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

ÇOK ÇEKİRDEKLİ SİSTEMLERİN VERİ ÖNBELLEKLERİ İÇİN GEÇİCİ HATALARIN MODELLENMESİ

Selçuk KOYUNCU

Bilgisayar Mühendisliği Anabilim Dalı

Tezin Sunulduğu Tarih: **12.10.2009**

Tez Danışmanı:

Yrd. Doç. Dr. İsmail KADAYIF

ÇANAKKALE

YÜKSEK LİSANS TEZİ SINAV SONUÇ FORMU

SELÇUK KOYUNCU tarafından **YRD. DOÇ. DR. İSMAİL KADAYIF** yönetiminde hazırlanan **"Çok Çekirdekli Sistemlerin Veri Önbellekleri İçin Geçici Hataların Modellenmesi"** başlıklı tez tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Yüksek Lisans tezi olarak kabul edilmiştir.

Yrd. Doç. Dr. İbrahim Türkyılmaz

Yönetici

Yrd. Doç. Dr. İsmail Kadayıf

Jüri Üyesi

Yrd. Doç. Dr. Can Aktaş

Jüri Üyesi

Sıra No: **461**

Tez Savunma Tarihi: 12/10/2009

Prof.Dr.Ahmet ERDEM

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.

TEŞEKKÜR

Sadece tezimin hazırlanması sürecinde değil her zaman her konudaki destek, yardım ve yönlendirmeleri için en başta değerli hocam Yrd. Doç. Dr. İsmail KADAYIF' a sonsuz teşekkürlerimi ve her zaman yanımda olan kıymetli babam Yaşar KOYUNCU, annem Perihan KOYUNCU ve biricik ablam Esra Gül KOYUNCU'ya, simülasyon testleri sırasında yardımcı olan değerli arkadaşım Musatafa ŞAHİN ve Wim HEIRMAN'a, sürekli moral ve motivasyonları ile destek veren arkadaşlarım Erhan TAŞKIN, Hande ŞEN, Bora UĞURLU ve yardımlarını hiçbir zaman esirgemeyen aziz dostum Mahir TÜRKCAN, Kirami KAÇAN ve Nihal YILDIRIM'a sonsuz teşekkürlerimi sunuyor ve bu çalışmamı onlara ithaf ediyorum.

Selçuk KOYUNCU

SİMGELER VE KISALTMALAR

API (Application Programming Interface): Uygula Programlama Ara Yüzü

AVF (Architectural Vulnerability Factor): Mimarisel Maruz Kalma Faktörü

CFG (Control Flow Graph): Kontrol Akış Diyagramı

CMP (Chip MultiProcessor): Çok İşlemcili Çip

DFG (Data Flow Graph): Veri Akış Diyagramı

DP (Dead Period): Ölü Period

ILP (Instruction Level Paralelism): Komut Seviyesi Paralellik

IP (Incubation Period): Kuluçka Dönemi

ISA (Instruction Set Architecture): Komut Küme Mimarisi

NRM (N Modular Redundancy): N Modüler Çoğullama

SECDED (Single Error Correct Double Error Detect): Tek Hata Düzeltme Çift Hata Tespiti

SER (Soft Error Rate): Geçici Hata Sıklığı

SMP (Shared Memory Processor): Paylaşımlı Bellek Sistemli İşlemci

SOI (SilicOne Insulator): Silikon Yalıtıcı

TMR (Triple Modular Redundancy): Üçlü Modüler Çoğullama

ÖZET

ÇOK ÇEKİRDEKLİ SİSTEMLERİN VERİ ÖNBELLEKLERİ İÇİN GEÇİCİ HATALARIN MODELLENMESİ

Selçuk KOYUNCU
Çanakkale Onsekiz Mart Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı Yüksek Lisans Tezi
Danışman: Yrd. Doç. Dr. İsmail KADAYIF
12.10.2009, 45

Son zamanlarda yüksek frekanslarda çalışan güçlü, karmaşık, yüksek güç tüketimi olan tek bir superscalar işlemci çekirdeğinden oluşan çip tasarımlarından basit, daha az enerji gereksinimi olan ve daha düşük frekanslarda çalışan, içerisinde birden çok işlemci çekirdeği barındıran çok işlemcili çip (chip multiprocessor, CMP) tasarımlarına doğru bir değişim sürecine tanıklık etmekteyiz. Bu değişimin temel etkeni, neredeyse fiziksel limitlere yaklaşmış olmamız (daha küçük boyutlu transistor tasarımının ve güç tüketimi kontrolünün çok daha zor olması) ve çoğu uygulamanın daha yüksek komut seviyesi paralelliğe olanak tanımamasından dolayı 4'ten büyük genişlikli işlemci tasarımının çok az ek performans artışına sebep olmasıdır. Ayrıca ağ sunucuları, veritabanları ve paralel bilimsel kodlar gibi çok çekirdekli bilgisayarlara ihtiyaç duyan çok iş parçacıklı uygulamalar gittikçe daha da yaygınlaşmaktadır.

Diğer yandan kozmik ışınlardaki nötron ve paketleme materyallerinden yayılan alfa taneciklerinin yarı iletken devrelere çarpmaları sonucu oluşan geçici hatalar mikroişlemcilerin güvenirli şekilde çalışmalarında önemli bir problem teşkil ederler. Bu çalışmada bir CMP sistemde birincil seviye veri önbelleğinde oluşan geçici hataları modellemeye çalışmaktayız. Bu modele göre önbellek sistemine ait mimarisel maruz kalma faktörünü (Architectural Vulnerability Factor, AVF) hesapladık. Bir bileşen için AVF, bu bileşende oluşan bir hatanın kendini programın çıktısında hangi ihtimalle hissettirebilme olarak tanımlanabilir. Tek çekirdekli işlemcilerdeki değişik işlemci bileşenleri için AVF'nin hesaplanması ve geçici hataların modellenmesi konusunda birçok değişik çalışmalar olmasına rağmen, CMP sistemleri için bu tip çalışmalar oldukça azdır.

Bizim modelimiz hem geçici hata oluşumunu ve hem de yayılımını modelleyebilmektedir. Kritik hataları farklı kategorilere ayırmak suretiyle veri önbellek sistemi için AVF'yi hesaplamaya çalışmaktayız. Deneysel sonuçlarımız toplam AVF içerisindeki en büyük payın L1 önbelleklerinden paylaşımlı L2 önbelleğe yayılan geçici hataların olduğunu göstermektedir.

Anahtar Sözcükler: Geçici Hatalar, Güvenirlik, Veri Önbellekleri, Mimarisel Maruz Kalma Faktörü.

ABSTRACT

MODELING SOFT ERRORS FOR DATA CACHES IN CHIP MULTIPROCESSOR SYSTEMS

Selçuk KOYUNCU
Canakkale Onsekiz Mart University
Graduate School of Science and Engineering
Thesis of Master of Science for Chair of Computer Engineering
Advisor: Ph. D. Ismail Kadayif
12.10.2009, 45

Recently we have been witnessing a dramatic shift in processor design from a very powerful, more complex, and more-power-hungry single superscalar cores running at very high clock speeds to chip multiprocessor (CMP) systems with multiple cores in the same chip with each less powerful, simpler, consuming less power, and operating at lower clock speeds. The force behind this shift is that we almost approach the physical hard limits (designing finer transistors and controlling power consumption are becoming more challenging) and that increasing issue rate beyond four brings little additional benefit in performance since most of applications do not support larger instruction level parallelism (ILP). Also, multi-core demanding multithreaded workloads such as web servers, databases, and parallel scientific codes have become more widespread.

On the other hand, radiation-induced soft errors introduced by particle strikes such as neutron particles in cosmic rays and alpha particles from packaging materials pose a significant problem for microprocessors to run in a reliable manner. In this study, we try to model the soft errors in first level data caches in a CMP system based on which we calculate the architectural vulnerable factor (AVF) for the first level data cache system. The AVF for a component is described as the probability with which a fault in this component can reveal itself in the final output of the program. Although there are a numerous number of previous studies related to the calculation of AVF and modelling the soft errors for different components in single core processors, there is dearth of such studies for CMPs. Our model models the soft error generation and propagation as well. We divide up the critical errors into different categories, based on which we calculate the AVF.

Our experimental results show that the soft errors spreading from private L1 caches into the shared L2 cache is a major contributor of the overall AVF.

Keywords : Soft Errors, Reliability, Data Caches, Architectural Vulnerability Factor.

İÇERİK

Sayfa

BÖLÜM 1

GİRİŞ

Geçtiğimiz son yirmi yıl içerisinde mikroişlemci tasarımcıları yüksek performanslı işlemcilerin performanslarını artırmak için temel iki fikir üzerinde durdular: yüksek işlemci frekansı ve komut seviyesi paralellik (Instruction Level Paralelism, ILP). Geliştirilen her yeni silikon teknolojisi, küçülen devrelerin daha yüksek hızlı işlemci frekanslarında anahtarlanabilmesini sağladı. Tasarımcıların tasarımlarında yüksek hızlı frekans kullanmalarına yardımcı olan bir diğer önemli unsur da pipelining tekniğinin kullanılması oldu. İşlemci komut yürütümünün eşit boyda küçük adımlara bölünmesi ve bu şekildeki işlem adımlarının hızlı işlemci frekanslarında çalıştırılabilir olması pipelining tekniğinin başarılı olmasında temel unsur oldu. Diğer taraftan, transistor boyutlarındaki küçülme aynı çekirdeğin içerisine eskisine göre daha fazla transistor sığdırılmasına olanak tanıdı. Transistor sayısındaki bu artış performansın artırılmasına yönelik yeni tekniklerin uygulanmasında kullanıldı. İşlemci komutlarının dinamik olarak sıralanması, çok sayıdaki işlemci komutlarının aynı anda çalışma ünitelerine gönderilmeleri (multiple instruction issue), dallanma tahminleri, yazmaçların yeniden isimlendirilmesi (register renaming), sıra dışı çalışma (out of order execution) gibi birçok teknik uygulama programlarında komut seviyesindeki paralellikten yararlanmak için modern işlemci donanımlarında başarılı bir şekilde kullanılmıştır.

Ancak limitlerinin neredeyse sonuna geldiğimiz tek çekirdekli işlemcilerde, tasarımcılar bütün bu teknikleri kullanarak işlemci performansını daha fazla artıramamaktadırlar. Örneğin, daha küçük boyutlu transistorları tasarlamak teknolojik limitlerden dolayı gittikçe daha zor hale gelmektedir. Çoğu uygulamalar daha yüksek ILP'yi desteklemediğinden dörtten fazla sayıdaki işlemci komutunun aynı anda çalışma ünitelerine gönderilmesi performans artışında çok küçük iyileşmeye neden olmaktadır. İşin kötü yanı, daha yüksek ILP'ye olanak tanıyan mikroişlemci çekirdeği tasarımı işlemci karmaşıklığını genişliğin karesi oranında artırmakta ve donanımın doğruluğunun tespitini çok zor ve çok pahalı hale getirmektedir. Bu yeniliklerin güç tüketimini üstel olarak artırması işleri daha da kötü hale sokmaktadır.

Sonuç olarak, son zamanlarda, yüksek hızlı frekanslarda çalışan güçlü, karmaşık, yüksek güç tüketimi olan tek bir superscalar işlemci çekirdeğinden oluşan çip tasarımlarından basit, daha az enerji gereksinimi olan ve daha düşük frekanslarda çalışan, içerisinde birden çok işlemci çekirdeği barındıran çip (chip multiprocessor, CMP) tasarımlarına doğru bir değişim sürecine tanıklık etmekteyiz (Barroso ve ark., 2000; Kongetira ve ark., 2005). Bu durumun nedenleri sadece yukarda saydığımız teknolojik limitlerin sonuna yaklaşılması değil, bununla birlikte yeni yazılımların ihtiyaç duyduğu gereksinimlerdeki değişimlerdir. İnternetin ortaya çıkması ile birlikte ILP açısından zayıf olan sunucu uygulamaları giderek daha fazla yaygınlaştı. Az sayıda basit istemci isteklerinin çok hızlı cevaplanması yerine çok sayıdaki istemcinin desteklenmesini gerektiren bu uygulamalar için çok çekirdekli işlemcilerden oluşan çipler daha uygundur. İstemci tarafından sunucuya gönderilen ve ILP açısından fakir olan bu isteklerin her birinin farklı bir işlemciye atanması, işlem paralelliğini kullanmaya olanak tanımaktadır.

BÖLÜM 2

ÖNCEKİ ÇALIŞMALAR

Kozmik ışınlardaki nötron parçacıklarının ve paketleme malzemelerinden yayılan alfa parçacıklarının (Ziegler, 1996; Irom ve ark., 2002) devrelere çarpmaları sonucu ortaya çıkan geçici hatalar (soft errors) mikroişlemcilerin güvenilir bir şekilde çalışmalarına önemli bir tehdit oluştururlar (Pradhan, 2003). Çip içerisinde bulunan yapıların tamamı bu hatalardan etkilenebiliyor olsa da ön bellekler çip içerisinde kapladığı büyük alandan dolayı bu anlamda daha fazla tehdit altındadırlar. Dahası, çip içerisindeki ön belleklerin CPU'ya olan yakınlığından dolayı burada meydana gelecek bir hata kolaylıkla sistemin veri yoluna yayılabilir ve yanlış sonuç üretilmesi, hatta sistemin arıza vererek beklenmedik şekilde çalışmayı tamamen durdurmasına yol açabilmektedir. Bu konuda daha önceki bazı çalışmaların ışığında çip üzerindeki bileşenlere (on-chip components) ait latch ve SRAM hücreleri gibi yapılarda oluşan hataların oranları teknolojik gelişimlerle birlikte aşağı yukarı aynı kalabildiği yada biraz azaltılabildiği görülmüştür (Hareland ve ark., 2001; Karnik ve ark., 2001). Bunun anlamı, işlemci tasarımcıları bu soruna "silikon işleme seviyesinde" bir çözüm bulamazlarsa SRAM önbellekler için SER'in (Soft Error Rate) (Nguyen ve Yagil, 2003) kullanılan transistor sayısındaki artış ile paralellik göstereceğidir.

Geçici hataların tespit edilip güvenilirliğin artırılması işlemcilerin doğru sonuç üretmeleri için son derece önemlidir (Gomaa ve Vijaykumar, 2005; Reinhardt ve Mukherjee, 2000). Güvenirliği artırmaya yönelik teknikler genel olarak üç kategoride incelenebilir; bunlar: süreç teknolojisi, devre ve mimari seviyelerdeki çözümlerdir (Mukherjee ve ark., 2005). Bu kategorilerden ilk ikisi için kullanılan teknikler etkin olmakla birlikte bu teknikler güç, performans ve kapladıkları ek alan itibari ile olumsuzluklara neden olmaktadır. Örneğin IBM, süreç teknolojisi tabanlı bir çözüm olarak SOI (silicon-insulator) kullanılarak SRAM aygıtları içerisindeki SER'i 5'te bire düşürmenin mümkün olduğunu belirtmiştir (Cannon ve ark., 2004). Devre temelli çözümler, hücre kapasitansı ve besleme voltajı gibi cihaz parametrelerini ayarlamak suretiyle hücre çıkışının bit değişikliğine uğrayabilmesi için gereken minimum yük miktarının artırılması yöntemine dayanmaktadır (Calin ve ark., 1996).

Mimari çözümler açısından baktığımızda farklı derecelerdeki güvenilirliğin farklı maliyetlerle sağlandığı görülmektedir. Byte-parity mekanizması her 8 bit veri için ekstra 1 şlik biti kullanmak sureti ile tek sayılı hataları yakalamayı sağlar. Bu yöntem, güç tüketimi açısından oldukça iyi olmasına rağmen %12,5'lik ek alan maliyetine neden olmaktadır. Hata düzeltilmesinde popüler bir yöntem olan Single Error Correct Double Error Detect (SECDED) (Chen ve Hsia, 1992) eşlik bitiyle karşılaştırıldığında daha fazla güvenirlik sağlamakla birlikte (tekli hataları düzeltebilir) gerçekleştirilmesi daha zordur ve hata kontrolü için daha fazla zamana ihtiyaç duymaktadır. Pipeline tabanlı olmayan önbelleklerde kullanıldığında önbelleğe erişim süresini ve güç tüketimini artırabilir. Veri güvenirliğini artırmakta kullanılan diğer bir teknik ise N Modular Redundancy (NMR) tekniğidir. Bu yöntem ile veri güvenirliği oldukça fazla alan külfeti karşılığında çoğunluk oylaması tekniğiyle sağlanır. Örneğin, Triple Modular Redundancy (TMR) yönteminin alan masrafı %200'dür (Carmichael, 2006). Bu durum alan bakımından kısıtlı olan sistemler için oldukça pahalıdır. Bunlara ek olarak redundant multi-threadinge dayalı veri güvenirliğinin artırılmasına ilişkin bir çok yöntemler önerilmiştir (Gomaa ve ark., 2003; Ray ve ark., 2001; Vijaykumar ve ark., 2002). Bu yöntemlerde aynı komut dizisinin birden fazla kopyasının çalıştırılması ile hata tespiti ve düzeltilmesi yapılmaktadır. Bu durum işlemci kaynakları üzerinde yoğun bir baskıya neden olmaktadır. Kumar ve Aggarwal (2006) çalışmalarında bu kaynak gereksinimini azaltarak sistem performansını artırmaya çalışmaktadırlar. Bu çalışmamızda ilk önce bir CMP sisteminde birinci seviyedeki veri önbelleklerindeki geçici hataları modellemeye çalışmaktayız. Her bir çekirdeğin L1 veri önbelleği ve sistem geneli için Architectural Vulnerability Factor (AVF)'ü hesaplamayı hedefliyoruz. Bir bilgisayar bileşeni için AVF (Mukherjee ve Weaver, 2003), bileşen içerisindeki bir hatanın program çıktısında kendini hissettirebilme olasılığı olarak tanımlanmaktadır. Tek çekirdekli işlemciler üzerindeki farklı bileşenlerin AVF'lerinin hesaplanması ve geçici hatalarının modellenmesi üzerine şimdiye kadar çok sayıda çalışma yapılmış olmakla birlikte (Kim ve Somani, 2002; Wang ve Patel, 2003; Wang ve ark., 2004; Karnik ve ark., 2004; Biswas ve ark., 2005; Degalahal ve ark., 2005; Kadayif ve Kandemir, 2007) bugün için CMP' ler üzerinde bu gibi çalışmalar oldukça yetersizdir. Tek çekirdekten oluşan işlemcilerde olduğu gibi, CMP'lerin çekirdeklerindeki farklı bileşenler içinde AVF değerlerinin belirlenmesi iki nedenden ötürü önem arz etmektedir. Birincisi, değişik bileşenlerin AVF'lerinin belirlenmesi sistem içerisindeki hataya en fazla maruz kalabilen, korunmasız bileşenlerin tespitinde bize yardım etmektedir. İkincisi, farklı hata

koruma mekanizmalarının verimliliklerinin değerlendirilmesinde bizlere yardım eder ve onlar içerisinden hedeflenen amaç doğrultusunda bozulabilirliği yüksek olan bileşenlerin geçici hatalara karşı korunmasında en uygun yöntemi seçmemize imkân verir. Modelimiz, bazı hataların program çıktısına tesir etmemesi gerçeğine dayanmaktadır. Örneğin, aynı veri üzerine art arda gelen iki yazma işlemi arasında meydana gelecek bir hata (write-after-write), program çıktısı üzerinde herhangi bir mantıksal değişliğe yol açmaz. Modelimizde önbellek bloklarında hataların oluşumunun yanı sıra, yayılım neticesinde hataların diğer önbellek bloklarındaki verileri etkileyişini de ele aldık. Hatanın nerede oluştuğuna bağlı olarak, hata yayılımı yerel (local) veya global olabilmektedir. Yerel olması, hatanın aynı işlemci çekirdeği üzerindeki veri yoluna ya da aynı önbellek üzerindeki farklı bir bloğa yayıldığını; genel olması ise hatanın L2 önbelleğine veya farklı bir işlemci çekirdeği üzerindeki önbelleğe yayıldığını ifade etmektedir. Nerede oluştukları ve nereye yayıldıkları göz önünde bulundurulmak suretiyle kritik geçici hataları üç kategoriye ayırdık. Burada kritik hata kavramını programın sonucuna tesir edebilen hata olarak kullanmaktayız. Intra-L1 hatalar, oluştukları aynı çekirdeğin L1 veri önbelleğine yayılan hatalardır. Inter-L1 hatalar, herhangi bir çekirdeğin L1 veri önbelleğinde oluşup farklı bir çekirdeğin L1 önbelleğine yayılan hatalardır. Inter-L2 hatalar ise herhangi bir çekirdeğin L1 önbelleğinde ortaya çıkıp direkt olarak paylaşımlı L2 önbelleğe yayılan hatalardır. Sistem L1 veri önbelleğine ait AVF, bu üç kategorideki hatalar göz önünde bulundurularak hesaplanılmaya çalışıldı. Biz, deneylerimizde simülatör olarak Multifacet General Execution-Driven Multiprocessor Simulator Toolset'ini (GEMS, 2009) kullandık. GEMS genel olarak veritabanı ve web sunucuları gibi çok iş parçacıklı uygulamaların ihtiyaç duyduğu çok işlemcili donanımların performansını incelemek ve karakterize etmek için oluşturulmuş bir simülatörler kümesidir (Martin ve ark., 2005). Biz bu simülatör kümesinden iki tane simülatör kullandık. Sistemdeki çekirdeklerin (CPU) simülasyonu için Opal'i kullandık. Opal, oldukça detaylı bir zamanlama modeli kullanarak günümüzdeki modern super-scalar işlemcileri SPARC ISA (Instruction Set Architecture; Komut Küme Mimarisi) altında simüle eder. CMP'ne ait bellek sisteminin zamanlama simülatörü olarak Ruby'i kullandık. Ruby önbellekleri, önbellek kontrolörleri, sistem içi haberleşme ağı, ana bellek bölmeleri ve ana bellek kontrolörlerini modeller. Deneylerimizde SPEC OMP2001 (SPEC, 2009) benchmarklarından 8 tane kullandık. SPEC OMP2001 paylaşımlı bellek sistemli işlemcilerin Shared Memory Processor, (SMP) performansını ölçen programları içermektedir

BÖLÜM 3

MATERYAL VE YÖNTEM

3.1. CMP Önbellekler İçin Soft Error Modeli

C CMP'lerde önbelleklere yönelik geçici hata modelimizin detaylarına girmeden önce modelimizin tanıtımına yardımcı olan temel terimlerden bahsedelim. Her bir önbellek satırının bir dizi sözcükten (words) oluştuğunu varsayıyoruz. Bu çalışmamızda önbellek bloğu ve önbellek satırını aynı manada kullanmaktayız. Burada kelime terimini sabit olarak 32 bitten oluşan bir kayıt alanı yerine, boyutu değişebilen ve verilerin kayıt edildiği alanlar anlamında kullanmaktayız. Her bir temel veri (C programlama dilindeki 32-bit integer, 64-bit double veya 8-bit char gibi) programın çalışması esnasında önbelleğe getirildikten sonra ilgili satır içerisindeki kelimelerden birisinin içerisinde tutulmaktadır.

3.1.1. Kelime İşlemleri

Yaşamları boyunca önbelleklerdeki satırların her bir kelimesi üzerinde bir dizi işlemler meydana gelmektedir. Bir kelimenin yaşamı, kendisini taşıyan satırın önbelleğe getirilmesi işlemi ile başlar, bu satırın geçersiz addedilmesi veya başka satırın bu satırın üzerine yazılması ile oluşan yer değiştirme işlemi ile son bulur. Bir kelime üzerindeki işlemler birbirlerini takip ederler. Aynı kelime üzerinde oluşan, birbirini takip eden iki işlemden bir önceki işlemle o anki işlem arasında geçen süre, o anki işlemin kuluçka dönemi (Incubation Period, IP) olarak adlandırılır. Temel işlemler ve bunların genel karakteristikleri Çizelge 1'de gösterilmektedir. Kelimeler üzerinde geçersiz kılma (invalidation), doldurma (filling), okuma (read) ve yazma (write) gibi temel işlemler gerçekleştirilmekle birlikte, bir kelime üzerinde o an yapılacak olan işlemin etkisi o kelime üzerinde yapılan bir önceki işleme dayanabileceğinden dolayı işlemlerin sayısını bu bağımlılığa bakarak artırdık. Böylelikle, bir işlemin ismini oluştururken, bir önceki işlemi de göz önüne almaktayız. Bu işlemlerden çoğunun CMP ve tek çekirdekli işlemcilerde ortak olmasına rağmen bazıları sadece CMP'lere mahsustur. L1'e veri yüklemeyi gerektiren (örneğin, fill-from-L2-after-fill-from-L1 ve fill-from-L1-after-fill-from-L2) veya geçersiz kılma işlemleri (örneğin, invalid-after-send) sadece CMP sistemlerinde mevcuttur. L1'den veri yükleme işleminden kasıt verinin en güncel şeklinin uzak

çekirdeklerden birinin yerel önbelleğinde olduğu ve buradan yerel belleğe verinin getirilmesidir. Geçersiz kılmak, paylaşılan verinin uzak çekirdeklerden birinin önbelleğinde değiştirildiğini, dolayısıyla veriyi tutarlı (coherent) tutmak için yerel kopyasının ilgili önbellekte geçersiz kılınması gerektiğini ifade eder.

Çizelge 1. Kelime üzerinde yapılan temel işlemler ve bu işlemlerin karakteristik özellikleri

İşlemler	İyi Huylu	Kritik		
		Intra-L1	Inter-L1	Inter-L2
invalid-after-send	X			
Fill-from-L2-after-invalid	X			
Fill-from-L2-after-fill-from-L2	X			
Fill-from-L2-after-fill-from-L1	X			
Fill-from-L2-after-read	X			
Fill-from-L2-after-evict	X			
Fill-from-L1-after-invalid			X	
Fill-from-L1-after-fill-from-L2			X	
Fill-from-L1-after-fill-from-L1			X	
Fill-from-L1-after-evict			X	
Fill-from-L1-after-read			X	
evict-after-fill-from-L2	X			X
evict-after-fill-from-L1	X			X
evict-after-read	X			X
evict-after-write				X
read-after-fill-from-L2		X		
read-after-fill-from-L1		X		
read-after-read		X		
read-after-write		X		
write-after-fill-from-L2	X			
write-after-fill-from-L1	X			
write-after-read	X			
write-after-write	X			

3.1.2. Hata Oluşumu

Eğer bir önbellek satırı içerisindeki kelimeyi oluşturan devreye yeteri kadar yüksek enerjili bir parçacık çarparsa ve devrede toplanan yük veriyi depolayan devrelerin çıkışlarını değiştirebilecek miktarda olursa, bu kelime geçici hataya maruz kalmış veya kelimede hata oluşmuş denir. Geçici hatalar işlemlerin IP'leri süresince oluşur. Bizim hata modelimizdeki kabullerden biri, önbelleklerdeki ham hataların sabit bir oranda görüldüğüdür (Li ve ark., 2005). Ayrıca bu hataların işlemci çekirdeklerinin önbellek satırlarında rastgele ve düzgün bir dağılım gösterdiğini düşünmekteyiz (Mukherjee ve ark., 2003). Bu yüzden bu iki kabul altında, bir kelimede geçici hata oluşma olasılığının veri büyüklüğüyle ve verinin parçacık çarpmalarına maruz kaldığı zaman aralığıyla (IP) doğru orantılı olduğu sonucunu çıkarabiliriz. Bunu ifade etmek için w kelimesi için $EGE_{t1,t2}(w)$ hata oluşum olayını (error generation event) tanımlamaktayız. Bu, t1 zamanından başlayarak t2 zamanına kadar geçen sürede (t1 kelime üzerinde gerçekleşen bir önceki işlemin, t2 ise o anki işlemin zamanıdır; bu yüzden t1-t2 işlemin IP sini belirler) w kelimesinde geçici hata oluşumunu gösterir. $|EGE_{t1,t2}(w)|$ ise hatanın skalar değerini ifade etmekte olup, büyüklüğü $|EGE_{t1,t2}(w)| = boyut(w) \times (t2\text{-}t1)$ ile belirlenir.

Çalışmamızda geçici hataları ikiye ayırmaktayız. Birinci grupta yer alan hatalar meydana geldikleri önbelleğin içerisinde izoledirler ve başka bir sistem bileşenine yayılmamaktadırlar. Bu hatalardan etkilenen veriler, programın kontrol akışına etki eden veya bellek hiyerarşisindeki diğer bellek bileşenlerine yazılan verilerin hesaplanmasında yer almazlar. Dolayısıyla bu veriler programın doğru çalışmasında herhangi bir sorun teşkil etmezler. Bu tip hataları "iyi huylu" olarak atfetmekteyiz. Örneğin, fill-from-L2-after-read işleminin IP'si boyunca toplanan hatalar bu türdendir. Bir kelimeye read-after-read işlemi uygulanmışsa ve ilgili okunan veri de ölü bir işlemci komutunu veya yanlış tahmin sonucu çalıştırılmış işlemci komutlarını besliyorsa, bu durumda da işlemin IP'si süresince toplanan hatalar iyi huylu hatalardır. Diğer gruptakiler ise kritik hatalardır. Bu hatalar, bazı hesaplamalar sonrası sistemin diğer bileşenlerine yayılmaktadırlar ve nihayetinde bazı hatalı verilerin sistemde bazı bellek bileşenlerine yazılmasına sebep olmaktadır. Bir kelime üzerinde gerçekleşen read-after-read işleminin IP'si boyunca toplanan hatalar eğer ilgili hatalı veri, bir önbellek bileşenine yazılan başka bir verinin hesaplanmasında yer alırsa bu kategorideki hatalara bir örnek teşkil eder. Bir işlemi iyi huylu veya kritik olarak atfetmek

için IP'si boyunca topladığı hataların tipi göz önünde bulundurulur. Enteresan bir şekilde bazı işlemler, önbellek satırlarının önbellekten çıkarılması zamanındaki durumuna göre iyi huylu veya kritik olabilir. Örneğin, satır kirliyse evict-after-fill-from-L2 işlemi kritiktir. Aksi takdirde ise bu işlem iyi huyludur.

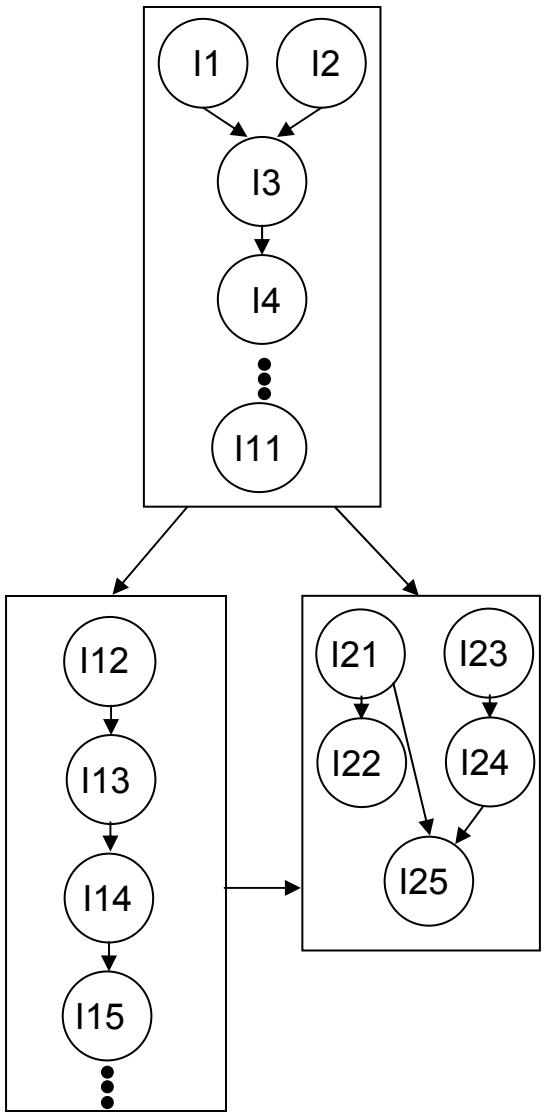
3.1.3. Hata Yayılımı

Programın çalışması sırasında, önbellek satırında bulunan kelimeler hata toplayabilir ve bu durumda bu kelimelerin tuttukları verilerin değeri bozulur. Bu bozulmuş değerler programın doğru çalışmasına engel olabilir. Hata yayılımı mekanizmamızın temel amaçlarından biri kritik hataları iyi huylu hatalardan ayrılmasını sağlamaktır (bir başka deyişle iyi huylu işlemlerin kötü huylu işlemlerden ayırt etmektir). fill-from-L2-after-read gibi bazı işlemlerin iyi huylu olup olmadığına meydana geldikleri anda karar verilir. Ancak bazı işlemlerin durumları meydana geldikleri anda dahi hesaplanamayabilinir. Bu tip hataların yayılımını takip ederek ilgili işlem hakkındaki kararı bozulmuş verinin hangi tip komutlara girdi sağladığının tespit edilmesine kadar erteleriz. Bu karar ancak hatalı verileri kullanan işlemci komutlarının özellikleri belli olunca verilebilir. Örneğin hatalı veri ölü bir komuta veya yanlış yol üzerindeki bir store komutuna girdi teşkil ediyorsa hatanın iyi huylu olduğuna karar veririz ve bizim modelimiz hata bütçesinin hesaplanmasında bu hatayı dikkate almaz. Diğer taraftan hatalı veri kullanılarak yapılan bir hesaplama sonucunda yeni bir verinin elde edilmesi ve devamında bu yeni verinin bir önbellek satırına kayıt edilmesi gibi bir durumda bu tür hatalar kritik hatalardır.

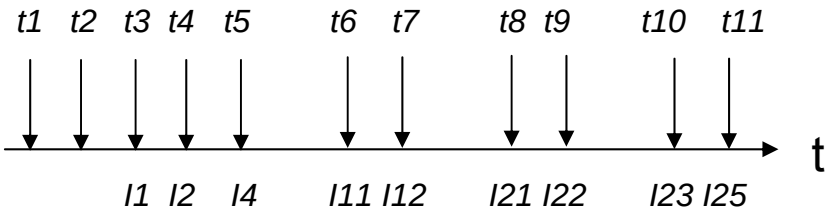
Hata yayılımını takip etmek için değiştirilmiş her bir w kelimesi için t zamanına kadar geçen toplam süre içerisinde bu w kelimesi üzerinde toplanan hataların kümesini $EPS_t(w)$ ile temsil etmekteyiz. Bir kelimenin EPS'si o kelimeyi barındıran önbellek satırı geçerli olduğu müddetçe muhafaza edilir. Bir kelime okunduktan sonra hatanın bu kelimenin bulunduğu yerde meydana geldiğini varsaymaktayız ve bu hatanın hedef kelimeye doğru yayıldığını düşünmekteyiz. Dolayısıyla bu hata olayını hedef kelimenin EPS'ne ekleriz. İki kelime arasındaki hedef kaynak ilişkisi veri akış diyagramındaki mimari yazmaçların izlenmesi ile belirlenir.

```
b=a+b;
.
.
.
if(a<>0) then {
e=b;
b=b+1;
.
.
.
}
c=b;
b=d+1



load R1, <a>     // I1
load R2, <b>     // I2
add R2, R1, R2   // I3
str R2, <b>      // I4
.
.
.
beq R1, target   // I11
load R2, <b>     // I12
str R2, <e>      // I13
add R2, R2, 1    // I14
str R2, <b>      // I15
.
.
.
targ: load R3, <b> // I21
str R3, <c>        // I22
load R1, <d>       // I23
add R1, R1, 1      // I24
str R1, <b>        // I25
```

Şekil 1. (Üstte) Kaynak kod parçası ve (altta) karşılık gelen assembly dil kodu.



Şekil 2. Şekil 1'deki kod parçasının CFG ve DFG si.



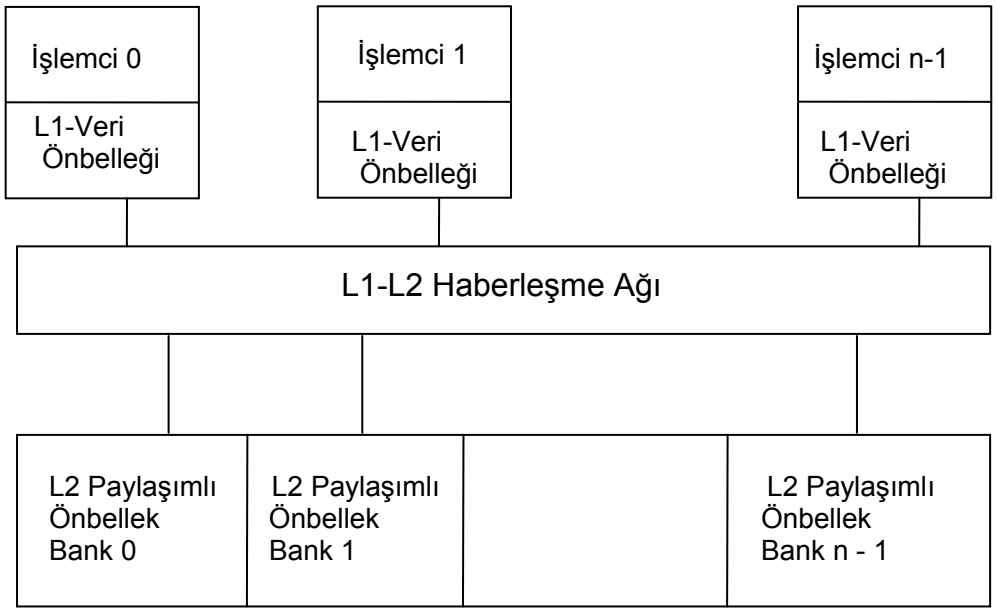
Şekil 3. Şekil 1'deki kod parçasının veri işleyişi için önbelleğe erişimini gösteren zaman çizelgesi. Load ve store lar için önbellek erişim zamanları gösterilmektedir.

Hata yayılımı mekanizmasını daha iyi anlamak için Şekil 1'de gösterilen örnek bir kaynak kod parçasına ve bu kaynak koda karşı gelen assembly koduna bakalım. Şekil 2'de bu kodun veri akış diyagramı (Data Flow Graph, DFG) ve kontrol akış diyagramı (Control Flow Graph) verilmektedir. Şekil 3'te ise kod parçacığı tarafından işlenen verilere ait zaman çizelgesi gösterilmektedir. Burada biz *a*, *b*, *c* verilerinin L2'den L1 önbelleğe *t1* anında getirilen aynı önbellek satırında saklandıklarını ve *d* ile *e*'nin de aynı önbellek satırında barındırılıp, ilgili satırın L1 önbelleğine t2 anında getirildiği varsayımını yapmaktayız. Basitlik olması için her bir kelimenin boyunun 4 byte olduğu düşünülmektedir. *a* kelimesinin önbellekten R1 yazmacına read-after-fill işlemi vasıtası ile yüklendiği *t3* anında meydana gelen hata olayı $EGE_{t1,t3}(a)$ ve büyüklüğü ise *4 x (t3-t1)*'dir. Aynı şekilde *b* kelimesi üzerinde gerçekleşen bir okuma işlemi ile $EGE_{t2,t4}(b)$ olayı oluşur. Daha sonra bu iki olaya ilişkin hatalar *b* kelimesine store komutunun (I4) gerçekleştiği *t5* anında R1 ve R2 yazmaçları aracılığıyla yayılır (write-after-fill-from-L2). Şuna dikkat çekmekte fayda var; *a* ve *b* kelimeleri daha önceden değiştirilmiş ve diğer kelimelerden hatalar bunlara taşınmış olabilir. Her ne kadar bu tip hatalar *t5* anından itibaren *b*'nin güvenirliğini etkilese de, bizim modelimizde bu *EGE*'leri *EPS(a)*'dan çıkarıp *EPS(b)*'ye koymamaktayız; çünkü böyle yapsak da *AVF* hesabında bir değişikliğe yol açmayacaktır. Yani $EPS_{t5}(b)$'nin değeri $\{EGE_{t1,t3}(a)\}$ U $\{EGE_{t1,t4}(b)\}$ olur. Eğer $|EPS_{t5}(b)|$, $EPS_{t5}(b)$'nin hata değerinin büyüklüğünü gösterirse, *t5* anında ilgili load komutu gerçekleştiğinde bu değer $4\times(t3-t1) + 4\times(t4-t1)$ olacaktır. *t6* anında gerçekleştirilen dallanma (branch) komutunu (I11) dikkate aldığımızda aynı hata olayı $EGE_{t1,t3}(a)$ (*b* kelimesine daha önce yayılan) dallanma komutuna da yayılır. Ancak *AVF* hesaplaması yaparken bu iki yayılımdan sadece biri hesaba katılmaktadır.

Dallanma komutunu (I11) izleyen load komutunun (I12) dallanmanın yanlış yolunda olduğunu ve spekülatif olarak çalıştırıldığını kabul edelim. Bu kabuller altında store komutu (I13) commit etmeyeceği için *b*'den *e*'ye herhangi bir hata yayılımı olmayacaktır. I21 ve I22 komutları *t5* anından *t8* anına kadar toplanan hataların *b*'den *c*'ye taşınmasına yol açar. Dolayısıyla $EGE_{t5,t8}(b)$, $EPS_{t9}(c)$'ye I22'nin commit ettiği zamanda (*t9*) eklenir. I23 komutu *t10* anında commit ettiğinde $EGE_{t2,t10}(d)$'nun *b*'ye *t11* anında yazılmasına sebep olur ($EGE_{t2,t10}(d)$ $EPS_{t11}(b)$'ye eklenir). Böylece $EPS_{t11}(b)$'nin değeri $\{EGE_{t2,t10}(d)\}$ olur. Eğer *b* daha sonra bazı işlemlerde yer alırsa, sadece *t11* anından sonra toplanan hataları dikkate almalıyız. Bunun nedeni, *t11* anından önce toplanan hataların üzerine store komutu (I25) ile yazma yapılmış olmasıdır.

3.1.4. Önbellekler İçin AVF'lerin Hesaplanması

Şekil 4'te bir CMP sistemin basit diyagramı çizilmiştir. CMP sistemimizde her biri ayrı L1 veri önbelleği barındıran n tane işlemci çekirdeği bulunmaktadır. Tipik olarak n değeri 4, 8 veya 16 değerlerinden biri olabilir. Paylaşılır durumdaki L2 veri önbelleği erişim gecikmelerini azaltmak için n parçaya ayrılmıştır. L1 önbellekleri L2 önbellek bölmelerine şekilde de görüldüğü üzere bir iletişim ağıyla bağlıdır. Bütün çekirdekler, bölmeli durumdaki L2 önbelleği ve bu bellekler arasındaki iletişim ağı aynı chip içerisindedir. Yani üzerinde durduğumuz sistem, tek çipli çok işlemci çekirdeklidir (single chip multicore). Aşağıdaki AVF hesaplamaları tek çipli çok çekirdekli bir sistemi temel alsa da bu hesaplamalar çok çipli çok çekirdekli sistemlere (multi chip multi core systems) kolayca adapte edilebilir.



Şekil 4. CMP sistemimizin genel görünümü.

Bu bölümde bizim hata oluşumu ve yayılımı mekanizmalarımızı kullanmak suretiyle hem her bir çekirdekte bulunan L1 önbellekler için birer AVF hem de sistemdeki tüm L1'ler için genel bir AVF hesaplayacağız. Hesaplamalarımızda sadece L1 veri önbelleklerinde toplanan kritik hatalarla ilgileniyoruz. Yani, işlemci çekirdeklerinin ve paylaşılan L2 veri önbelleğin hatalardan etkilenmediklerini düşünüyoruz. Kritik hatalar hata bütçemizi oluşturan yegâne geçici hatalardır. Nerede oluştuklarına ve nereye

yayıldıklarına bağlı olarak bu hataları, Tablo 1'de gösterildiği üzere Intra-L1, Inter-L1 ve Inter-L2 olarak üç kategoriye ayırmaktayız. Intra-L1 hatalar, oluştukları L1 önbelleğine yayılan hatalardır. Diğer taraftan Inter-L1 hatalar, bir L1 önbelleğinde oluşan fakat başka bir L1 önbelleğe yayılan hatalardır. Inter-L2 hatalar ise L1 önbelleğinde oluşmuş ve buradan direk olarak L2 önbelleğine yayılan hatalardır.

$$Intra-L1(i) = \sum_{\substack{i.\ L1'dan\ L2'ya \\ l\ satırı}} \sum_{kelime\ w} \sum_{\substack{intra-error \\ e \in EPS(w)}} |e| + \sum_{\substack{branch \\ b}} \sum_{\substack{intra-error \\ e \in EPS(b)}} |e| \tag{3.1}$$

Birinci eşitlikte yer alan *Intra-L1(i) i.* L1 önbelleğe ait intra hataları ifade etmektedir. Eşitliğin ilk terimi, önbellek kelimelerin aktif dönemlerinde oluşan ve aynı önbellekteki kelimelere yayılan hataları gösterilmektedir. Bir önbellek kelimesi için aktif dönem (Active Period, AP), ilgili satırın önbelleğe getirildiği andan başlayarak bu kelimeye yapılan son erişime kadar geçen dönemdir. Eşitlikteki ikinci terim, kontrol akışına (Control Flow, CF) etki eden hataları gösterir. Diğer bir deyişle bunlar, bir dallanma komutunun koşulunun ve/veya hedef adresinin belirlenmesinde yer alan verileri etkileyen hatalardır.

$$Inter-L1(i,j) = \sum_{\substack{i.\ L1'dan\ j.\ L1'a \\ satır\ l}} \sum_{kelime\ w} \sum_{\substack{L1\ inter-error \\ e \in EPS(w)}} |e| \tag{3.2}$$

İkinci eşitlikte tanımı yapılan *Inter-L1(i,j), i.* L1 önbellekte oluşan ve *j.* L1 önbelleğe yayılan hataları göstermektedir. Bu hatalar, *i.* L1 önbelleğin bloklarında tutulan kelimelerin bekleme dönemlerinde (Idle Period, IP) topladığı hatalardır. Bir önbellek satırında tutulan bir kelimenin bekleme dönemi o kelimeye yapılan son erişim zamanından başlar ve o kelimenin ait olduğu satırdaki verilerin kopyasının başka bir L1 önbelleğe taşınmasına kadar devam eder. Bu eşitlikten yola çıkarak aşağıdaki eşitlik bize *i.* önbellekten diğer önbelleklere yayılan toplam inter hataları gösteren *Inter-L1(i)*'yi verecektir. Bu eşitlikte n, sistemdeki işlemci çekirdeklerin diğer bir deyişle L1 önbelleklerin sayısını göstermektedir.

$$Inter-L1(i) = \sum_{j=0 \wedge j \neq i}^{n-1} Inter-L1(i,j) \tag{3.3}$$

Kritik hataların son kategorisi *Inter-L2* hatalardır. Bu hatalar kirli bloklara ait kelimelerin ölü dönemlerinde (Dead Period, DP) toplanır ve ilgili kirli bloğun L1 önbelleğinden çıkartılması esnasında paylaşılan L2 önbelleğe geri yazılma işlemi ile L2

önbelleğe taşınırlar. *Inter-L2(i), i.* L1 önbellekten yayılan bu tip hataları temsil etmekte olup aşağıda 3.4. eşitlikteki gibi ifade edilebilir.

$$Inter-L2(i)=\sum_{\substack{i.L1'dan\\ satır\ l}}\ \sum_{\substack{L2'ya\\ kelime\ w}}\ \sum_{\substack{L2\ inter-error\\ e}}|e| \tag{3.4}$$

3.1, 3.3 ve 3.4. eşitliklerden yola çıkarak *i.* işlemci çekirdeğine ait L1 önbelleği için AVF aşağıda yer alan 3.5. eşitlikle ifade edilebilir.

$$AVF(i)=\frac{Intra-L1(i)+Inter-L1(i)+Inter-L2(i)}{Cache\_Size(i)\times Execution\_Cycles(i)} \tag{3.5}$$

Son olarak *n* tane işlemci çekirdeğinden oluşan bir CMP sisteminde bütün L1 önbelleklerini göz önünde bulundurarak hesaplanan genel AVF,

$$AVF=\frac{\sum_{i=0}^{n-1}Intra-L1(i)+Inter-L1(i)+Inter-L2(i)}{\sum_{i=0}^{n-1}Cache\_Size(i)\times Execution\_Cycles(i)} \tag{3.6}$$

şeklinde verilebilir.

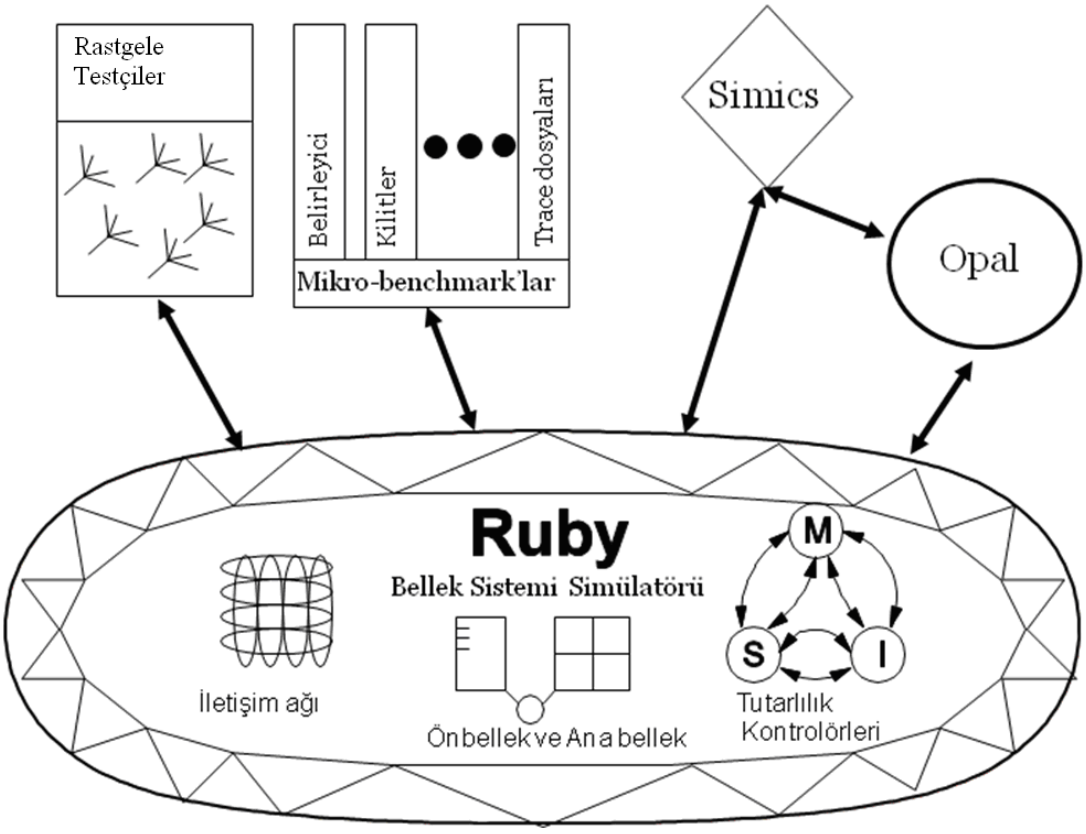
3.1.5. Simülasyon Ortamı

B iz deneylerimizde simülatör olarak Multifacet General Execution-Driven Multiprocessor Simulator Toolset'ini (GEMS, 2009) kullandık. GEMS genel olarak veritabanı ve web sunucuları gibi sistemlerde kullanılan çok işlemcili donanımların performansını incelemek ve karakterize etmek için oluşturulmuş GNU-GPL (GNU-GPL, 2009) altında lisanslanmış bir simülatörler kümesidir (Martin ve ark., 2005). Simülatörler, bilgisayar mimarisi ile uğraşan araştırmacıların ortaya koydukları donanımsal yeniliklerin test edilmesi için çok sıkça başvurdukları bir yöntemdir.

Simülasyonlarda önemli olan iki unsur vardır. Bunlardan birincisi; hedef alınan mimari sistemin yeterli derinlikteki detayını hızlı bir şekilde simüle edebilmektir. İkincisi ise simüle edilen sistemi gerçekçi programlar ve işletim sistemleri altında çalıştırılabilmektir. Örneğin SimpleScalar (Austin ve ark., 2002; SimpleScalar, 2009) bilgisayar mimarisi üzerine yapılan araştırmalarda şimdiye kadar oldukça çok kullanılmıştır. Bu simülatör oldukça hızlı olmasına rağmen sadece kullanıcı modu komutlarını simüle edebilmekte ve SPEC CPU2000 (SPEC, 2009) gibi tek iş parçacıklı

(single threaded) uygulama programları çalıştırabilmektedir. Ancak günümüzde veri tabanları, web sunucuları ve paralel bilimsel programları çalıştırabilen daha karmaşık ve çok iş parçacıklı (multi threaded) makinelere olan ilgi gittikçe artmaktadır. Bu tip makinelerde çalışan programlar giriş/çıkış ve iş parçacıkları arası senkronizasyon gibi işletim sisteminin fonksiyonlarına oldukça bağımlılık gösterirler. GEMS, çok işlemcili makinelerde çok iş parçacıklı uygulamaların simüle edilmesini sağlar.

GEMS toolsetinin genel yapısı Şekil 5'te gösterilmektedir. Ruby, bellek sistem simülatörü GEMS'in kalbi durumundadır. Şekilden de görüleceği üzere Ruby'e gönderilen bellek işlemlerine kaynak teşkil edebilecek GEMS'in dört farklı sürücüsü vardır. Bunlar:



Şekil 5. Gems simülatörüne yukardan bakış (Martin M. K. ve ark., 2005).

3.1.5.1. Rastgele Testçiler

Ruby'nin en basit şekildeki sürücüsüdür. Bellek sisteminde stres testleri ve uç koşulları test etmeye yarar. Bellek sistemi hataları, önbellek tutarsızlık problemleri ve yarış koşullarını tespit etmekte kullanılır.

3.1.5.2. Mikro-Benchmark

Bu sürücü, genel bir arayüz vasıtası ile çeşitli mikro-benchmarklara destek verir. Bu modül basit durum zamanlamaların doğruluğunu belirlemede kullanıldığı gibi aynı zamanda özel durumlardaki performans analizlerini yapmakta da kullanılabilir.

3.1.5.3. Simics

Bu sürücü, Simics (Magnusson, 2002) fonksiyonel simülatörünü kullanmak suretiyle pipeline beklemesi olmayan ve in-order olarak çalışan basit bir işlemciyi modeller. Bütün bellek işlemleri (load, store ve instruction fetch) istekleri Ruby'ye aktarılır. Ruby birinci seviye önbellek erişim isteklerini karşılamaya çalışır. Eğer önbellekte aranılan veri veya komut bulunmuşsa Simics çalışmaya devam eder ve çoklu işlemci ortamında süreçler arasında anahtarlama yapılır. Aranılan veri önbellekte bulunmazsa (ıska durumu) Ruby Simics'in çalışmasını askıya alır ve önbellek ıskasını simüle eder. Simics fonksiyonel doğruluğu sağlar (çalıştırılan komutlardan elde edilen sonuçların doğru olmasının sağlanması). Ruby ise zamana bağlı fonksiyonelliği sağlar (örneğin, hangi işlemcinin ne zaman bellek bloğuna erişebileceği gibi).

3.1.5.4. Opal

Bu sürücü dinamik-planlı (dynamically-scheduled) SPARC v9 işlemcisini modeller ve fonksiyonel doğruluk tespitinde Simics'i kullanır. Günümüzdeki işlemcilerin pekçoğu dinamik planlı super-scalar işlemciler olduğu için modern işlemcilerin modellenmesi açısından Opal oldukça önemlidir. Opal oldukça detaylı bir zamanlama modeli kullanarak günümüzdeki modern super-scalar işlemcilerinde oldukça yaygın olan derin pipeline, dallanma tahmini, dinamik komut salımı, çoklu çalışma üniteleri, out-of-order bellek işlemleri ve bellek pay basına izin vermek için load/store ünitelerini modeller. Opal komut fetch işlemi, komut çözümü, dallanma tahmini, dinamik komut planlama, komutları çalıştırma ve spekülatif olarak bellek işlemlerini gerçekleştirmek suretiyle Simics'in zamanlama açısından önünde çalışır. Opal bir komutun retire olma zamanın geldiğini tespit edince bir komut ilerlemesi için ilgili Simics işlemcisine talimat verir. Opal daha sonra komutu doğru çalıştırdığından emin olmak için kendi işlemci durumunu Simics işlemci durumu ile karşılaştırır. Çoğu durumda Opal ve Simics komut yürütümünde uyuşma gösterirler. Ancak kesme, I/O işlemleri veya Opal tarafından gerçekleştirilemeyen nadir

kernel komutlarında Opal farklılığı tespit eder ve kendi işlemci durumunu Simics işlemci durumuna göre günceller.

Biz, bu çalışmamızda çok işlemcili bellek sisteminin zamanlama simülatörü olarak Ruby'i kullandık. Ruby önbellekleri, önbellek kontrolörleri, sistem içi haberleşme ağı, ana bellek bölmeleri ve ana bellek kontrolörlerini modeller. Ruby bunlara ek olarak noktadan noktaya anahtarlamalı ara bağlantı ağını (point-to-point switched interconnection network) modeller. Ruby, hem directory tabanlı (directory-based) ve hem de dinleme tabanlı (snooping-based) sistemlere destek verir.

Çizelge 2. Bellek sisteminin temel özellikleri

L1 D-cache (birinci seviye veri önbelleği)	64 KB, 4 way, 64 B blok boyu, 3cycle cevap süresi
L1 I-cache (birinci seviye komut önbelleği)	64 KB, 4 way, 64 B blok boyu, 3 cycle cevap süresi
L2 U-cache (ikinci seviye birleşik önbellek)	16 MB, 8 way, 64 B blok boyu, banklara ayrılmış, 6 cycle önbellek cevap süresi
Ana bellek	4 GB adres uzayı, 80 cycle bellek cevap süresi
Directory-latency (directory gecikmesi)	80 cycle
Sayfa boyu	4 KB
On chip link latency (chip içi bağlantı gecikmesi)	1 cycle
Network link latency (chip dışı bağlantı gecikmesi)	40 cycle
Ağ topolojisi	Hierarchical Switch
Cache coherency protocol (önbellek tutarlılık protokolü)	Snooping-based

Çizelge 2'de deneylerde kullandığımız CMP'nin bellek sistemine ait temel konfigürasyon parametreleri verilmektedir. Bu parametrelerin çoğunun değeri simülatör kodu ile bize sunulan varsayılan değerlerle aynıdır. Birinci seviyedeki veri ve komut önbelleklerinin özellikleri aynı olup büyüklükleri 64 KB ve blok boyları 64 byte dir. Hem komutları hem de verileri saklayan bütünleşik ikinci seviyedeki önbelleğin büyüklüğü 16 MB olup blok boyu 64 byte dir. Erişim gecikmelerini azaltmak için ikinci seviyedeki önbelleğin banklara (bölmelere) ayrıldığı varsayılmıştır. L2 önbellek bankların blok adresin en düşük bitleriyle seçildiği varsayımını yapıyoruz. Ruby çok sayıda değişik bellek sistemi topolojisine destek vermektedir. Biz bu çalışmamızda ağ topolojisi olarak hierarchical switch topolojisini seçtik.

Çizelge 3. İşlemcinin temel özellikleri

Pipeline	3 fetch-stage, 4-decode-stage, 3 retire-stage, değişken execution stage
Fetch/decode/dispatch/execute/retire rate	4
ALU sayıları	4 integer (çarpma/toplama), 2 integer (bölme), 2 dallanma ünitesi, 4 FP (toplama), 2 FP (çarpma), 2 FP (bölme/karekök), 2 load ünitesi, 2 store ünitesi
Yanlış dallanma tahmini cezası	1 cycle
ROB (Reorder Buffer) Size	128
IWINDOW (komut penceresi) büyüklüğü	64

Çizelge 3'te deneylerimizde kullandığımız CMP'nin her bir işlemcisine ait önemli özelliklerden bazıları verilmektedir. Kullandığımız işlemci günümüz modern işlemcilerinde mevcut pipeline tekniğine sahiptir. Komut getirme aşaması (fetch-stage) 3 adım (3 cycle), kod çözme aşaması (decode-stage) 4 adım, komutların sonlandırılması (retire-stage) 3 adım ve kod yürütümü (execution-stage) ise komutun tipine bağlı olarak adım sayısında değişiklik göstermektedir. Örneğin tamsayı (integer) ALU işlemleri 1 cycle, integer çarpma 4 cycle ve kayan noktalı (floating point) karekök alma işlemi ise 24 cycle zaman almaktadır. Her bir çekirdeğin genişliği 4'tür. Yani, aynı anda en fazla 4 komut fetch/decode/dispatch/execute/retire edilebilmektedir. ALU sayıları komut tiplerine göre değişiklik göstermektedir. Çekirdekler 2'şer tane load/store üniteleri barındırmaktadırlar. İşlemciler dallanma tahmini mekanizmasına sahip olup, yanlış dallanma cezası 1 cycle dir. İşlemcilerde out of order olarak çalıştırılan komutların inorder olarak sonlandırılmalarını sağlamak için günümüz superscalar işlemcilerinde mevcut Reorder Buffer (ROB) mekanizması mevcut olup, bu 128 komutu barındırabilmektedir. Fetch edilen komutlar, boyu 64 olan bir pencerede (IWINDOW) tutulmaktadır.

BÖLÜM 4

ARAŞTIRMA BULGULARI VE TARTIŞMA

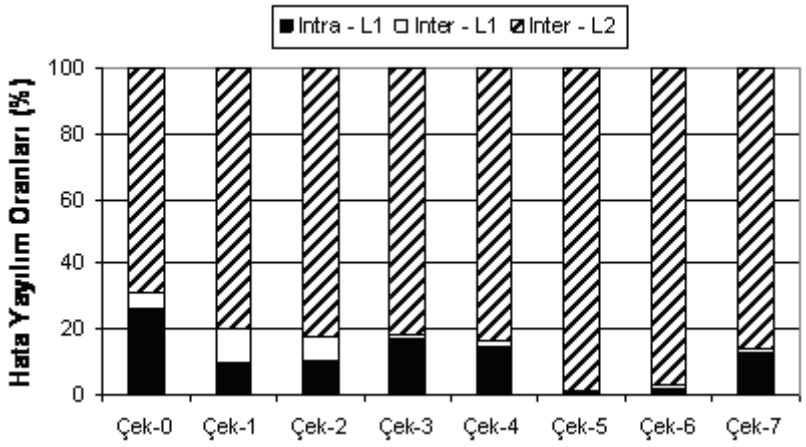
4.1. Benchmarkların Temel Karakteristikleri

Deneylerimizi tek çipli ve 8 çekirdekli bir CMP için gerçekleştirdik. Deneylerimizde SPEC OMP2001 (SPEC, 2009) benchmarklarından 8 tane kullandık. SPEC OMP2001, paylaşımlı bellek sistemli işlemcilerin (Shared Memory Processor, SMP) performansını ölçmek için geliştirilmiş çeşitli uygulama programları içermektedir. SPEC OMP2001, SMP sistemlerine odaklanmak suretiyle sistem işlemcisinin (CPU), paylaşımlı bellek sisteminin, derleyicilerin ve OpenMP (OpenMP, 2009) gerçekleştiriminin performanslarını ölçmeyi hedefler. Biz uygulama programlarını paralelleştirmek için SMP sistemlerinde en yaygın kullanılan paralelleştirme yöntemlerinden OpenMP (OpenMP, 2009) yöntemini kullandık. OpenMP, API (Application Program Interface) LINUX ve Windows platformlarını da içeren birçok değişik platformda C/C++ ve Fortran dillerinde SMP sistemleri için paralel programlamaya destek sağlar. Biz , simülatörleri çalıştırma platform olarak Intel x86 tabanlı işlemciler ve Red Hat Linux ortamını seçtik.

Geçici hata oluşumu ve yayılımı modelimiz doğrultusunda GEMS'in Opal ve Ruby simülatör kaynak kodlarında gerekli değişiklikleri yaptık. Opal ve Ruby'nin her ikisi de C++ dilinde yazılmış olup gcc altında derlenebilmektedir. Uygulama programlarının baştan sona simülasyonu çok uzun zaman aldığı için (aylarca), biz bu çalışmamızda 0 nolu çekirdeğin 100 milyon komut çalıştırma süresi boyunca simülatörlerden istatistikler topladık. Çizelge 4'te simülatörden elde ettiğimiz benchmarklarımıza ilişkin önemli özellikler gösterilmektedir. Çizelgenin ikinci sütünü L1 veri önbelleği erişim sayılarını, ikinci sütün L1 veri önbelleğindeki ıskalar yüzünden banklara ayrılmış L2 veri önbelleğe yapılan erişim sayılarını ve son sütün ise yürütme sürelerinin cycle cinsinden değerlerini göstermektedir. Erişim sayıları 8 çekirdeğe ait erişim sayılarının toplamından oluşmaktadır.

Çizelge 4. Benchmarklarımıza ait önemli istatistikler

Benchmark	L1 Dcache Erişim Sayısı	L2 Dcache Erişim Sayısı	Yürütme Süresi (cycle)
310.wupwise	55409816	494006	77950159
312.swim	55297159	1310444	76116179
314.mgrid	57577342	1251780	78151479
316.applu	54311865	491391	75925030
318.galgel	56677152	391717	76073868
320.equake	56446139	450340	79110210
324.apsi	52597703	1352592	78109864
328.fma3d	56872931	499399	80499621

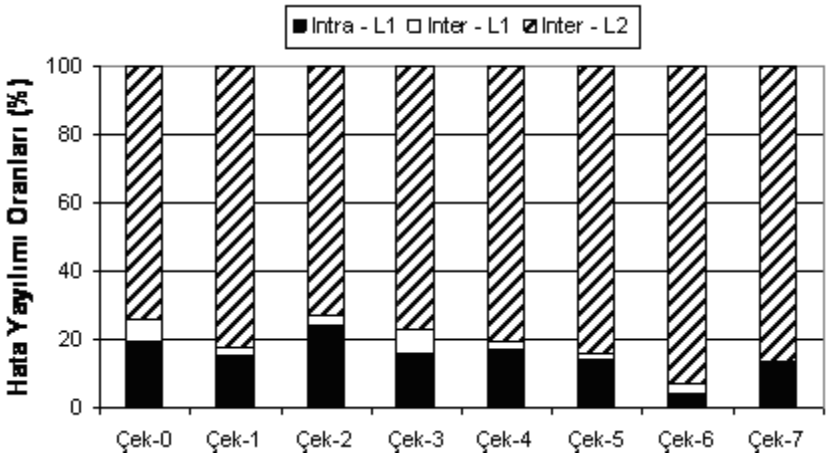


Şekil 6. 316.applu uygulama programına ait birikimli hata yayılım oranları.

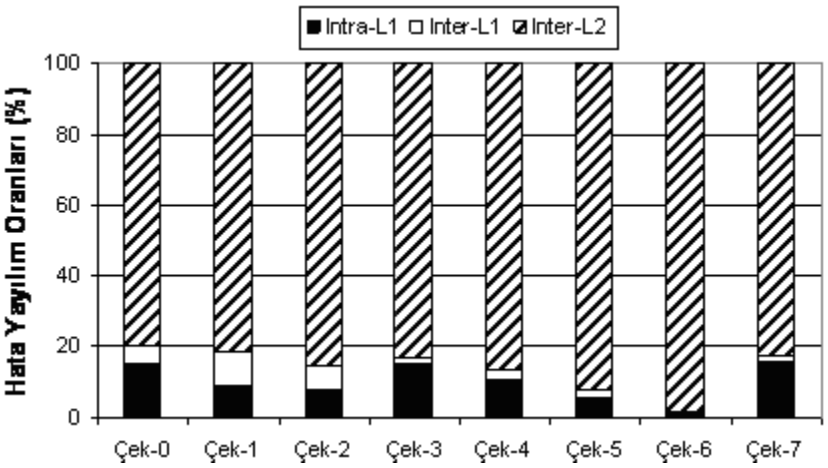
4.2. Benchmarklara Ait Birikimli Hata Yayılım Oranları

Daha önce önerdiğimiz geçici hata oluşumu ve yayılımı modeline göre bu kısımda hata yayılım çeşitlerinin birikimli oranlarını değişik uygulama programları için vermeye çalışıyoruz. Şekil 6, Şekil 7 ve Şekil 8 sırasıyla 316.applu, 320.equake ve 312.swim programlarının hata yayılım oranlarını 8 çekirdek için göstermektedir. Bu üç şekilden de kolayca görülebileceği gibi Inter-L2 hataları, Intra-L1 ve Inter-L1 hatalarına göre oldukça baskındır. Örneğin Şekil 6'e bir göz atacak olursak, 316.applu programı için 0 nolu çekirdekte hata yayılımının %69'unu Inter-L2 hata yayılımı teşkil ederken, Inter-L1 ve Intra-L1 yayılan hataların sırasıyla yaklaşık %4,5 ve %26,5'luk kısımlarını teşkil etmektedirler. Bu genel gözlemimiz 316.applu uygulamasında diğer çekirdekler için de

geçerlidir. Şekil 7 ve Şekil 8'e bakarak benzer gözlemleri 320.equake ve 312.swim uygulamaları için de yapabiliriz.



Şekil 7. 320.equake uygulama programına ait birikimli hata yayılım oranları.



Şekil 8. 312.swim uygulama programına ait birikimli hata yayılım oranları.

4.3. Inter-L2 Hata Yayılımının İrdelenmesi

Yukarıda da bahsedildiği gibi yayılan hataların çok önemli bir kısmı Inter-L2 hata yayılımı şeklinde olmaktadır. Dolayısıyla bu tür hata yayılımlarının azaltılmasına yönelik mimari yaklaşımlar çok önemlidir. Inter-L2 hata yayılımı, L1 önbelleklerde kirli önbellek bloklarında tutulan kelimelerin IP'leri süresince topladıkları hataların paylaşımlı L2 önbelleğine taşınmaları sonucu gerçekleşir. Biz bu bölümde Inter-L2 hatalarının yayılma şekillerini (kirli kelime veya temiz kelime vasıtası ile taşınma) irdelemeye çalışacağız.

Çizelge 5. 316.applu benchmarkına ait Inter-L2 hata yayılımına sebep olan kirli L1 blokların kirli kelime oranlarına göre irdelenmesi

		Çekirdek Numarası							
		çek-0	**çek-1**	**çek-2**	**çek-3**	**çek-4**	**çek-5**	**çek-6**	**çek-7**
Kirli kelime Oranları (%)	**1**	28	38	36	46	41	3	18	51
	2	44	68	69	77	71	5	85	76
	3	96	80	84	91	83	7	87	91
	4	97	94	94	94	89	7	87	94
	5	97	95	95	96	93	8	87	96
	6	98	97	97	98	94	11	99	97
	7	99	99	97	100	96	13	99	98
	8	99	99	97	100	99	99	99	100
	9	99	99	99	100	100	100	99	100
	10	99	99	99	100	100	100	100	100
	11	99	99	99	100	100	100	100	100
	12	99	99	99	100	100	100	100	100
	13	99	99	99	100	100	100	100	100
	14	99	99	99	100	100	100	100	100
	15	99	99	99	100	100	100	100	100
	16	100	100	100	100	100	100	100	100

Çizelge 5, 6 ve 7 sırasıyla 316.applu, 320.equake ve 312.swim uygulama programlarına ait Intra-L2 hata yayılımına sebep olan kirli L1 blokların kirli kelime oranlarına göre irdelenmesine ışık tutmaktadır. Biz bu irdelemeyi sistemdeki her bir çekirdek için gerçekleştirdik. L1 önbellek blok uzunluğunun 64 byte olduğunu ve her bir kelimenin boyunun 4 byte olduğunu kabul ettik. Dolayısıyla her bir L1 önbellek bloğu 16 kelimeden oluşmaktadır. Bu çizelgelerden kolayca görüleceği gibi Intra-L2 hata yayılımları genellikle az sayıda kirli kelimeler ve çok sayıda değişikliğe uğramamış kelimeler üzerinden yapılmaktadır. Bu temel gözlem her bir çekirdek önbelleği için geçerlidir. Örneğin 320.equake benchmarkını göz önünde bulunduralım. 4. çekirdekte L1 önbelleğinden paylaşımlı L2 önbelleğe geri yazılan blokların %47'si sadece 1 tane kirli kelime içermektedir. Aynı önbelleğe ait hiçbir blok 10 veya daha fazla sayıda kirli kelime içermemektedir. Yine 6. çekirdek için L2 önbelleğe geri yazılan blokların %30'u sadece 1 tane kirli kelime (15 tane değişikliğe uğramamış kelime) içermektedir. Aynı çekirdek için

L2'ye geri yazılan blokların %50'den fazlası 3 veya daha az sayıda kirli kelime içermektedir.

Çizelge 6. 320.applu benchmarkına ait Inter-L2 hata yayılımına sebep olan kirli L1 blokların kirli kelime oranlarına göre irdelenmesi

		Çekirdek Numarası							
		çek-0	**çek-1**	**çek-2**	**çek-3**	**çek-4**	**çek-5**	**çek-6**	**çek-7**
Kirli kelime Oranları (%)	**1**	27	40	26	30	47	55	30	50
	2	46	71	36	64	74	77	48	74
	3	96	81	40	77	83	89	52	90
	4	97	96	44	90	89	95	54	94
	5	97	97	46	93	93	96	54	95
	6	98	98	47	95	94	97	55	96
	7	98	98	48	95	96	98	55	97
	8	99	100	93	98	99	100	99	100
	9	99	100	94	98	100	100	99	100
	10	99	100	94	98	100	100	99	100
	11	99	100	94	98	100	100	99	100
	12	99	100	94	98	100	100	100	100
	13	99	100	94	98	100	100	100	100
	14	99	100	94	98	100	100	100	100
	15	99	100	94	99	100	100	100	100
	16	100	100	100	100	100	100	100	100

Çizelge 7'ye bakarak benzer şeyleri 312.swim uygulaması için de söyleyebiliriz. 4. çekirdeğin L1 önbelleğinden L2 önbelleğe nter-L2 hatası taşıyan kirli blokların %39'u sadece 1 tane değişikliğe uğramış kelime içermektedir. Bu çekirdek için 9 veya daha fazla sayıda kirli kelime içeren L1 önbellek bloğu mevcut değildir. 7. çekirdekte Inter-L2 hatası taşıyan blokların %43'ü sadece 1'er tane kirli kelime içermektedirler. Aynı çekirdek için 10 veya üzeri kirli kelime barındırıp, Inter-L2 hatası taşıyan L1 bloğu mevcut değildir.

Bu gözlemlerimiz bize Inter-L2 hatalarının azaltılması hususunda önemli bir ipucu vermektedir. CMP sistemlerinde L1 önbelleklerinden paylaşımlı L2 önbelleğe geri yazma işlemi sırasında, kirli blokların tamamını değil de sadece değişikliğe uğrayan kelimeleri yazmak Inter-L2 hata yayılımının azaltılmasında önemli bir etken olabilir. Bu çalışmanın konusu sadece hata yayılımları konusunda bir model oluşturmak olduğu için biz bu durumun araştırmasını başka bir çalışmada gerçekleştirmeyi planlamaktayız.

Çizelge 7. 312.swim benchmarkına ait Inter-L2 hata yayılımına sebep olan kirli L1 blokların kirli kelime oranlarına göre irdelenmesi

		Çekirdek Numarası							
		çek-0	çek-1	çek-2	çek-3	çek-4	çek-5	çek-6	çek-7
Kirli kelime Oranları (%)	1	25	33	34	39	39	40	14	43
	2	48	66	67	67	66	46	95	68
	3	96	80	82	79	77	47	96	82
	4	98	91	91	87	84	49	97	90
	5	98	93	93	91	88	50	97	93
	6	99	94	95	93	92	51	97	96
	7	99	96	96	94	95	52	97	96
	8	99	99	98	100	100	99	99	96
	9	99	99	98	100	100	99	99	100
	10	99	99	98	100	100	99	99	100
	11	99	99	98	100	100	99	100	100
	12	99	99	99	100	100	99	100	100
	13	99	99	99	100	100	99	100	100
	14	99	99	99	100	100	99	100	100
	15	99	99	99	100	100	99	100	100
	16	100	100	100	100	100	100	100	100

4.4. AVF Değerleri

Biz bu bölümde, bizim CMP sistemimiz için 6. eşitlik aracılığıyla hesapladığımız AVF değerlerini sunacağız. Daha önce de belirttiğimiz gibi bir donanım bileşeni için AVF, o bileşende oluşabilecek bir geçici hatanın yüzde kaç ihtimalle kendini programın çıktısında gösterebileceğini (programın yanlış sonuç üretebileceğini) ifade eder. Sistemdeki donanım bileşenlerinin AVF'lerinin belirlenmesi suretiyle geçici hatalardan en fazla etkilenebilen bileşenler belirlenir. AVF değerinin yüksek olması o bileşenin geçici hatalara çok duyarlı olması (hatalardan çok etkilenmesi) anlamına gelir. AVF değeri yüksek olan bileşenler için hatalara karşı koruma yöntemleri geliştirilme yoluyla sistemin daha korunmalı hale getirilebilmesi mümkün olabilmektedir.

Çizelge 8. Benchmarklara ait AVF değerleri

Benchmark	Intra-L1 AVF	Inter-L1 AVF	Inter-L2 AVF	Toplam AVF
310.wupwise	0,003	0,001	0,0057	0,061
312.swim	0,004	0,002	0,062	0,068
314.mgrid	0,006	0,001	0,021	0,027
316.applu	0,005	0,002	0,092	0,099
318.galgel	0,005	0,002	0,020	0,027
320.equake	0,013	0,003	0,054	0,070
324.apsi	0,007	0,001	0,022	0,030
328.fma3d	0,012	0,003	0,066	0,081

Değişik uygulama programlarına ait AVF değerleri Çizelge 8'de verilmektedir. Çizelgenin ikinci sütunu Intra-L1 hata yayılımın getirdiği AVF değerini, üçüncü sütün Inter-L1 hata yayılımının getirdiği AVF değerini ve dördüncü sütün ise Inter-L2 hata yayılımının sağladığı AVF değerini göstermektedir. Öte yandan çizelgenin son sütünü ise toplam AVF değerini vermektedir. 312.swim uygulaması için toplam AVF değeri 0,068'dir. Bu şu anlama gelmektedir: CMP sisteminin bir çekirdeğine ait L1 önbelleğinde parçacık çarpması sonucu oluşabilecek bir hata %6,8 ihtimalle ilgili programın yanlış sonuç üretilmesine sebep olmaktadır. Bu oranın çok küçük olduğu izlenimine kapılabiliriz. Hâlbuki bu ihmal edilemeyecek bir orandır. Genel olarak işlemcilerde önbellekler, işlemci için harcanan transistorların önemli bir kısmını içerdiğinden olası parçacık çarpmalarının en önemli ana hedeflerinden birini teşkil ederler. Ayrıca, önbellekte oluşabilecek bir hata sistemin diğer bileşenlerine (CPU, L2 önbelleği) kolaylıkla yayılabileceğinden önbellekteki geçici hatalar programın doğru sonuç üretmesinde önemli etkilere sebep olur. Çizelgeden de görülebileceği üzere toplam AVF içinde en önemli katkıyı Inter-L2 hata yayılımı yapmaktadır.

BÖLÜM 5

SONUÇ VE ÖNERİLER

Günümüzde aynı yonga içerisine artan sayıda işlemci çekirdeği yerleştirmek gittikçe yaygınlaşmaktadır. Bunun temelde iki sebebi vardır. İlki, işlemci teknolojisin gelişmesi sonucu küçülen transistor boyutlarıyla aynı alana daha fazla transistor yerleştirmek mümkün olmaktadır. Bu artan sayıdaki transistoru tek çekirdek için harcamak hem işlemcinin doğruluk testini çok zor ve pahalı hale getirmekte hem de açığa çıkan ısının kontrolünü zorlaştırmaktadır. Ayrıca uygulamaların sınırlı düzeyde komut seviyesi paralelliğine imkân vermesi çok karmaşık tek çekirdekli işlemcilerin tasarımını gereksiz kılmaktadır. İkinci sebep olarak, günümüzde yaygınlaşmış web uygulamaları, veritabanı uygulamaları ve paralelleştirilmiş bilimsel programlar performans açısından çok işlemcili makinelere ihtiyaç göstermektedir.

Bilgisayarlarda yüksek performans gibi veri güvenirliği de çok önemli bir tasarım kriteridir. Yüksek enerjili parçacıkların devrelere çarpması sonucu devre çıkışlarının depoladığı yük miktarı değişebilmekte ve sonuçta istenmeyen geçici bit değişiklikleri ortaya çıkabilmektedir. Bu bit değişiklikleri, küçülen transistor boyutları sonucu azalan besleme gerilimleri ile daha da kolaylaşmaktadır. Geçici hata diye adlandırılan bu durum programların doğru sonuç üretememesine hatta bilgisayar sisteminin veya uygulamanın çökmesine de yol açabilmektedir.

Bu çalışmada CMP sistemlerin L1 önbelleklerinde ortaya çıkan geçici hataları modellemeye çalıştık. Modelimiz hem hata oluşumu hem de hatanın yayılımını göz önünde bulundurmaktadır. Belli bir veride oluşabilecek veri bozulmasının şiddeti hem o verinin geçici hatalara maruz kalma süresiyle ve hem de o verinin uzunluğuyla doğru orantılıdır. Modelimize göre, herhangi bir L1 önbelleğinde oluşan hata üç farklı bileşene yayılabilir. Birincisi; aynı L1 önbelleğine, ikincisi; farklı bir çekirdeğe ait L1 önbelleğine, üçüncüsü ise paylaşımlı L2 önbelleğe. Biz, bu üç farklı hata yayılımını dikkate alarak CMP'nin L1 önbellek sisteminin AVF'sini formülize ettik ve zamanlama desteği olan bir simülatörle de farklı uygulama programları için AVF değerlerini hesapladık. Deneysel sonuçlarımız bize AVF'ye en büyük katkıyı L1 özel önbelleklerden paylaşımlı L2 önbelleğe yayılan hataların yaptığını göstermektedir. Bu tür hatalar, önbellek bloklarında tutulan kelimelerin DP'leri

süresince topladıkları hatalardır. Bu hatalar kirli önbellek bloklarının geri yazılmaları sonucu L2 önbelleğe yayılmaktadırlar. Dolayısıyla bu hataların yayılımını önleyen mekanizmaların geliştirilmesi veri güvenilirliği açısından son derece önemlidir. Deneysel sonuçlarımızdan çıkardığımız önemli ikinci husus şudur. L2 önbelleğe yayılan hataların çok önemli bir bölümü temiz (üzerinde değişiklik yapılmamış) veriler üzerinden olmaktadır. Bu durum bize, kirli L1 önbellek bloklarının L2 önbelleğine geri yazılırken blok verilerinin hepsinin değil de sadece üzerinde değişiklik yapılmışların yazılmasının veri güvenirliliğini artırmada önemli bir rol oynayabileceğini göstermektedir.

KAYNAKLAR

Austin T., Larson E. ve Dan E. D., 2002. SimpleScalar: An Infrastructure for Computer System Modeling. *IEEE Computer,* 35(2):59–67.

Barroso L. A., Gharachorloo K., McNamara R., Nowatzyk A., Qadeer S., Sano B., 2000. Smith S., Stets R., ve Verghese B. Piranha: a scalable architecture based on single-chip multiprocessing*. Int. Symposium on Computer Architecture, pp.* 282-293.

Biswas A., Racunas P., Cheveresan R., Emel J., Mukherjee S. S., ve Rangan R., 2005. *Computing architectural vulnerability factors for adress-based structures. Int. Symp. on Comp. Arch.*

Calin T., Nicolaidis M., ve Velazco R., 1996. Upset hardened memory design for submicron CMOS technology. *IEEE Trans. on Nuclear Science,* 43(6).

Cannon E. H., Reinhardt D. D., ve Makowenskyj P. S., 2004. SRAM SER in 90, 130 ve 180nm Bulk ve SOI Technologies. *Int. Rel. Phys. Symp.*

Carmichael C., 2006. "Triple Module Redundancy Design Techniques for Virtex FPGAs*, Xilinx Aplication Notes* 197 (v1.0.1).

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

Gomaa M., Scarbrough C., Vijaykumar T. N., ve Pomeranz I., 2003. Transient-fault recovery for chip multiprocessors. *Int. Symp. on Comp. Arch.*

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

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

Kadayif I. ve Kandemir M., 2007. Modeling ve improving data cache reliability. *ACM SIGMETRICS Conference.*

Kim S. ve Somani A. K., 2002. Soft error sensitivity characterization for microprocessor dependability enhancement strategy. *Int. Conf. on Dep. Sys. And Net.*

Kongetira P., Aingaran K., ve Olukotun K., 2005. Niagara: a 32-way multithreaded SPARC processor. *IEEE Micro,* 25(2):21-29.

Kumar S. ve Aggarwal A., 2006. Reducing resource redundancy for concurrent error detection techniques in high performance microprocessors. *Int. Symp. On High-Per. Comp.Arch.*

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

Magnusson P. S., *2002.* Simics: A Full System Simulation Platform. *IEEE Computer,*35(2):50–58.

Martin M. K., Sorin D. J., Beckmann B. M., Marty M. R., Xu M., Alameldeen A. R., Moore K. E., Hill M. D. ve Wood D. A., *2005.* Multifacet's General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *Computer Architecture News (CAN).*

Mukherjee S. S., Weaver C., Emer J., Reinhardt S. K., ve Austin T., 2003. A systematic methodology to compute the architectural vulnerability factors for a high-performance microprocessor. *Int. Symp. On Micro., Dec.*

Mukherjee S. S., Emer J., ve Reinhardt S. K., 2005. The soft error problem: an architectural perspective. *Int. symp. on High-Perf. Comp. Arch.*

OpenMP, 2009. The OpenMP API Specification for Parallel Programming. www.openMP.org

Ray J., Hoe J., ve Falsafi B., 2001. Dual use of superscalar datapath for transient-fault detection ve recovery. *Int. Symp. on Micro.*

Vijaykumar T., Pomeranz I., ve Cheng K., 2002. Transient-fault recovery using simultaneous multithreading. *Int. Conf. on Comp. Arch.*

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

Wang N. J., Quek J., Rafacz T. M., ve Patel S. J., 2004. Characterizing the effects of transient faults on a high-performance processor pipeline. *Int. Conf. on Dep. Sys. and Net.*

SimpleScalar, 2009. SimpleScalar LLC Toolset. http://www.simplescalar.com

SPEC, 2009. Standard Performance Evaluation Corporation. http://www.spec.org

Degalahal V., Li L., Narayanan V., Kandemir M., ve Irwin M. J., 2005. Soft errors issues in low-power caches. *IEEE Trans. on Very Large Scale Integ. Sys.,* 13(10):1157–1166.

GEMS, 2009. Multifacet General Execution-Driven Multiprocessor Simulator Toolset. www.wisc.edu/gems

GNU-GPL, 2009. Free Software Foundation. GNU General Public License . http://www.gnu.org/copyleft/gpl.html

Gomaa M. A. ve Vijaykumar T. N., 2005. Opportunistic transient-fault detection. *Int. Symp. on Comp. Arch.*

Irom F., Farmamesh F. F., Johnson A. H., Swift G. M., ve Millward D. G., 2002. Single-event upset in commercial silicon-on-insulator PowerPC microprocessors. *IEEE Trans. on Nucl.Sci.,* 49(6).

Karnik T., Hazucha P., ve Patel J., 2004. Characterization of soft errors caused by single event upsets in CMOS processes. *IEEE Trans. on Dep. and Sec. Comp,* 1(2):128–143.

Nguyen H. T. ve Yagil Y., 2003. A systematic approach to SER estimation ve solutions. *IEEE Int. Rel. Phys. Symp.*

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

Reinhardt S. K. ve Mukherjee S. S., 2000. Transient fault detection via simultaneous multithreading.*Int. Symp. on Comp. Arch.*

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

ÇİZELGELER LİSTESİ

Sayfa

ŞEKİLLER

Sayfa

ÖZGEÇMİŞ

KİŞİSEL BİLGİLER

Adı Soyadı :Selçuk KOYUNCU

Doğum Yeri :Kayseri

Doğum Tarihi :29.05.1984

EĞİTİM DURUMU

LisansÖğrenimi: Çanakkale Onsekiz Mart Üniversitesi-Mühendislik Fakültesi-Bilgisayar Mühendisliği Bölümü 2002–2006

Yüksek Lisans Öğrenimi : -

Bildiği Yabancı Diller: İspanyolca, İngilizce, Almanca

BİLİMSEL FALİYETLERİ

a) Yayınlar-SCI-Diğer :

Capturing and Optimizing the Interactions Between Prefetching and Cache Line Turnoff, by I. Kadayif, A. Zorlubas, S. Koyuncu, O. Kabal, D. Akcicek, Y. Sahin, and M. Kandemir. Microprocessors and Microsystems (Elsevier), 32(7):394-404, October 2008

b) Bildiriler –Uluslararası-Ulusal :

Exploiting potentially dead blocks for improving data cache reliability against soft errors Koyuncu, S.; Kadayif, I.; IEEE -Computer and Information Sciences, 2007. ISCIS 2007. 22nd of the International Symposium on Computer and Information Sciences

c) Katıldığı Projeler : TUBITAK Kariyer Projesi (no :*105E094)*

İŞ DENEYİMİ

Çalıştığı Kurumlar ve Yıl:

Çanakkale Onsekiz Mart Üniversitesi, Bilgisayar Mühendisliği Bölümü, Arş.Gör. 2006-2009

32 bit Bilgisayar Hizmetleri Ltd., Yazılım Mühendisi 2009- …

İLETİŞİM

E-posta Adresi: selcuk.koyuncu@gmail.com

Telefon : 0505 229 2179

Ev Adresi : Yaprak Sokak Kumru Apt No:17-2
Acıbadem-İSTANBUL

İş Adresi : Tübitak Marmara Araştırma Merkezi Teknoloji Serbest Bölgesi
Yeni Teknoloji Binaları A Blok-No:207-210 Gebze - KOCAELİ