T.C.
ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
YÜKSEK LİSANS TEZİ

İŞLEM DEĞİŞİKLİKLERİNİN ETKİSİNİ AZALTMAK İÇİN KOMUT ÖNBELLEĞİ ERİŞİM GECİKMESİNİN KODLANMASI

Seher KIZILTEPE
Bilgisayar Mühendisliği Anabilim Dalı
Tezin Sunulduğu Tarih: **25/06/2010**

Tez Danışmanı:
Yrd. Doç. Dr. İsmail KADAYIF

ÇANAKKALE

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

SEHER KIZILTEPE tarafından **YRD. DOÇ. DR. İSMAİL KADAYIF** yönetiminde hazırlanan **"İŞLEM DEĞİŞİKLİKLERİNİN ETKİSİNİ AZALTMAK İÇİN KOMUT ÖNBELLEĞİ ERİŞİM GECİKMESİNİN KODLANMASI"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. İsmail KADAYIF

_____

Danışman

Yrd. Doç. Dr. İbrahim TÜRKYILMAZ

_____

Jüri Üyesi

Yrd. Doç. Dr. İbrahim BULUT

_____

Jüri Üyesi

Sıra No:……

Tez Savunma Tarihi: 25/06/ 2010

Prof Dr. İsmail TARHAN

_____

Müdür

Fen Bilimleri Enstitüsü

İNTİHAL (AŞIRMA) BEYAN SAYFASI

Bu tezde görsel, işitsel ve yazılı biçimde sunulan tüm bilgi ve sonuçların akademik ve etik kurallara uyularak tarafımdan elde edildiğini, tez içinde yer alan ancak bu çalışmaya özgü olmayan tüm sonuç ve bilgileri tezde kaynak göstererek belirttiğimi beyan ederim.

Seher KIZILTEPE

TEŞEKKÜR

Tezimin hazırlanması sürecinde her zaman her konudaki destek, yardım ve yönlendirmeleri için değerli Yrd. Doç. Dr. İsmail KADAYIF' a, tecrübelerini ve yardımlarını esirgemeyen değerli Yrd. Doç. Dr. İbrahim TÜRKYILMAZ' a, tez çalışmalarım boyunca beni sabırla destekleyen, güvenimi kaybetmeme hiç bir zaman izin vermeyen sevgili eşim Gökhan KIZILTEPE' ye ve bugünlere gelmemde her türlü çabayı gösteren aileme tüm kalbimle teşekkürlerimi sunarım.

Seher KIZILTEPE

SİMGELER VE KISALTMALAR LİSTESİ

SRAM (Static RAM) : Statik RAM

Cache: Önbellek

iL1 (Level 1) : Birinci Seviyeli Komut Önbelleği

CS: Muhafazakâr Yöntem

LCS: Daha Az Muhafazakâr Yöntem

CRBS: Kod Yer Değiştirmeye Dayalı Yöntem

L1 (Level 1) : Birinci Seviye

L2 (Level 2) : İkinci Seviye

P (Perfect) : Kusursuz

D (Delayed) : Geciktirilmiş

O (Oracle) : Kâhin

TLB (Translation Lookaside Buffer) : Adres Dönüştürme Önbelleği

BTB (Branch Target Buffer): Dallanma Hedef Tamponu

CFG (Control Flow Graph) : Kontrol Akış Diyagramı

ISA (Instruction Set Architecture) : Komut Küme Mimarisi

TLB (Translation Lookaside Buffer): Adres Dönüşüm Tamponu

PT (Page Table): Sayfa Tablosu

ÖZET

İŞLEM DEĞİŞİKLİKLERİNİN ETKİSİNİ AZALTMAK İÇİN KOMUT ÖNBELLEĞİ ERİŞİM GECİKMESİNİN KODLANMASI

Seher KIZILTEPE
Çanakkale Onsekiz Mart Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Tezi
Danışman: Yrd. Doç. İsmail KADAYIF
25.06.2010, 19

Daha küçük geometrili devrelerin üretilmesi teknolojik açıdan mümkün olurken, bu durum kanal genişliği, kapı oksit kalınlığı ve dopant iyon yoğunluğu gibi cihaz (transistor) karakteristiklerinde dramatik değişkenliklere yol açabilen kritik fiziksel parametrelerin kontrolünü son derece zorlaştırmaktadır. SRAM hücreleri küçük boyutlu transistorlarla tasarlandıklarından bu değişkenlikler SRAM' lerde daha yaygın olarak ortaya çıkabilmektedir. Bunun sonucu olarak aynı şekilde tasarlanmış devre bileşenlerinin erişim gecikmelerinde ve enerji tüketimlerinde dalgalanmalar söz konusu olabilmektedir. Örnek olarak bu, aynı önbelleğe ait farklı blokların farklı erişim gecikmelerine sahip olması anlamına gelmektedir. Bu problemle baş etmenin en basit yolu en yavaş erişim gecikmesi paradigmasını kullanmaktır. Diğer bir ifadeyle bu, bütün önbellek blokların erişim gecikmesinin en yavaş bloğun erişim gecikmesine sahip olduğunu kabul etmektir. Bu şekildeki düzenli önbellek yönetimi, basit olmasına rağmen performansta kayıplara yol açabilmektedir. Bu husus, bu yöntemin geleceğin yüksek performanslı işlemcilerde uygulanmasısın dezavantajlı kılar. Bu çalışmada biz, işlem değişkenliklerinin komut önbelleklerinde yol açtığı erişim gecikmelerinin olumsuz etkilerini azaltmak için derleyiciye dayalı bazı yöntemler üzerinde duracağız. Önerdiğimiz derleyici teknikleri, komutların erişim gecikmelerinin komutlarda kodlanması yeniliğini getirmektedir. Daha açık olarak ifade etmek gerekirse, bir komutun önbellek erişim gecikmesi o komuttan önce çalıştırılan komuta/komutlara kodlanır. Deneysel sonuçlarımız SPEC2000'a ait uygulamalar için geliştirdiğimiz uyarlanabilir önbellek yönetim yöntemlerimizin, işlem değişkenliklerinden kaynaklanan performans kayıplarını azaltmada oldukça etkili

olduklarını göstermektedir. Buna ek olarak sonuçlarımız, kod yer değişikliğinin erişim gecikmeleri kodlaması ile kullanılması halinde performans kayıplarının daha da azaltılabileceğini göstermektedir. Örneğin, hem kod yer değişimi hem de kodlamayı kullanan yöntemlerimizden birinin, %30,2'lik performans kaybının %5,9'a indirilebilmesini sağlamaktadır.

Anahtar sözcükler: İşlem Değişkenliği, Erişim Gecikmesi, Komut Önbelleği, SRAM

ABSTRACT

ENCODING INSTRUCTION CACHE ACCESS LATENCY FOR MITIGATING THE IMPACT OF PROCESS VARIATIONS

Seher KIZILTEPE

Çanakkale Onsekiz Mart University

Graduate School of Science and Engineering

Chair for Computer Engineering Thesis of Master of Science

Advisor: Assistant Prof. Dr. İsmail KADAYIF

25.06.2010, 19

As technology moves toward finer process geometries, it is becoming extremely difficult to control the critical physical parameters such as channel length, gate oxide thickness, and dopant ion concentration, which in turn leads to dramatic variations in device characteristics. These variations are more pronounced in SRAM cells as they are typically designed with minimum sized transistors for density reasons, and they manifest themselves as fluctuations in access latencies as well as power consumptions of the identically-designed components. This means for example that the different lines of the same cache may have different access latencies. A simple solution to address this problem is to adopt the worst case latency paradigm, i.e., all the cache lines are assumed to have the latency of the slowest cache line. While this egalitarian cache management is simple, it may introduce a significant performance overhead, making it unfeasible for future high performance processors. In this study, we investigate several compiler techniques to mitigate the effect of process variation on the instruction cache. Our proposed techniques annotate the cache access latency of instructions within themselves. More specifically, the access latency of the cache set of the instruction to be accessed next is encoded in its predecessor instruction(s). Our experimental results with the SPEC2000 suite show that, using our adaptive cache management policies based on varying instruction cache access latencies, it is possible to reduce the performance overhead (resulting from process variation) significantly. In addition, our results also show that applying code relocation with encoding can reduce performance overheads even further. For example, using one of our schemes that employs both code relocation and encoding, we can cut the original 30.2% average performance overhead introduced by an access mechanism employing the worst case latency paradigm to 5.9%.

Keywords: Process Variation, Access Latency, Instruction Cache, SRAM

İÇERİK

Sayfa

BÖLÜM 1

GİRİŞ

Daha küçük boyutlu transistor geometrilerinin geliştirilmesine paralel olarak transistor kalitesini istenen sınırlarda tutabilmek gittikçe zorlaşmaktadır. Bunun sonucu olarak parametre değişkenliği gelecek nesil tasarımlar için önemli bir problem olarak karşımıza çıkmaktadır. Genel olarak parametre değişkenliklerini üç ana grupta toplayabiliriz. Bunlar voltaj, sıcaklık ve işlem değişkenlikleridir. Voltaj değişkenlikleri güç kaynağındaki dalgalanmalar ve anahtarlama faaliyetlerindeki değişenliklerden kaynaklanır. Sıcaklık değişkenlikleri ise farklı işlevsel birimlerdeki ısı yayılımı ve soğutma materyallerinin aynı özelliklere sahip olmayışından ileri gelir. Bu çalışmada biz, sadece komut önbelleğindeki işlem değişkenliği ve bunun performans üzerinde oluşturduğu olumsuz etkileri üzerinde duracağız.

İşlem değişkenliği kanal genişliği, kapı oksit kalınlığı ve dopantların düzeni gibi çeşitli devre parametrelerinde istenilen veya niyet edilen değerlerinden sapmalar olarak tanımlanabilir. Bu değişkenlikler benzer olarak tasarlanmış komşu aygıtlar arasında olabildiği gibi (intra-die variation) (Agarwal ve ark., 2003), aynı teknolojiyle tasarlanmış farklı yongalar (inter-die variation) (Nassif, 2001) arasında da olabilir. Bu değişkenlikler etkilerini, hem performans ve hem de güç tüketimi karakteristiklerinde gösterebilmektedirler (Borkar ve ark., 2003). Örneğin işlem değişkenliğinin sonucu olarak, bir önbellek bloğuna erişim başka bir bloğa kıyasla daha fazla zaman alabilir veya daha fazla enerji tüketebilir. Aynı devrenin farklı kısımlarının aynı özellikte olması isteneceğinden, bu husus genelde sorunlara sebep verir.

İşlem değişkenliğinden kaynaklanan davranış karakteristiklerinin olumsuz etkilerini gidermeye yönelik çok çeşitli devre seviyesi çözümler (Chen ve Naffziger, 2005; Gregg ve Chen, 2004; Tschanz ve ark., 2002) ve mimari seviyesi çözümler (Meng ve Joseph; 2006) mevcuttur. Ancak aynı amaca yönelik derleyici düzeyinde çözümlerin olup olamayacağının belirlenmesi, eğer varsa bu çözümlerin devre ve mimari seviyeli çözümlerle kıyaslanması ve etkileşimlerinin araştırılması oldukça önemlidir. Bizim bu çalışmadaki amacımız bu boşluğu gidermeye çalışmaktır.

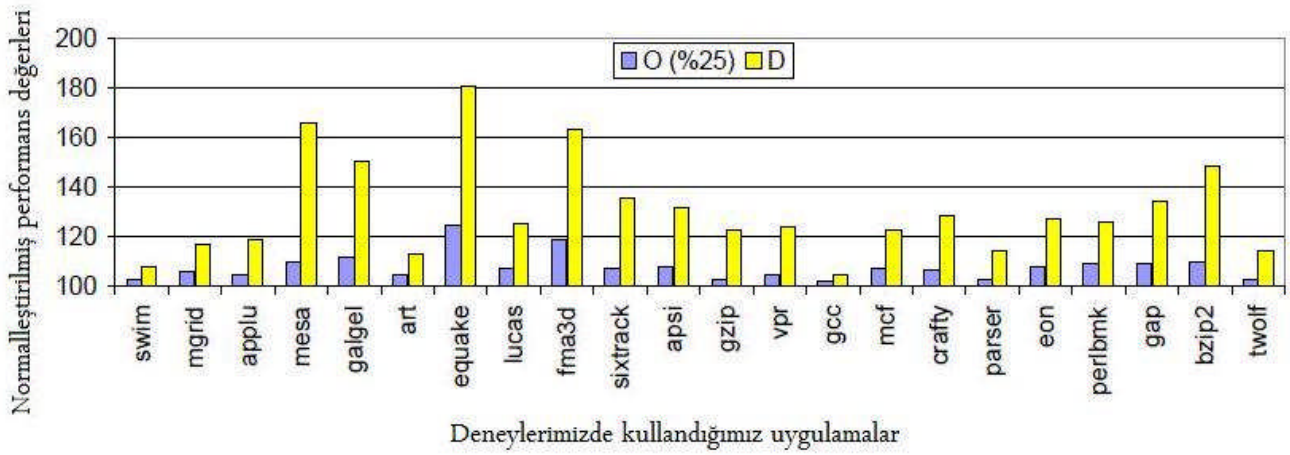
İşlem değişkenlikleri SRAM' ler, yazmaç dosyaları ve fonksiyonel üniteler gibi sistem bileşenlerinin önemli parametrelerini etkileyebilir. Bunun temel sebebi, bu tip bileşenlerin minimum boyutlu transistorlarla tasarlanmasıdır (Papanikolaou ve ark., 2005). Bu değişkenlikler daha uzun erişim gecikmesi ve daha fazla enerji tüketiminin yanında, karasız okuma/yazma işlemleri (Agarwal ve ark., 2005; Chen ve ark., 2005) gibi güvenirlik kaygılarına da yol açabilir. Literatürde işlem değişkenliğinden kaynaklanan SRAM bellek hatalarına çözüm getirmeye çalışan çeşitli yöntemler vardır. Örneğin, işlem değişkenliklerinin sebep olduğu güç ve erişim ihlallerinden kaynaklanan hatalı cip kayıplarını (yield loss) azaltmak için veri önbelleklerine yönelik bazı mimari temelli teknikler (Özdemir ve ark., 2006) önerilmiştir. Özdemir ve arkadaşlarının yaptıkları çalışmaları özetleyecek olursak; geliştirdikleri yöntemlerin birinde tasarım sonucu erişim gecikmesi ihlali yapan ve/veya aşırı sızıntı enerji tüketimine sahip önbellek yollarını kapatmaya çalışmaktadırlar. Bu yöntem YAPD (Yield-Aware Power-Down) olarak adlandırılır. Diğer bir yöntemde ise yolların yerine ihlalleri yapan yatay bölgeler kapatılır (Horizantal YAPD, H-YAPD). Diğer bir yöntemde değişken erişim gecikmeli veri önbelleği geliştirilmiştir (Variable - Latency Cache Architecture, VACA). Bu yöntem, farklı yükleme komutlarının farklı sürelerde tamamlanmasına izin verir.

Yukarıda bahsi geçen tekniklerin başlıca dezavantajları ya önemli ölçüde performans kayıplarına yol açmaları (Pour ve Hill, 1993) ya da ek devrelere (Vergos ve Nikolos, 1995) gereksinim duymalarıdır. Oysa bu çalışmanın temel hedefi, lüzumsuz donanım karmaşıklığına yol açmaksızın veya performansa kısıtlama getirmeksizin komut önbelleğindeki (instruction data cache, iL1) erişim gecikmeleri farklılığından kaynaklanan performans kayıplarını azaltmaya yönelik derleyici temelli teknikler geliştirmektir.

Bu çalışmanın geri kalan kısmında aksi belirtilmedikçe önbellek kavramını birinci seviye komut önbelleği (L1 instruction cache, iL1) anlamında kullanacağız. Ayrıca ön bellek bloğu ve ön bellek satırı kavramlarını aynı anlamda kullanacağız. Sorunlu (mükemmel olmayan, imperfect) kümeyle, işlem değişkenliğinden etkilenmiş en az bir önbellek bloğuna sahip ön bellek kümesini kastedeceğiz. İşlem değişkenliğinin önbelleklerde oluşturduğu sorunlardan belki de en önemlisi veri erişim gecikmelerinin düzenli olamamasıdır. Yani farklı önbellek satırlarının erişim gecikmelerinde farklılıkların olabilmesidir. Bu problemin en basit çözümü, en kötü erişim gecikmesini bütün önbellek satırları için benimsemektir. Bu pratikte herhangi bir önbellek bloğunun erişim gecikmesinin, o bloğun sorunlu bir önbellek kümesine ait olup olmadığına bakmaksızın en

kötü erişim gecikmesine sahip olan bloğun erişim gecikmesine eşit kabul edilmesi ve tasarımın buna göre yapılması anlamına gelir. Bu kabul tasarımı oldukça basitleştirmesine rağmen, komut önbelleğine erişim program yürütümünün kritik yolu üzerinde yer aldığından performansa olumsuz yönde etki eder. Performans üzerindeki bu olumsuz etkinin daha küçük transistor geometrilerine gidildikçe artacağı beklenmektedir (Bowman ve ark., 2002).

Şekil 1 bize, işlem değişkenliğinden etkilenmiş bir veri önbelleğinin performans üzerindeki etkisini göstermektedir. Grafikteki sonuçlar işlem değişkenliğinin olmadığı veri önbelleğine ait performans sonuçları temel alınarak verilmiştir. Her bir veri önbellek erişimin 1 çevrimde tamamlandığı (eğer veri önbellekte mevcutsa) önbellek erişim mekanizmasını bu çalışmada mükemmel (Perfect, P) erişim mekanizması olarak tanımlıyoruz. Grafikteki her bir uygulama programına karşı gelen ilk çubuk, kâhin (Oracle, O) mekanizmasına ait performans sonucunu göstermektedir. O yöntemi, her bir veri başvurusunun mükemmel bir kümeye karşı gelip gelmediğinin önceden öngörülebilmesi prensibine dayanmaktadır. Daha önceden de vurguladığımız gibi mükemmel bir kümeyle, tüm bloklarının işlem değişkenliğinden etkilenmeyen bir önbellek kümesini kastediyoruz.



Şekil 1. O ve D yöntemleri için normalleştirilmiş çevrim (cycle) değerleri. Tüm değerler, önbelleğin işlem değişkenliğinden etkilenmediği duruma karşı gelen mükemmel (P) yöntemin sonuçlarına göre normalize edilmiştir. O yöntemi için önbellek kümelerinin %25'inin işlem değişkenliğinden etkilendiği kabulünü yapmaktayız.

Bu deneylerimizde, sorunlu kümeye ait bir önbellek bloğuna erişimin ekstra 1 çevrim daha alarak 2 çevrimde tamamlandığını varsaydık. Eğer erişilen blok sorunlu bir kümeye ait değilse, veri erişimi 1 çevrimde tamamlanmaktadır. Buradaki sonuçlar kümelerinin %25'i sorunlu olan bir veri önbelleği içindir. Yukarıdaki kabullere ek olarak bu deneylerde sorunlu kümelerin önbellek boyunca rastgele dağıldıklarını düşündük. Grafikteki ikinci çubuklar en kötü erişim gecikmesi mantığına dayalı önbellek erişim mekanizmasına aittir. Biz, bu tip önbellek erişim mekanizmasını geciktirilmiş (Delayed, D) erişim olarak adlandırmaktayız. Bu şekle göre O (%25) ve D yöntemlerine ait performans kayıpları sırasıyla %8,1 ve %30,2'dir. Yine bu grafikten mesa, equake ve fma3d gibi bazı uygulamalar için en kötü erişim gecikmesini temel alan yöntem için çalışma sürelerinin %60'dan daha fazla arttığını görebiliriz. Özellikle yüksek performanslı bilgisayar sistemleri için kabul edilemez bu durum bizi, veri önbelleklerinde işlem değişkenliğinin sebep olduğu performans kayıplarını telafi edebilmek için yeni teknikler araştırılmasına teşvik etmektedir.

Bu çalışmanın ana katkısı en kötü erişim gecikmeli yönteme alternatif yöntemler geliştirmek ve bu yöntemleri performans açısından irdelemektir. Bu alternatif yöntemler, derleyici analizlerini kullanmak suretiyle uygulamanın kontrol akış grafiğini (Control Flow Graph, CFG) inşa ederek, her bir komuta ait erişim gecikmesini ilgili komuttan hemen önce gelen komuta/komutlara gömerler. Böylece bir önceki komutun kod çözüm evresinde sonradan gelen komuta ait erişim gecikmesi donanıma tanıtılmış olacağından, donanım veriye erişirken bu bilgiye göre kendini ayarlama fırsatı bulur. Komutlara ait erişim gecikmeleri komutların kullanılmayan bit boşluklarına (slot) kodlanır. Komut, iL1 önbelleğinden işlemciye getirilir getirilmez (fetch stage), bu bitler çözümlenerek (decode edilerek) sonraki komuta ait önbellek erişim gecikme süresi öğrenilir. Bu bilgi sonraki erişim için donanımın ne kadar beklemesi gerektiğini belirler. Daha sonra bahsedileceği gibi, bu fikre dayanan çeşitli derleyici yöntemlerini tanıtacağız ve onları performans açısından birbirleriyle kıyaslayacağız. Komutlara ait erişim gecikmelerini önceden öğrenmenin diğer bir yolu, gecikmeleri saklayan bir donanım tablosunun kullanılmasıdır. Öngörüye dayalı olarak tahmin edilen küme adresiyle bu tablo indekslenerek sonraki komuta erişmek için gereken süre belirlenebilir. İki sebepten dolayı bu alternatif yöntemi tercih etmedik. Bunlardan birincisi bu ek tablo alan ve enerji tüketimi açısından ek külfet getirir. İkincisi ise bu tablonun kendisi işlem değişkenliğine maruz kalabileceğinden performans gecikmelerine sebep olabilir.

Bu tezin geri kalan kısmı aşağıdaki gibi düzenlenmiştir. Sonraki bölümde işlem değişkenliğinin yol açtığı performans kayıplarını azaltmaya yönelik donanım desteklerini açıklayacağız. Yine o bölümde erişim gecikmelerinin komutlara nasıl kodlandığı anlatılıp, önerdiğimiz derleyici yöntemleri tanıtacağız. 3. bölümde ise deneysel düzeneğimizi ve deneylerden elde ettiğimiz sonuçları vereceğiz. Son bölümde ise bulgularımızı özetleyeceğiz.

BÖLÜM 2

ÖNCEKİ ÇALIŞMALAR

Donanımızda bir önbellek bloğunu depolayabilecek kapasiteye sahip ve iL1 önbelleği ile CPU arasına yerleştirilmiş bir tamponun olduğunu varsayıyoruz. İhtiyaç duyulan komutlar bu tamponda mevcutsa komut erişim istekleri bu tampondan karşılanır. Aksi takdirde bir önbellek erişimi gerçekleştirilir ve ihtiyaç duyulan komutu barındıran komut bloğu önbellekten okunarak bu tampona yerleştirilir. Bu işleme paralel olarak ihtiyaç duyulan komut ise CPU'ya gönderilir. Sonraki komut erişimleri mümkünse tampondan karşılanmaya çalışılır. İşlemcimizin komut küme mimarisinin (Instruction Set Architecture, ISA) kullanılmamış bit boşlukları barındırdığını kabul ediyoruz. Çoğu işlemciler gittikçe 64-bit'e destek vermeye başlamıştır. 64-bitlik mimariler, komutları kodlamak için oldukça uzun bir bit alanı sağlamaktadır. Genellikle bu platformlar, ihtiyaçlar doğrultusunda sonradan eklenebilen yeni komut ilavesine destek vermek için kullanılmayan bit boşlukları barındırırlar. Bu boşlukları komutların erişim gecikmelerini kodlamak için kullanabiliriz. Derleme zamanında genellikle komutların eşleştiği kümeler (komutun yer aldığı bloğun aksine) belirlenebildiğinden erişim gecikmelerini önbellek kümeleri seviyesinde kodlamaktayız (blok seviyesinde değil). Bir kümenin erişim gecikmesi o kümenin barındırdığı en yavaş bloğun erişim gecikmesi olarak belirlenmektedir.

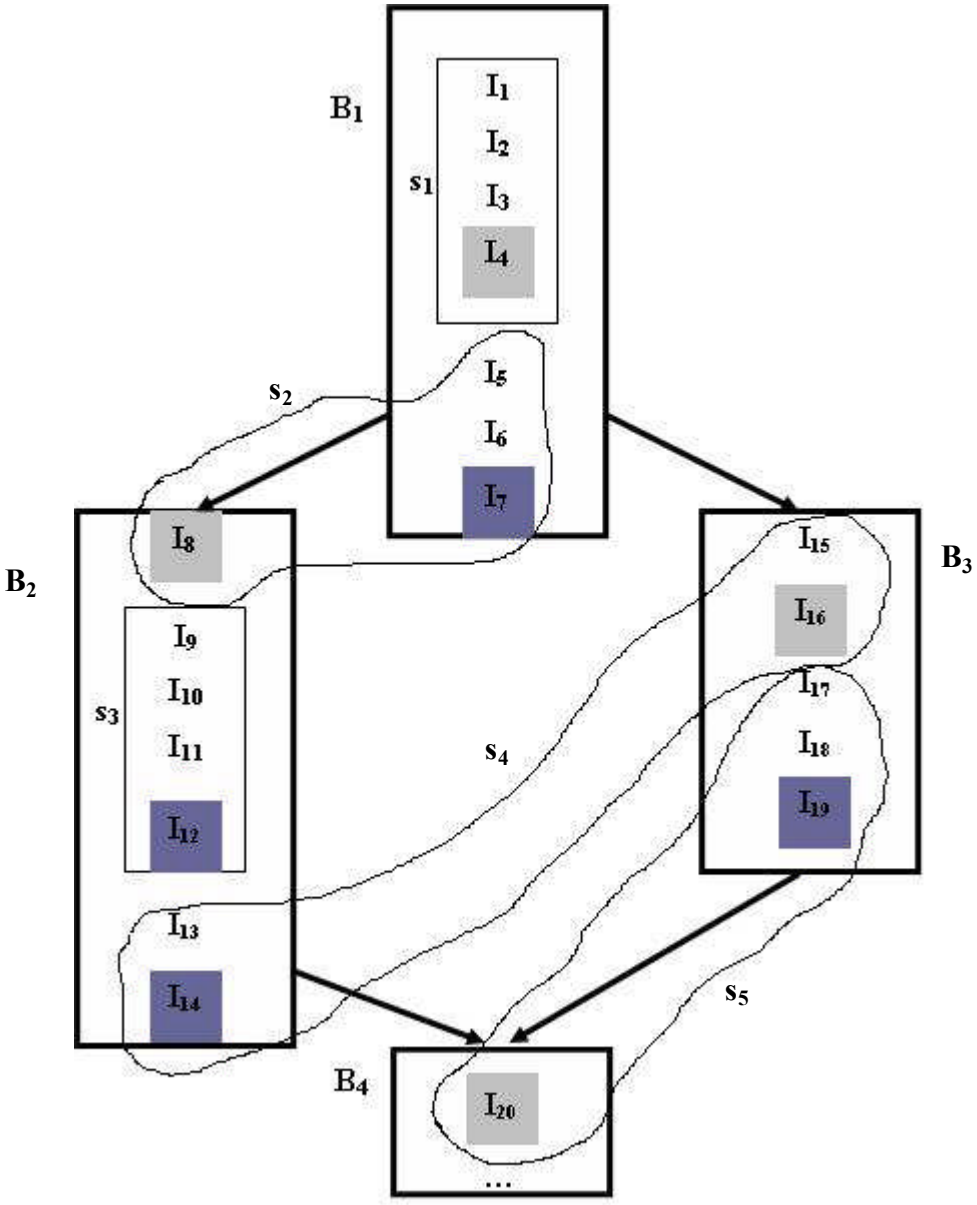
Erişim gecikmelerini belirlemek için March testini (Chen ve ark., 2005) kullanabiliriz. Bu test orijinal olarak bellek bileşenlerinin fonksiyonelliğini test etmek için önerilmiştir ve belleğin değişik bölgelerine yapılan bir dizi işlemler gerektirir. Belleklerde işlem değişkenliğinden kaynaklanan erişim zamanı hatalarını ortaya çıkarmak için March testine bir takım ilaveler yapılmıştır (Chen ve ark., 2005).

Komut erişim gecikmelerini kodlamak için derleyici öncelikle programa ait CFG' yi inşa eder. Derleyici daha sonra sanal adreslere göre komutları önbelleğin kümeleriyle ilişkilendirir. Sanal adresler işlemci tarafından çalışma zamanında üretilen adresler olup, bellek yönetim ünitesi (Memory Management Unit, MMU) tarafından çalışma zamanında fiziksel adreslere dönüştürülür. MMU ise hem yazılım hem de donanımdan ibarettir. Donanım kısmında son zamanlardaki adres dönüşümlerini tutan adres dönüşüm tamponu (Translation Lookaside Buffer, TLB) gibi tamponlar yer alır. Yazılım kısmı ise, işletim

sistemi denetiminde yürütülen ve bellekte tutulan sayfa tablolarına (Page Table, PT) erişebilen bir takım rutinlerden oluşur. Çoğu CPU için komut önbelleği indekslemesi sanal adreslerle yapılabildiğinden komut erişim gecikmelerinde fiziksel adreslerin yerine sanal adreslerin dikkate alınması herhangi bir soruna yol açmaz. n farklı erişim gecikmesini kodlamak için *logn* tane bit boşluklarını kullanmak yeterlidir.

2.1. Erişim Gecikmelerini Kodlamak İçin Önerilen Yöntemler

Bizler bu bölümde önerdiğimiz yöntemleri açıklamaya çalışacağız. Her bir yöntemi Şekil 2'de verilen örnek CFG'ye göre açıklamaya çalışacağız.



Şekil 2. Örnek Kontrol Akış Diyagramı (CFG).

Anlaşılması basit olması açısından her bir önbellek bloğunun şekilde görüldüğü gibi 4 komutu barındırabildiğini kabul edelim. Örneğin, B_1 temel bloğundaki (basic block) I_1, I_2, I_3 ve I_4 komutları bir blok oluşturur ve bu bloğa karşı gelen önbellek kümesi ise s_1'dir. Şimdiye kadarki anlatımımızda, her bir komutun o komuttan sonra çalışacak olan komutun erişim gecikmesini kodladığını üstü kapalı olarak kabul ettik. Ancak bizim sistemimizde önbellek bloğunun tamamının önbellekten okunup bir tamponda tutulduğu varsayıldığından, biz sadece iki farklı tipteki komutlara erişim gecikmelerini kodlamaya gereksinim duyarız. Bunlardan birincisi, yürütüm açısından sonra gelen komutu barındıran kümenin erişim gecikmesi ilgili bloğun son komutuna kodlanır (örneğin I_4 ve I_{12} komutları). İkinci olarak, hedefi blok sınırları dışında olan ve programın yürütüm akışını değiştiren komutlardır (örneğin I_7 ve I_{14} komutları). Daha önceden de bahsettiğimiz gibi n farklı erişim gecikmesini kodlamak için *logn* bit boşluğuna ihtiyaç duyarız. Basitlik açısından, bu çalışmada 1 bit boşluğu kullanarak iki farklı erişim gecikmesini kodlamaktayız. Yukarıdaki şekilde açık gölgeli komutlar sorunsuz kümelerle eşleşen önbellek kümelerine ait erişim gecikmesini kodlamaktadır. Koyu gölgeli komutlar ise sorunlu kümelerle eşleşen önbellek kümelerine ait erişim gecikmesini kodlamaktadır.

Yukarıda verilen bilgileri temel alarak şimdi önerdiğimiz üç yöntemi açıklamaya çalışalım.

2.2.1 Muhafazakâr Yöntem (Conservative Scheme CS)

Bu yöntemde derleyici herhangi bir analiz yapmadan dallanma komutu hedefinin sorunlu önbellek kümesine düştüğü kabulünü yapar. Bu aynı zamanda Şekil 2'de yapılan kabuldür. Bu yönteme göre I_7, I_{14} ve I_{19} komutları sorunlu kümelerin erişim gecikmesiyle kodlanır. Şekilde bu tip komutlar koyu gölgeli olarak gösterilmektedir.

2.2.2 Daha Az Muhafazakâr Yöntem (Less Conservative Scheme LCS)

Bu yöntemde derleyicimiz her bir dallanma komutunun hedefini analiz eder. Dallanmayı izleyen komut (fall-through) ve hedefteki komutun her ikisi de mükemmel kümelere karşı gelirse ilgili dallanma komutuna sorunsuz kümelerin gecikmesi kodlanır. Aksi takdirde dallanma komutuna sorunlu kümelerin erişim gecikmesi kodlanır. Her ne kadar LCS yöntemi CS yönteminden daha iyi olsa da, derleyici ikili kodda bazı dallanma komutlarının hedefini belirleyemeyeceğinden LCS yine muhafazakârlık gösterir.

Derleyicinin hedefini belirleyemeyeceği dallanmalara örnek olarak hedef adresi bir yazmaçtan elde edilen dallanma komutlarını gösterebiliriz. Bu tip komutları statik olarak analiz edilemeyen dallanma komutları olarak adlandırmaktayız. Bu tip komutların sayısı neyse ki statik olarak analiz edilebilen dallanma komutlarına kıyasla oldukça azdır. Bu yönteme göre, eğer Şekil 2'de s_2 ve s_4 kümelerinin her ikisi de sorunsuz ise LCS yöntemi I_7 komutuna sorunsuz kümelerin erişim gecikmesini kodlar.

2.2.3 Kod Yer Değiştirmeye Dayalı Yöntem (Code Relocation Based Scheme CRBS)

Bu yöntemde LCS yönteminde olduğu gibi derleyici dallanma komutlarının hedeflerini analiz etmeye çalışır. Buna ek olarak derleyici kod yer değişikliği yapmak suretiyle, hem dallanmayı izleyen komutu hem de dallanmanın hedefindeki komutu aynı erişim özellikli önbellek kümelerine yerleştirmeye çalışır. Eğer bunlardan biri sorunlu/sorunsuz kümeye düşerse derleyici diğer komutu da sorunlu/sorunsuz bir kümeyle eşleştirmeye çalışır. Ayrıca bu yöntemde, hem çok sıklıkla çalıştırılan dallanma komutlarını (örneğin, döngü kapama dallanmaları, loop closing branches) izleyen komutların hem de onların hedef adreslerindeki komutların sorunsuz kümelerle eşleştirilmesine çalışılır. Böylece çok sıklıkla erişilen komutların işlemciye hızlı bir şekilde getirilmesi temin edilerek performans artışı sağlanabilmektedir.

BÖLÜM 3

DENEYSEL DÜZENEK VE SONUÇLAR

3. 1. Kurulum

Tablo 1. Yapılandırma parametreleri ve deneylerimizde kullandığımız değerleri

İşlemci Çekirdeği (Pocessor Core)	
Fonksiyonel birimler	8 tamsayı ALU 4 tamsayı çarpma/bölme 8 FP toplama, 4 FP çarpma 4 FP bölme/karekök alma
RUU büyüklüğü	256 komut
LSQ büyüklüğü	64 komut
Fetch/decode/issue/ commit genişliği	8 komut/cycle
Fetch Queue büyüklüğü	8 komut
Önbellek ve Bellek Hiyerarşisi	
L1 komut önbelleği	64KB, 4-way (LRU), 64 byte block, 1 cycle latency
L1 veri önbelleği	64KB, 4-way (LRU), 32 byte blocks, 1 cycle latency
L2 önbelleği	1MB unified, 8-way (LRU), 128 byte blocks, 12 cycle latency
Data/Komut TLB	128 entries, full-associative 30 cycle miss penalty
Bellek	160 cycle latency
Sayfa büyüklüğü	8K
Dallanma Tahmini (Branch Prediction)	
Dallanma Tahmincisi (Branch predictor)	Combined, Bimodal 2K table, 2-Level 1K table, 8 bit history, 4K chooser
Dallanma Hedefi Tamponu (Branch Target Buffer, BTB)	1K-entry, 4-way
Geri dönüş adres yığını (Return-address stack)	8
Yanılgı cezası (Mispredict penalty)	20 cycles

Daha önce açıklanan yöntemlerimizin hepsini SimpleScalar 3.0'in (SimpleScalar toolset) kodunu değiştirmek suretiyle gerçekleştirdik. SimpleScalar özellikle yüksek performanslı işlemcilerin bellek hiyerarşisi ve işlemci performansını irdelemek için kullanılır. Bu çalışmada biz daha çok sim-outorder bileşeninin kodunu değiştirdik. Bu, out-of-order issue özellikli superscalar makineleri simüle eden SimpleSclar'ın en önemli bileşenlerinden biridir. Deneylerimizde Alfa-benzeri bir mimariyi hedef aldık. Bütün derleyici analizleri ve kod yer değiştirme yöntemleri önceden derlenmiş ikili kodlar üzerinde gerçekleştirildi. Deneylerimizde kullandığımız işlemci ve bellek hiyerarşisinin temel özellikleri Tablo 1'de gösterilmektedir. İşlemci ve bellek hiyerarşisinin temel özelliklerine bir göz atalım. İşlemcimizde tamsayı işlemleri yapabilen 8 ALU bulunmaktadır. İşlemci tamsayı çarpma ve bölme yapabilen 4 fonksiyonel ünite barındırmaktadır. Kayan noktalı işlem yapabilen 8 toplama, 4 çarpma, 4 bölme ve 4 karekök alma ünitesi vardır. İşlemcimizin aynı çevrimde 8 tane komutu salabilmektedir ve yine aynı çevrimde 8 komutun sonuçları nihai olarak ilgili depolama birimlerine aksettirilebilmektedir. L1 komut önbelleğimiz 64 KB kapasiteli, 4 yollu, blokların kapasitesi 64 byte olup, erişim gecikmesi 1 cycle'dir. L2 önbelleğimiz ise bütünleşik olup (hem komut hem de verileri barındıran), 1 MB kapasiteli, LRU algoritmasına göre yönetilen 8 yollu, 128 byte bloklara sahip ve 12 cycle erişim gecikmesine sahiptir. Ana bellek erişim gecikmesi 120 cycle'dır. Komut ve veri adres dönüşüm tamponlarının özellikleri aynı olup 128 giriş barındırmaktadırlar. Sayfa büyüklüğü ise 8 KB'tır. İşlemcimiz bir dallanma tahmincisine sahiptir. Bu dallanma tahmincisi birleştirilmiş, bimodal tekniği ve 8 bit history bilgisine göre işler. Geri dönüş adres yığınımız 8 adres barındırabilecek kapasitededir. Dallanma tahmin mekanizması için yanılgı cezası ise 20 cycle'dir.

Deneylerimizde SPEC2000 suite'deki (SPEC 2000 suite) uygulamaları kullandık. SPEC2000'deki herhangi bir uygulamayı baştan sona kadar simüle etmek oldukça fazla zaman aldığından, her bir uygulama programı için önce ilk 1 milyar komutu hızlı bir şekilde çalıştırdıktan sonra (istatistik toplamaksızın) izleyen 500 milyon komut normal çalıştırılarak istatistikler topladık (Sherwood ve ark., 2001). SPEC2000'deki 26 uygulamadan 22'sini deneylerimizde kullandık. Geri kalan 4 uygulamaya ait binariler elimizde olmadığın onları deneylerde kullanamadık. Tablo 2, P yöntemi kullanıldığında Tablo 1'deki yapılandırma parametreleri için her bir uygulama programına ait iL1 önbellek erişim sayısı ile program yürütme çevrimlerini göstermektedir.

Tablo 2. Deneylerimizde kullandığımız uygulamalar ve onların temel özellikleri

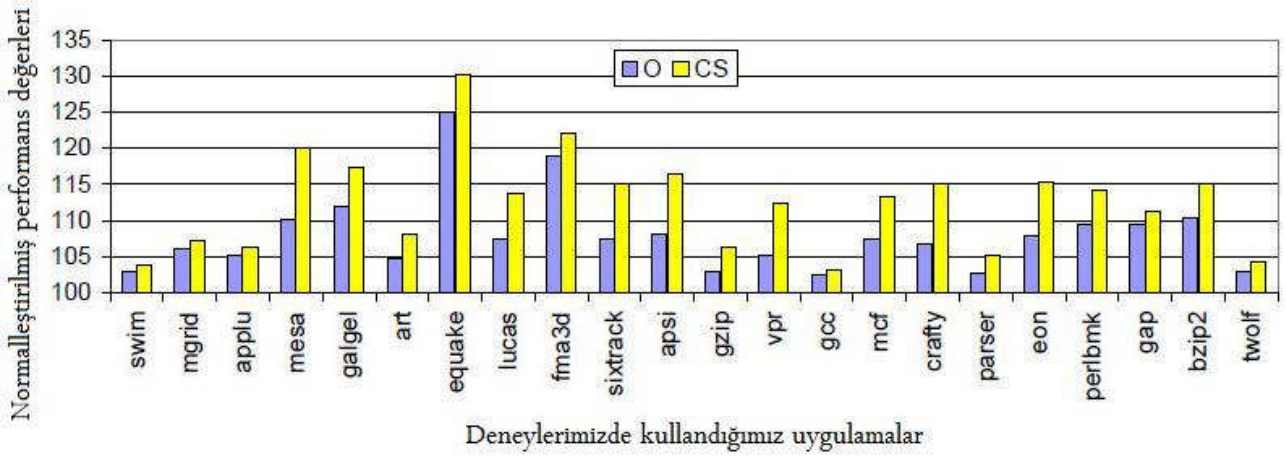
	swim	**mgrid**	**applu**	**mesa**	**galgel**	**art**
iL1 Accesses	347110044	258785124	260821970	120163487	186777092	556587751
Exe. Cycles	347982163	260407090	261959435	160312282	191221037	630700741
	equake	**lucas**	**fma3d**	**sixtrack**	**apsi**	**gzip**
iL1 Accesses	109969817	236223857	109322546	122194786	189621174	167916846
Exe. Cycles	130813719	278925939	180113099	195349559	210554664	272520611
	vpr	**gcc**	**mcf**	**crafty**	**parser**	**eon**
iL1 Accesses	308024576	204429261	616617384	153696081	237277415	140157315
Exe. Cycles	435778113	205505024	643506260	256467119	371959380	257521967
	perlbmk	**gap**	**bzip2**	**twolf**		
iL1 Accesses	186549445	143319970	168465412	404191737		
Exe. Cycles	367955497	212187011	231748913	569302265		

3. 2. Sonuçlar

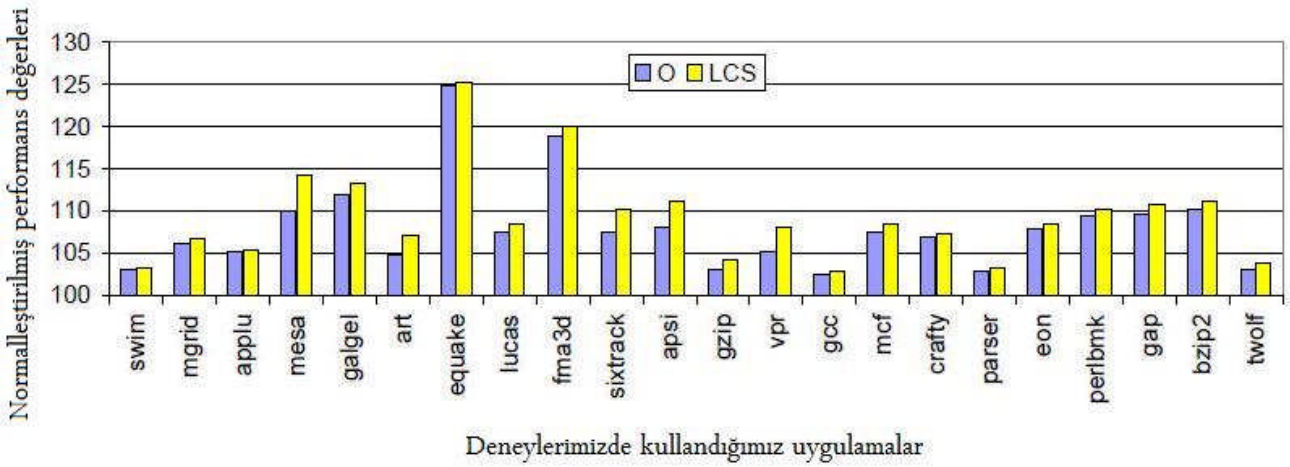
Bu bölümde önerdiğimiz yöntemlere ait deneysel sonuçları vereceğiz. Burada bütün performans sonuçlarımızı P yöntemine ait performans sonuçlarına göre normalleştirilerek vermekteyiz. P yöntemi, ideal durumu temsil edip işlem değişkenliğinin olmadığı durama karşı gelir. Deneysel sonuçlarımızı, önbellek kümelerinin %25'inin işlem değişkenliğinden etkilendiği kabulü altında elde ettik. Bizler March testini (Chen ve ark., 2005) kullanarak sorunlu önbellek kümelerini tespit edebilirdik veya bir modelle (Friedberg ve ark., 2005) önbellekteki sorunlu kümeleri ilişkilendirebilirdik. Çalışmamızın konusu olmadığından bunları yapmadık. Bunun yerine, işlem değişkenliğinden etkilenen önbellek kümelerin önbellek boyunca rast gele dağıldıkları ve işlem değişkenliğine maruz kalan bir kümeye yapılan erişimin okumayı 1 cycle geciktirdiği kabulünü yaptık. Diğer bir ifadeyle sorunlu kümelere yapılan load işlemleri 2 cycle'da tamamlanır. Tamponlanmış store işlemleri

kullandığımızdan, tamponda yer olduğu sürece işlem değişkenliği store işlemi için herhangi bir performans kaybına yol açmaz.

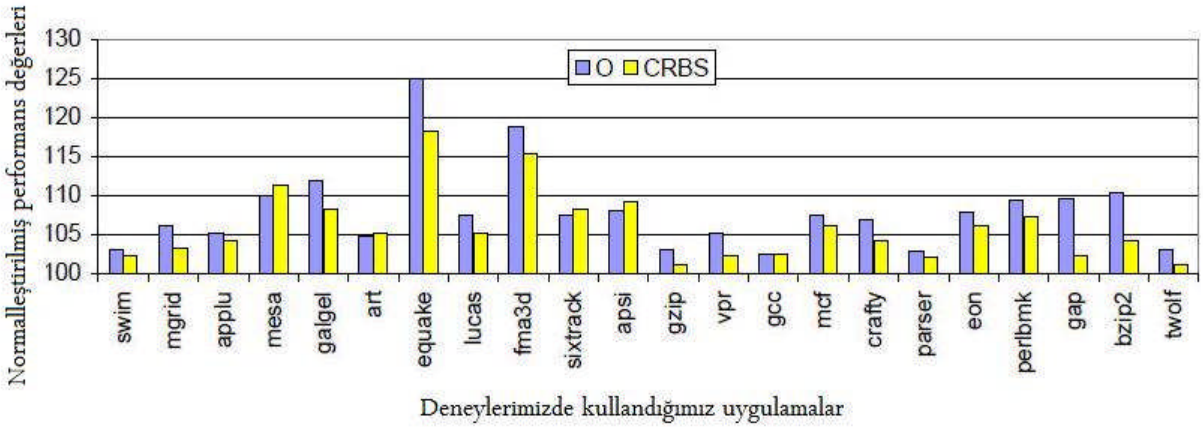
Şekil 3, 4 ve 5'teki sütun grafikleri sırasıyla CS, LCS ve CRBS yöntemlerine ait performans değerlerini göstermektedir. Bu üç grafikte her bir uygulama programı için, ilk çubuk O yöntemine ait performans değerini, ikinci çubuk ise ilgili yönteme ait performans değerini göstermektedir (CS, LCS, veya CRBS). Bütün performans değerleri ise P yöntemine ait performans değerlerine göre normalleştirilerek verilmiştir.



Şekil 3. P yöntemi performans değerlerine göre normalleştirilmiş O ve CS yöntemlerine ait performans değerleri. Şekil 1'de olduğu gibi önbellek kümelerinin %25'inin işlem değişkenliğine maruz kaldığı kabulü yapılmaktadır.



Şekil 4. P yöntemi performans değerlerine göre normalleştirilmiş O ve LCS yöntemlerine ait performans değerleri.

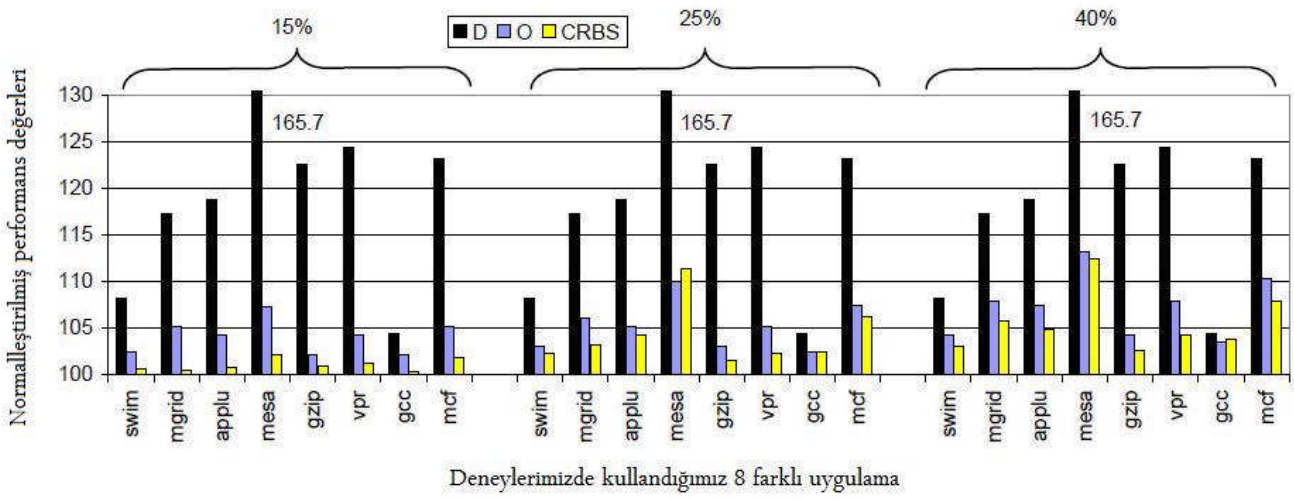


Şekil 5. P yöntemi performans değerlerine göre normalleştirilmiş O ve CRBS yöntemlerine ait performans değerleri.

Yukarıdaki üç grafiğe göre, CS ve LCS yöntemlerine ait ortalama performans kayıpları yaklaşık %12,5 ve %9,2'dir. Öte yandan CRBS yöntemine ait ortalama performans kaybı %5,9 civarındadır. D ve O yöntemlerine ait ortalama performans kayıplarının %30,2 ve %8,1 olduğunu göz önünde bulundurursak aşağıda verilen gözlemlere ulaşabiliriz. Bunlardan ilki, en kötü erişim gecikmesi mantığını kullanan ve %30,2 gibi yüksek oranlı performans kaybına sahip D yöntemi, işlem değişkenliğinin sebep olduğu performans kayıplarını telafi edebilmek için yeni yöntemlerin geliştirilmesini gerekli kılar. İkincisi, dallanma komutlarının hedef adreslerine karşı gelen komutları analiz etmeden bile erişim gecikmelerini komutlarda kodlamak performansı olumlu yönde oldukça çok etkiler ve performans kaybının %12,5'e indirgenmesine yol açar. Üçüncü nokta, LCS yöntemi ile performans kaybı %9,2'ye kadar indirilebilir ve bu değer, O yönteminin %8,1'lik performans kaybına oldukça yakın bir değerdir. Son olarak, erişim gecikme kodlaması ve kod yer değiştirmeyi birlikte uygulayarak, komut önbelleğinin işlem değişkenliğine maruz kalması sonucu ortaya çıkan performans kayıplarını ortalama olarak %5,9'a kadar düşürmek mümkündür. Bu performans, önbellek kümelerinin erişim gecikmelerini önceden yüzde yüzlük doğrulukla tahmin edebilen ve donanımı ona göre senkronize eden O yönteminin sağlayacağı performanstan daha iyidir.

Performansı en iyi olan CRBS yöntemimizin avantajlarını daha iyi belirleyebilmek için, bu yöntemin değişik yüzdeli sorunlu önbellek kümelerine karşı duyarlılığını belirlemeye çalıştık. Bunlarla ilgili deneylerin sonuçları Şekil 6'da verilmektedir.

Bu deneylerde geri kalan uygulamalar için benzer sonuçlar elde ettiğimizden, sunum bakımından kolaylık sağlaması açısından burada sadece 8 farklı uygulama için deneysel sonuçları veriyoruz.



Şekil 6. D, O ve CRBS yöntemlerine ait performans değerleri. Bütün değerler, P yöntemi kullanılarak elde edilen performans değerlerine göre normalleştirilmiştir. İşlem değişkenliğinden etkilenen önbellek kümelerinin yüzdesini %15, %25 ve %40 olarak değiştirdik.

Şekilden de kolayca görülebildiği gibi CRBS yönteminin verimliliği, farklı sorunlu küme oranları için de geçerlidir. İşlem değişkenliğinden etkilenen önbellek kümelerinin oranını %15, %25 ve %40 öngördüğümüzde, O yöntemi için ortalama performans kayıpları sırasıyla %4.1, %5,3 ve %7,3'tür. CRBS yöntemine ait ilgili performans kayıpları ise sırasıyla %1,0, %4,2 ve %5,6'tır. Bu grafikten önemli bir gözlem ise şudur. Eğer sorunlu kümelerin oranı az ise, CRBS yönteminin performansı P yönteminin performansına oldukça yaklaşır. Bu durum şu anlama gelir. CRBS yöntemini uygun şekilde kullanarak işlem değişkenliğinden kaynaklanan performans kayıplarını tamamıyla yok etmek neredeyse mümkün olabilmektedir.

BÖLÜM 4

SONUÇLAR VE ÖNERİLER

Gelişen silikon teknolojisiyle transistor kalitesini istenilen düzeyde tutmak gittikçe zorlaşmaktadır. Bunun sonucu olarak parametre değişkenliği bilgisayar tasarımında önemli bir sorun olarak ortaya çıkmaktadır. Voltaj ve sıcaklık değişkenliklerinin yanında işlem değişkenliği de üzerinde durulması gereken önemli bir parametre değişkenliğini teşkil eder.

SRAM'ler minimum alan kaplayacak şekilde tasarlandıklarından işlemci bileşenleri gibi işlem değişkenliğine maruz kalabilirler. Bunun sonucu olarak işlem değişkenliği, aynı donanım bileşenin farklı kısımları için farklı davranışların ortaya çıkmasına sebep olabilir. Örneğin, önbelleğin bir bloğu başka bir bloğa oranla daha fazla enerji tüketebilir veya daha fazla erişim gecikmesi gerektirebilir.

Bu çalışmamızda komut önbelleğindeki işlem değişkenliğinden kaynaklanan performans kayıplarını azaltmaya yönelik bazı yöntemler tanıttık. En kötü erişim gecikmesini temel alan yöntem (her bir önbellek bloğuna erişimin en yavaş önbellek bloğuna erişim kadar zaman aldığını kabul etmek) tasarım açısından basit olsa da, yaptığımız araştırmalar sonucu bu yöntemin performans açısından sorun teşkil edebileceğini belirledik. Buna alternatif olarak, önbellek kümelerine ait erişim gecikmelerini saklayan bir yöntemi önerebiliriz. Bu tabloyu bir sonraki adımda yürütülecek olan komutun ait olduğu kümenin indeksiyle indeksleyip, elde edilen erişim gecikmesine göre önbelleği senkronize edebiliriz. İlgili tablo işlemcide ekstra alan işgal edeceğinden ve tablonun kendisinin de işlem değişkenliğine maruz kalabileceğinden dolayı bu seçenek uygun görülmemiştir.

Bizim bu çalışmamızda işlem değişkenliğinin sebep olduğu önbellek performansındaki kayıpları azaltmak için derleyiciye dayalı bir takım teknikler tanıttık. Bu tekniklerin ortak özelliği, komutların ait olduğu önbellek kümelerin erişim gecikmelerini komutlarda kodlamalarıdır. Program yürütümü sırasında işlemci aktif olarak pipeline'a giren komutun kodunu çözerken, bir sonraki adımda yürütülecek olan komutun komut önbelleği erişim gecikmesine ait bilgiyi elde eder. Bu bilgiye göre komut önbelleği erişimini senkronize eder. SPEC2000 suit'e ait programlar üzerinde yaptığımız deneyler

derleyiciye dayalı bu yöntemlerin, işlemci değişkenliğinden kaynaklanan performans kayıplarını telafi etmekte oldukça başarılı olduklarını göstermektedir. Ayrıca kod yer değişiminin de performans kayıplarını azaltmada oldukça başarılı olabileceğini gördük. Kod yer değişimimin temel mantığı, hem dallanmayı izleyen komutu hem de dallanmanın hedefindeki komutu aynı erişim özellikli önbellek kümelerine yerleştirmek; sıklıkla yürütülen dallanma komutlarını ve bu komutların hedef adreslerindeki komutları mümkünse sorunsuz önbellek kümeleriyle eşleşecek şekilde kod yer değişimi yapmaktır. Performans açısından en verimli yöntemimiz CRBS yöntemi olup, hem erişim gecikmelerini kodlamaktadır hem de kod yer değişimi işlemini yapabilmektedir. Bu yöntem sayesinde komut önbelleğindeki işlem değişkenliğinden kaynaklanan performans kayıplarını ortalama olarak %30,2'den %5,9'a indirebiliriz.

KAYNAKLAR

SimpleScalar toolset. http://www.simplescalar.com

SPEC 2000 Benchmark. http://www.spec.org

Agarwal A., Blaauw D. ve Zolotov V., 2003. Statistical Timing Analysis For Intra-Die Process Variations With Spatial Correlations. *Int. Conference On Computer Aided Design*. 900–907.

Agarwal A., Paul B. C., Mukhopadhyay S. ve Roy K., 2005. Process Variation In Embedded Memories: Failure Analysis And Variation Aware Architecture. *IEEE Journal of Solid-State Circuits*, 40(9): 1804–1814.

Borkar S., Karnik T., Narendra S., Tschanz J., Keshavarzi A. ve De V., 2003. Parameter Variations And Impact On Circuits And Microarchitecture. *ACM/IEEE Design Automation Conference*, 338–342.

Bowman K., Duvall S. G. ve Meindl J. D., 2002. Impact Of Die-To-Die And Within-Die Parameter Fluctuations On The Maximum Clock Frequency Distribution For Gigascale Integration. *IEEE Journal of Solid-State Circuits*, 37(2): 183–190.

Chen Q., Mahmoodi H., Bhunia S. ve Roy K., 2005. Modeling and Testing Of SRAM For New Failure Mechanisms Due To Process Variations In Nanoscale CMOS. *VLSI Test Symposium*, 292–297.

Chen T. ve Naffziger S., 2003. Comparison Of Adaptive Body Bias (ABB) and Adaptive Supply Voltage (ASV) For Improving Delay And Leakage Under The Presence Of Process Variation. *IEEE Transactions on VLSI Systems*, 11(5): 888–899.

Friedberg P., Cao Y., Cain J., Wang R., Rabaey J. ve Spanos C., 2005. Modeling Within-Die Spatial Correlation Effects For Process-Design Co-Optimization. *Int. Symposium on Quality Electronic Design*, 516–521.

Gregg J. ve Chen T., 2004. Post Silicon Power/Performance Optimization In The Presence Of Process Variation Using İndividual Well Adaptive Body Biasing(IWABB). *Int. Symposium on Quality Electronic Design,* 453–458.

Meng K. ve Joseph R., 2006. *Process* Variation Aware Cache Leakage Management. *Int. Symposium on Low Power Electronics and Design*, 262–267.

Nassif,S. R., 2001. Modeling And Analysis Of Manufacturing Variations. *IEEE Conference on Custom Integrated Circuits*, 223–228.

Özdemir S., Sinha D., Memik G., Adams J. ve Zhou H., 2006. Yield-Aware Cache Architectures. *IEEE/ACM Int. Symposium on Microarchitecture*, 15–25.

Papanikolaou A., Lobmaier F., Wang H., Miranda M. ve Catthoor F., 2005. A Systemlevel Methodology For Fully Compensating Process Variability Impact Of Memory Organizations In Periodic Applications. *IEEE/ACM/IFIP Int. Conference on Hardware/Software Codesign and System Synthesis*, 117–122.

Pour A. F. ve Hill M. D., 1993. Performance Implications Of Tolerating Cache Faults.*IEEE Transactions on Computers*, 42(3): 257–267.

Tschanz J. W., Kao J. T., Narendra S. G., Nair R., Antoniadis D. A., Chandrakasan A. P. ve De V., 2002. Adaptive Body Bias For Reducing Impacts Of Die-To-Die and Within-Die Parameter Variations On Microprocessor Frequency And Leakage. *IEEE Journal of Solid-State Circuits*, 37(11): 1396–1402.

Vergos H. T. ve Nikolos D., 1995. Performance Recovery In Direct-Mapped Faulty Cache Via The Use Of Very Small Fully Associative Spare Cache. *Int. Computer Performance and Dependability Symposium*, 326–332.

Sherwood T., Perelman E. ve Calder B., 2001. Basic Block Distribution Analysis To Find Periodic Behavior And Simulation Points In Applications, In: *International Conference On Parallel Architectures And Compilation Techniques*, , 3 – 14.

TABLOLAR LİSTESİ

Sayfa No

ŞEKİLLER LİSTESİ

Sayfa No

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı : Seher KIZILTEPE

Doğum Yeri : Aksaray

Doğum Tarihi : 19.05.1982

EĞİTİM DURUMU

Lisans Öğrenimi : Çanakkale Onsekiz Mart Üniversitesi Bilgisayar ve Öğretim Teknolojileri Eğitimi Bölümü

Yüksek Lisans Öğrenimi : Çanakkale Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Ana Bilim Dalı

Bildiği Yabancı Diller : İngilizce

BİLİMSEL FAALİYETLERİ

İŞ DENEYİMİ

2009 Yaz - … : Harmancık İlköğretim Okulu Bilgisayar Öğretmeni Harmancık / Bursa

2004 Yaz : Harmancık 75. Yıl Çok Programlı Lisesi Bilgisayar Öğretmeni Harmancık / Bursa

İLETİŞİM

E-Posta Adresi : shrk19@gmail.com