).

Balch, M. 2003. *Complete Digital Design*. McGraw-Hill Publishers. 150-158.

Borka, S. 1999. Design Challenges Of Technology Scaling. *IEEE Micro*, 19(4):23-29.

Brorsson, M. (2001). Low-Energy SoC Computer Architectures http://researchprojects.kth.se/index.php/kb_1/io_8698/io.html

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

Catthoor, F., Wuytack, S., Greef, E.D., Nachtergaele, Vandecappelle, A. 1998. *Custom Memory Management Methodology Exploration Of Memory Organization For Embedded Multimedia System Design*. Kluver Academic Publishers, MA. 364 p.

Chandrakasan, A., Bowhill, W.J., Fox, F. 2001. Design Of High-Performance Microprocessor Circuits. *IEEE Press*.

Chen, T.F., Baer, J.L. 1995. Effective Hardware-based Data Prefetching For High-Performance Processors. *IEEE Transactions on Computers,* 44(5): 609-623

Cooksey, R. 2002. Content-Sensitive Data Prefetching. PhD Dissertation (Doktora Tezi). The University of Colorado, USA. s: 9-12.

Cooksey, R., Jourdan, S., Grunwald, D. 2002. A Stateless Contend-Directed Data Prefetching Mechanism. In Proc. International Conference On Architectural Support For Programming Languages And Operating Systems.

Cotterell, S., Vahid, F. 2002. Synthesis Of Customized Loop Caches For Core-based Embedded Systems. International Conference On Computer-Aided Design, pp. 655-662.

Crago, S.P. 1997. HiDISC: A High Performance Hierarchical, Decoupled Computer Architecture. PhD Dissertation (Doktora Tezi). University of Southern California, USA. s: 13-18.

Flautner, K., Kim, N.S., Martin, S., Blaauw, D., Mudge, T. 2002. Drowsy Caches: Simple Techniques For Reducing Leakage Power. The 29th International Symposium on Computer Architecture. 148-157.

Ghose, K., Kamble, M.B. 1999. Reducing Power In Superscalar Processor Caches Using Subbanking, Multiple Line Buffers, And Bit-Line Segmentation. International Symposium on Low Power Electronics and Design, 70-75.

Gomathisankaran, M., Somani, A. 2003. Efficient Energy Saving Scheme For On-Chip Caches. Technical Report, Iowa State University.

Gordon-Ross, A., Vahid, F., Dutt., N. 2004. Automatic Tuning of Two-Level Caches to Embedded Applications. Design, Automation and Test in Europe Conference and Exhibition, Volume I.

Gornish, E.H. 1995. Adaptive And Integrated Data Cache Prefetching For Shared-Memory Multiprocessors. PhD Dissertation (Doktora Tezi). University of Illinois, Urbana-Champaign, USA.

Gupta, A., Hennessy, J., Gharachorloo, K., Mowry, T., Weber, W.D. 1991. Comparative Evaluation Of Latency Reducing And Tolerating Techniques. In Proc. The International Symposium On Computer Architecture, pp. 254-263.

Haraszti, T.P. 2002. *CMOS Memory Circuits*. Kluwer Academic Publishers. 61-65.

Hennessy, J.L., Patterson, D.A. 2003. *Computer Architecture: A Quantitative Approach*. Morgan Kaufman Publisher. 376-430.

Hung, L.D. 2004. Dynamic Cache Way Allocation for Power Reduction. PhD Dissertation (Doktora Tezi). The University of Tokyo, Japan. s: 4-13.

Inoue, K., Ishihara, T., Murakami, K. 1999. Way-predicting Set-associative Cache For High Performance and Low Energy Consumption. International Symposium on Low Power Electronics, 273-275.

Jeong, T. 2004. Dynamic CMOS Circuit Power Dissipation Methodology in Low Power High Bandwidth Chip Design. PhD Dissertation (Doktora Tezi). The University of Texas, Austin, USA. s: 3.

Johnson, T.L., Hwu, W.W. 1997. Run-time Adaptive Cache Hierarchy Management via Reference Analysis. In Proc. International Symposium On Computer Architecture.

Jouppi, N.P. 1990. Improving Direct-mapped Cache Performance By The Addition Of A Small Fully-Associative Cache And Prefetch Buffers. Proc. 17th International Symposium On Computer Architecture. p. 364-373.

Kadayıf, I., Sivasubramaniam, A., Kandemir, M., Kandiraju, G., Chen, G. 2002. Generating Physical Addresses Directly For Saving Tlb Energy. International Symposium on Microarchitecture, Istanbul, Turkey. 185-196.

Kadayif, I., Sivasubramaniam, A., Kandemir, M., Kandiraju, G, Chen, G. 2005. Optimizing Instruction TLB Energy Using Software And Hardware Techniques. ACM Transactions on Design Automation of Electronic Systems, 10(2), 229-257.

Kaxiras, S., Hu, Z., Martonesi. 2001. Cache Decay: Exploiting General Behavior To Reduce Cache Leakage Power. Proceedings of the 28th annual international symposium on Computer architecture, *IEEE*. 240-251.

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

Kim, N.S. 2004. Circuit and Microarchitectural Techniques for Processor On-Chip Cache Leakage Power Reduction. PhD Dissertation (Doktora Tezi). University of Michigan, USA. s: 4.

Kim, N.S., Flautner, K., Blaauw, D., Mudge, T. 2002. Drowsy Instruction Cache-Leakage Power Reduction Using Dynamic Voltage Scaling. In Proc. International Symposium On Microarchitecture.

Kim, N.S., Flautner, K., Blaauw, D., Mudge, T. 2004. Circuit And Microarchitectural Techniques For Reducing Cache Leakage Power. *IEEE Transactions on VLSI*, 12(2), 167-184.

Kim, N.S., Flautner, K., Blaauw, D., Mudge, T. 2004. Single VDD And Single VT Super-Drowsy Techniques For Low-Leakage High-Performance Instruction Caches. In Proc. International Symposium On Low Power Electronics And Design.

Krishnamurthy, K., Alvandpour, A., Mathew, S., Anders, M., De, V., Borkar, S. 2002. High-Performance, Low-Power, And Leakage-Tolerance Challenges For Sub-70nm Microprocessor Circuits. ESSCIRC 2002.

Kumar, S., Wilkerson, C. 1998. Exploiting Spatial Locality In Data Caches Using Spatial Footprints. Proc of 25th Annual ACM/IEEE ISCA.

Lai, A.C., Fide, C., Falsafi, B. 2001. Dead-Block Prediction And Dead-Block Correlating Prefetchers. Proceedings of the 28th annual international symposium on Computer architecture, IEEE. 144-154.

Li, L., Kadayif, I., Tsai, Y.F., Vijaykrishnan, N., Kandemir, M., Irwin, M.J., Sivasubramaniam, A. 2002. Leakage Energy Management in Cache Hierarchies. 11th International Conference on Parallel Architectures and Compilation Techniques (PACT'02), pp. 131-140.

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

Lim, H. 1999. On The Integration of Compiler-Directed Cache Coherence And Data Prefetching. PhD Dissertation (Doktora Tezi). University of Illinois, Urbana-Champaign, USA. s: 24-29.

Luk, C.K., Mowry, T.C., 1996. Compiler Based Prefetching For Recursive Data StrucTures. In Procedings of the 7th International Conference on Architectural Support For Programming Languages and Operating Systems.

Mano, M. 1993. *Computer System Architecture* (3th ed.). Prentice-Hall Publishers. 456-468.

McIntosh, N. 1998. Compiler Support For Software Prefetching. PhD Dissertation (Doktora Tezi). University of Texas, USA. s:13-15.

Metcalf, C. 1993. Data Prefetching: A Cost/Performance Analysis. http://www.incert.com/metcalf/papers/prefetch/

Montenaro, J., Witek, R.T., Anne, K., Black, A.J., Cooper, E.M., Dobberpuhl, D.W., Donahue, P.M., Eno, J., Hoeppner, W., Kruckemyer, D., Lee, T.H., Lin, P.C.M., Madden, L., Murray, D., Pearce, M.H., Santhanam, S., Snyder, K.J., Stephany, R., Thierauf, S.C. 1996. A 160MHz, 32-b, 0.5-W CMOS RISC Microprocessor. *IEEE J. SolidState Circuits*, vol.32, (11): 1703-1714.

Murdocca, M., Heuring, V.P. 1999. *Principles Of Computer Architecture*. Prentice-Hall Publishers. 266-277.

Null, L., Lobur, J. 2003. *The Essentials of Computer Organization And Architecture*. Johns and Bartlett Publishers. 237-250.

O'Boyle, M.F.P., Knijnenburg, P.M.W. 1998. Integrating Loop And Data Transformations For Global Optimisation. In Proc. International Conference On Paralel Architectures And Compilation Techniques.

Parikh, D., Skadron, K., Zhang, Y., Barcella, M., Stan, M. 2002. Power Issues Related To Branch Prediction. International Symposium on HPCA'02.

Ranganathan, P., Adve, S., Jouppi, N. 2000. Reconfigurable Caches And Their Application To Media Processing. In Proc. International Symposium On Computer Architecture, pp. 214-224.

Ro, W. 2004. Decoupled Memory Access Architectures With Speculative Pre-Execution. PhD Dissertation (Doktora Tezi). University of Southern California, USA.

Sair, S. 2003. Predictor-Directed Data Prefetching For Pointer-based Applications. PhD Dissertation (Doktora Tezi). University of California, San Diego, USA. s: 17.

Sakurai, T., Newton., A. R. 1990. Alpha-power Law MOSFET Model and Its Application to CMOS Inverter Delay and Other Formulas. *IEEE Journal on Solid State Circuits*, Vol. 25, (2):548.

Smith, J.E., Hsu, W. 1992. Prefetching In Supercomputer Instruction Caches. Proceedings of the 1992 ACM/IEEE conference on Supercomputing. 588-597.

Solihin, Y. 2002. Improving Memory Performance Using Intelligent Memory. PhD Dissertation (Doktora Tezi). University of Illinois, Urbana-Champaign, USA. s: 1-3.

Stacpoole, R., Jamil, T. 2000. Cache Memories. IEEE Potentials, 19 (2): 24-29.

Stallings, W. 2003. *Computer Organization And Architecture* (6th ed.). Prentice Hall. 16-120.

Su, C.L., Despain, A. 1995. Cache Design Trade-offs For Power And Performance Optimization: A Case Study. In Proc. International Symposium On Low Power Electronics And Desing.

Su, J. 1995. Cache Optimization For Portable Computers. PhD Dissertation (Doktora Tezi). Brigham Young University, Utah, USA.

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

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

Yang, C. 2001. The Push Architecture: A Prefetching Framework For Linked Data Structures. PhD Dissertation (Doktora Tezi). Duke University, USA. s: 8-10.

Yang, S., Powell, M.D., Falsafi, B., Roy, K., Vijaykumar, T.N. 2001. An Integrated Circuit/Architecture Approach To Reducing Leakage In Deep-Submicron High-Performance I-Caches. In Proc. International Symposium On High-Performance Computer Architecture.

Zhang, W., Hu, J. S., Degalahal, V., Kandemir, M., Vijaykrishnan, N., Irwin, M. J. 2002. Compiler-Directed Instruction Cache Leakage Optimization. 35th Annual International Symposium on Microarchitecture (MICRO-35), 208-218.

Zyuban, V., Kogge, P. 1998. Split Register File Architectures For Inherently Lower Power Microprocessors. Power-Driven Microarchitecture Workshop (in conjunction with International Symposium on Computer Architecture), 32-37.

http://research.compaq.com/wrl/people/jouppi/CACTI.html.

IMTCC, Intel Microprocessor Transistor Count Chart. (2006) http://www.intel.com/pressroom/kits/events/moores_law_40th/index.htm

ITRS, International Technology Roadmap for Semiconductors, (2005). http://public.itrs.net

Spec cpu2000 benchmark. http://www.spec.org/.

ÇİZELGELER

Çizelge No **Çizelge Adı** **Sayfa No**

ŞEKİLLER

Şekil No **Şekil Adı** **Sayfa No**

YAŞAM ÖYKÜSÜ

Kişisel Bilgiler

Adı-Soyadı: Ayhan ZORLUBAŞ

Doğum Yeri ve Yılı: İstanbul -1980

Adres: Çanakkale Onsekiz Mart Üniversitesi, Enformatik Bölümü, Anafartalar Yerleşkesi, 17100 ÇANAKKALE

Eğitim Durumu

1986-1989: Bahçelievler İlkokulu ERZİNCAN

1989-1991: Şeker İlkokulu BURDUR

1991-1994: Burdur Lisesi BURDUR

1994-1995: Cumhuriyet Lisesi BURDUR

1995-1998: Adana Erkek Lisesi ADANA

1998-2002: Çanakkale Onsekiz Mart Üniversitesi – Bilgisayar Mühendisliği Bölümü

Stajlar

Temmuz 2000: AKÇANSA Çimento (ÇANAKKALE)

Temmuz 2001: Casper Bilgisayar (ADANA)

Mesleki Deneyim

2003 - : Çanakkale Onsekiz Mart Üniversitesi, Enformatik Bölümü, Okutman

Çalışma ve İlgi Alanları

Bilgisayar Mimarisi, İşletim Sistemleri

ÇİZELGELER

Çizelge No **Çizelge Adı** **Sayfa No**

ŞEKİLLER

Şekil No **Şekil Adı** **Sayfa No**

YAŞAM ÖYKÜSÜ

Kişisel Bilgiler

Adı-Soyadı: Ayhan ZORLUBAŞ

Doğum Yeri ve Yılı: İstanbul -1980

Adres: Çanakkale Onsekiz Mart Üniversitesi, Enformatik Bölümü, Anafartalar Yerleşkesi, 17100 ÇANAKKALE

Eğitim Durumu

1986-1989: Bahçelievler İlkokulu ERZİNCAN

1989-1991: Şeker İlkokulu BURDUR

1991-1994: Burdur Lisesi BURDUR

1994-1995: Cumhuriyet Lisesi BURDUR

1995-1998: Adana Erkek Lisesi ADANA

1998-2002: Çanakkale Onsekiz Mart Üniversitesi – Bilgisayar Mühendisliği Bölümü

Stajlar

Temmuz 2000: AKÇANSA Çimento (ÇANAKKALE)

Temmuz 2001: Casper Bilgisayar (ADANA)

Mesleki Deneyim

2003 - : Çanakkale Onsekiz Mart Üniversitesi, Enformatik Bölümü, Okutman

Çalışma ve İlgi Alanları

Bilgisayar Mimarisi, İşletim Sistemleri