BBL 503
BİLGİSAYAR MİMARİSİNDE YENİ YAKLAŞIMLAR

DÖNEM ÖDEVİ

HAREKET TABANLI BELLEK (TRANSACTIONAL MEMORY)

Ayşe Genç
504061504

Prof Dr. Bülent Örencik
10/12/2007

ÖZET **3**

HAREKET TABANLI BELLEK İLE İLGİLİ TEMEL KAVRAMLAR **5**

HAREKET 5
VERİ-VERSİYONLAMA 5
ÇAKIŞMALAR 5

HAREKET TABANLI BELLEK SİSTEMLERİ **8**

DONANIMSAL HAREKET TABANLI BELLEK (HARDWARE TM-HTM) 8
Hareketlerin soyutlanması: *8*
Veri-versiyonlama : *9*
Çakışma Sezme ve Önleme : *9*
Bilinen Sorunlar/Zayıf Yönler : *9*
YAZILIMSAL HAREKET TABANLI BELLEK (SOFTWARE TM-STM) 9
Hareketlerin soyutlanması: *10*
Veri-versiyonlama : *10*
Çakışma Sezme ve Önleme : *11*
Bilinen Sorunlar/Zayıf Yönler : *11*

BÜTÜNLEŞİK HAREKET TABANLI BELLEK **12**

DONANIM DESTEĞİ 12
Güncelleme uyarısı (alert-on-update) *12*
Programlanabilir Veri Soyutlama (programmable data isolation) *13*
Cep Etiketi Kodlaması *15*
YAZILIM DESTEĞİ 15
BTHB Uygulama Programı Arabirimi *15*
Metadata *16*
BÜTÜNLEŞİK HAREKET TABANLI BELLEK SİSTEMİ 17

KAYNAKÇA **18**

Özet

Çok çekirdekli ve çok işlemcili sistemler yaygınlaştıkça paralel programlamanın da önü açılmıştır. Bu sistemler aynı belleği paylaşıyorlarsa, paralel işçik yürütme sırasında paylaşılan belleğe erişim durumlarında ortaya çıkabilecek sorunlara senkronizasyon ya da yeniden sıralama mekanizmalarının uygulanması gerekmektedir. Geleneksel çok işcikli programlamada karşılıklı dışlama(mutual exclusion) garantisi veren kilit(lock) mekanizması kullanılır. Kilitler programcı için karmaşık ve hata yapmaya açıktır. Hem ölümcül kilitlenmelerin(deadlock), veri tutarsızlıklarının önüne geçmeye çalışıp hem de yüksek düzeyde paralellik içeren program yazmanın zorluğu ve paralel sistemin performansı üzerindeki etkisi ilgili çevrelerde kabul edilmiş bir gerçektir.

Kilit tabanlı senkronizasyona alternatif olarak Hareket Tabanlı Bellek mekanizması, paylaşılan belleğe/veriye eşzamanlı erişimin karmaşıklığını bir çeşit soyutlama yöntemiyle ortadan kaldırıyor. Böylece aynı anda pek çok işçiğin aynı paylaşılan bellek alanına erişimine izin veriliyor ve işçiğin tamamlanıp tamamlanması durumlarına göre ayarlamalar yapılıyor. Hareket Tabanlı Belleklerde *atomik hareket(atomic transaction)* yapısı kullanılıyor. Geleneksel sistemlerdeki kilitlerin yerine programcı kritik bölümleri *atomik* olarak belirlemekle yetiniyor ve Hareket Tabanlı Bellek uygulamayı çalışma zamanında o kritik bölümün diğer işçiklerden etkilenmeden yürütülmesini garanti ediyor.

```
atomic{
        hist[index++];
}
```

Yukarıdaki kod örneğinde klasik yaklaşımın semafor yapısını kullanırsak, *hist* dizisinin tüm elemanlarına erişim iznini sadece tek bir işçiğe vermiş oluruz, bu durumda dizinin farklı elemanlarına erişmek isteyen diğer işçikler bekletilir. Aslında dizinin aynı elemanına erişmek isteyen işcik yoksa, diziye erişimin kısıtlanmamış olması gerekir. Hareket tabanlı belleklerdeki *atomic* yapısı sayesinde bu sorun aşılmıştır.

Hareket Tabanlı Bellekler paralel yürütülen hareketlerin paylaşılan bellek erişimlerinde bir çakışma olmayacağı iyimserliği üzerine kurulmuştur.

Eğer hareketler herhangi bir çakışmaya neden olmamışsa, veri üzerinde bir karşılıklı dışlama kilidi koymakla uğraşmaktan kurtulmuş oluruz. Ancak hareketler herhangi bir çakışmaya neden olursa çakışmaya yol açan hareketlerden birini kesmek(abandon) zorunda kalırız, bu durumda kesilen hareketin o zamana kadar yürüttüğü işlemlerin diğer işçiklere görünmez kılınması yani onlardan etkilenmemelerinin sağlanması gerekir. Hareket Tabanlı Bellek yapısı üzerinde bu çakışmaları sezmek ve kesilen hareketin etkilerini toplamak için çeşitli mekanizmalar geliştirilmektedir.

Gerçekten de çoğu uygulamada hareketlerin çakışmaya yol açması az rastlanır bir durum olduğundan Hareket Tabanlı Bellek yapısının iyimser yaklaşımı gelecek programlama modelleri için kullanılacaktır. Hareket Tabanlı Bellek yapısının kilitlere göre üstün yanlarını söyle sıralayabiliriz:

- Paralel program yazmayı kolaylaştırır, çünkü daha üst düzeyde bir soyutlama sağlar.
- Ölümcül Kilitlenmeler oluşmaz.

Zayıf yanları ise:

- Geri alınamaz işlemler hareket yapısı içinde yer alamaz. Yani aklımıza gelebilecek her türlü işlemi hareket içine alamayabiliriz.
- Kilit yapısına göre tasarlanmış algoritmalar hemen Hareket Tabanlı Bellek yapısına dçnüştürülemez, kilitlerin yerine atomik kullanmak bazı durumlarda beklenenden kötü sonuç verecektir.
- Hareket Tabanlı Bellek için programlama yapmak için gerekli birikimi arttırmak ve programlamayı daha net hale getirmek gerekmektedir.

Hareket Tabanlı Bellek ile ilgili temel kavramlar

Hareket

Hareket belleğe okuma yazma komutlarını da içeren bir komutlar kümesidir. Bir hareket için iki durum söz konusudur: ya içerdiği tüm komutları yürütmeyi tamamlar (commit) ya da hiç birşey yapmamış olur (kesilme-abort- durumunda). Bir hareket tamamlandığında gerçekleştirdiği tüm yazma işlemleri diğer işçiklerin kullanımına hazır olur, ama eğer kesilirse sistem o ana kadar yaptığı tüm (yazma) işlemleri geri alır. Kesilen hareket daha sonra tekrar yürütülecektir.

Veri-versiyonlama

Hareket Tabanlı Bellekler yazma işlemleri ile ilgili şüpheli(speculative) durumları kotarmak için bir veri-versiyon (data-versioning) sistemine ihtiyaç duyarlar. Hareketin tamamlanması ya da kesilmesi durumunda veriyi diğer işçiklere görünür kılmak ya da eski değerini koruyabilmek için bu veri-versiyon sisteminden yararlanır. İki tip veri-versiyon uygulaması vardır:

1. Geri-al Günlüğü (undo log)
2. Güncelleme tamponu kullanımı (buffered updates)

Geri-al Günlüğü kullanımında hareketin gerektirdiği tüm güncellemeler doğrudan ilgili bellek alanlarına yapılır ve her biri için bir kayıt tutulur ki kesilme durumunda geri alınabilsinler.

Güncelleme Tamponu kullanımında ise hareketin gerektirdiği tüm şüpheli güncellemeler harekete özel tampomlarda tutulur ve hareketin tamamlanması halinde güncellemeler gerçekleştirilir.

Çakışmalar

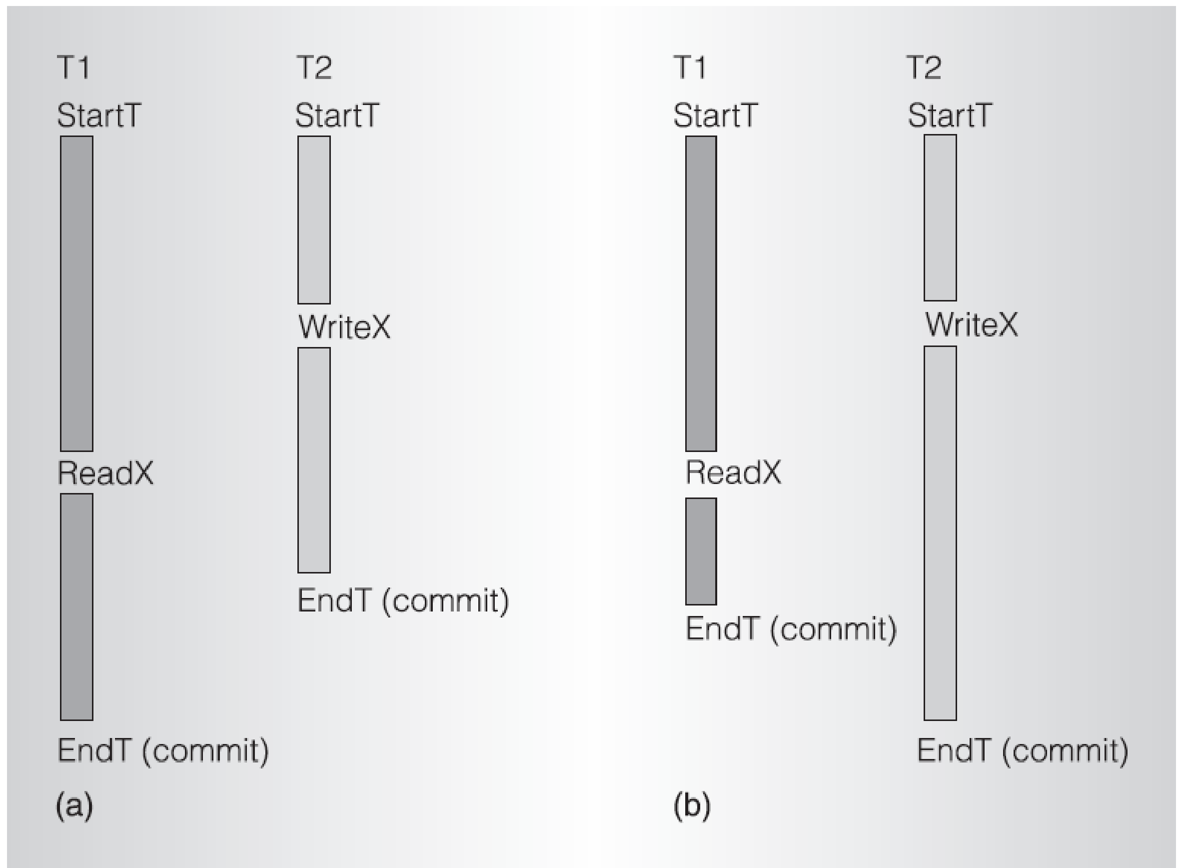
Hareket Tabanlı Belleklerde bir hareketin kesilmesi iki şekilde olabilir: çakışma durumunun sezilmesi halinde ya da hareketin kesilmesi için özel bir komut kullanarak. Çakışma durumları ile ilgili çakışmayı önceden sezme (detect) ve bu çakışma durumunu ortadan kaldırma (resolution) üzerine çalışmalar yapılmaktadır. Çakışma durumlarını kotarabilmek için her hareket bir *yazma kümesi(write set)* ve *okuma kümesi (read set)* ile ilişkilendirilmiştir. Harekette her bellekten veri alma (load) komutu için bellek adresi okuma kümesine yazılır. Benzer şekilde her yazma (store) komutunda da bellek adresi ve yazılacak veri yazma kümesinde tutulur.

Çakışmayı önceden sezmek için iki metod kullanılmaktadır:

1. sabırsız sezme (eager detection)
2. uyuşuk sezme (lazy detection)

Sabırsız sezme yönteminde, diğer işçiklerle bir çakışmaya sebep olmamak için bir hareketteki tüm okuma yazma kümeleri kontrol edilir. Sabırsız sezme yönteminin kullanılması için sistemdeki tüm hareketlerinin okuma ve yazma kümelerinin birbirlerine görünür olması gerekmektedir.

Uyuşuk sezme yönteminde, hareket tamamlanacak noktaya gelene kadar beklenir ve ondan sonra okuma yazma kümeleri diğer işçiğin yazma kümeleri ile karşılaştırılır.

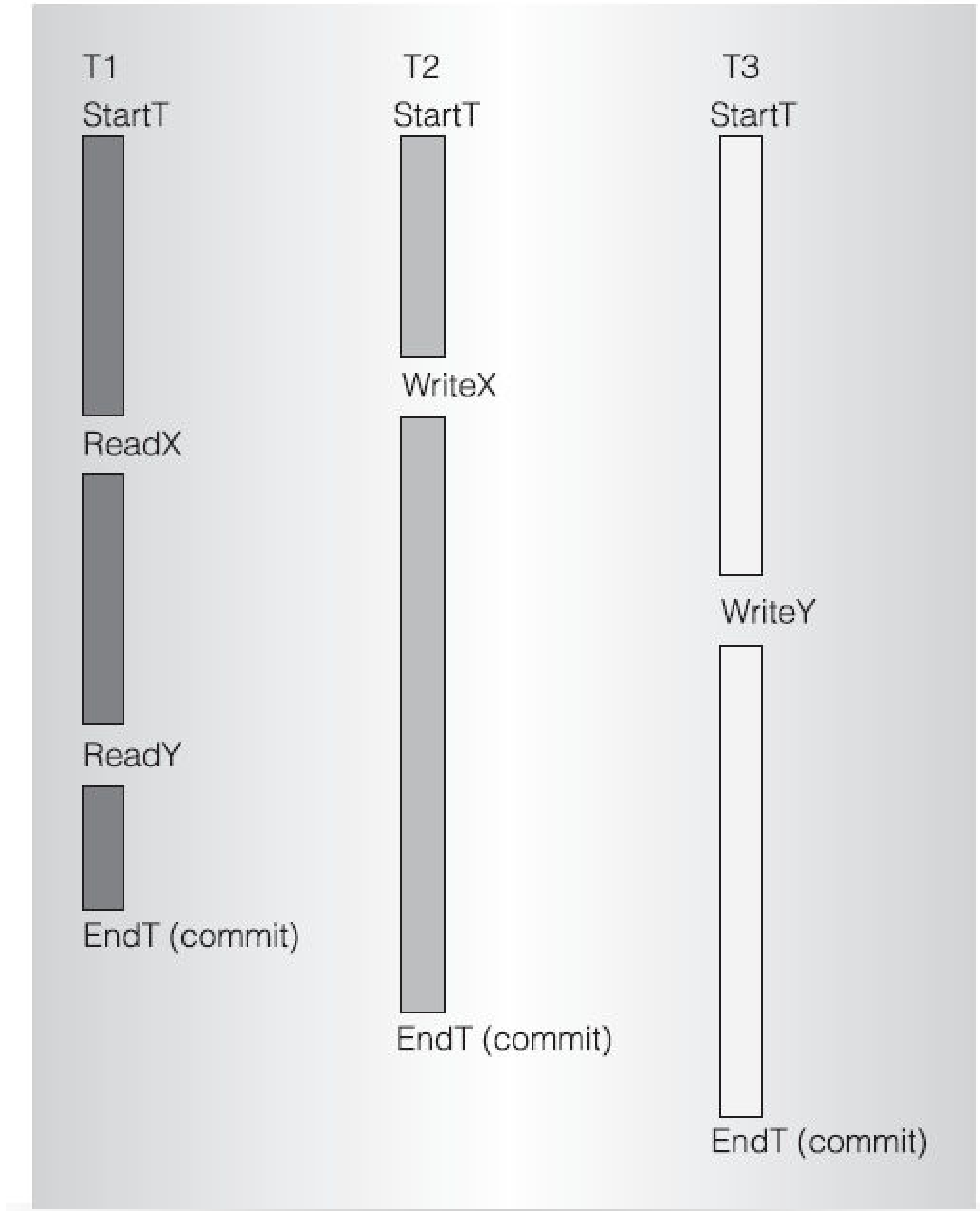


Şekil 1

Şekil1 üzerindeki hareketlerde sabırsız ve uyuşuk çakışma sezme yöntemlerini nasıl farklı davrandığını görebiliriz.
Şekil1a 'da X adresine okuma ve yazma şeklinde erişmek isteyen iki ayrı hareketin varlığı sabırsız sezme yönteminin bir çakışma olacağını belirlemesini sağlar. Aynı şekilde T2 hareketinin önce sonlanması yazma işleminin gerçekleşeği anlamına geldiğinden ve T1in X adresini okurken T2den önceki ya da sonraki değeri okuyabileceği ihtimalini ortaya çıkarır ve uyuşuk sezme yöntemi de bir çakışma olacağını belirtir.

Şekil 1b, sabırsız sezme yöntemine göre yine çakışma olacağı kararını verir. Ancak uyuşuk sezme yöntemine göre önce tamamlanacak olan hareket T1 olacağı için T2nin yaptığı değişiklikten etkilenmeyecektir, bu durumda her iki hareketin de sonlanmasına izin verilir.

Çakışma olacağı sezildikten sonra bulunan çözüm genellikle çakışmaya neden olan hareketlerden birinin kesilmesi olacaktır. Ancak hangi hareketin kesilmesi hangi hareketin tamamlanması gerektiğine karar vermek de karmaşık bir problemdir.



Şekil2

Şekil2 de gördüğümüz üç harekette, uyuşuk sezme yöntemine göre sorun çıkmamakla birlikte sabırsız sezme yöntemi kullanılırsa çakışma olacağı nedeniyle bir çözüm arayışına gidilecektir. T1 ve T2 hareketleri aynı X alanına, T1 ve T3 hareketleri de aynı Y alanına erişmektedir. T1 ve T2 arasında çakışma olacağı belirlendiğinde, sistem T1 ve T2den birinin kesilmesine karar verecektir. T2yi kesersek, daha sonra T1 ve T3 arasındaki çakışma dolayısıyla yine bir kesme işlemi gerekecektir. Böyle bir durumda en etkin olan çözüm T1i kesip, T2 ve T3ün tamamlanmasına izin verecek bir çözüm mekanizmasıdır.

Hareket Tabanlı Bellek Sistemleri

1. ***Donanımsal Hareket Tabanlı Bellek*** (HTM): ilk Hareket Tabanlı Bellekler donanım temelli olarak gerçeklenmiştir.
2. ***Yazılımsal Hareket Tabanlı Bellek*** (STM): donanımsal hiçbir değişikliğe gerek duymayan sistemlerdir
3. ***Karma Hareket Tabanlı Bellek***: Donanım temelli olup yazılımdan da destek alan (HyTM-Hybrid TM) ya da Yazılım temelli olup donanımdan da destek alan (HaSTM-Hardware assisted TM) çeşitleri vardır.

Bu sistemlerden her biri farklı sistemler için uygun olabilir, kullanılacak sistemde gerek duyulan hareketlerin özelliklerine ve gereken performans ihtiyaçlarına göre değerlendirmek gerekir.

Donanımsal Hareket Tabanlı Bellek (Hardware TM-HTM)

İlk HTMler donanım temelli gerçekleştirilmiştir. Minimalist bir yaklaşımla tasarlanmış olmalarına rağmen tamamen işlevseldiler. Cep tutarlılık protokolü (cache consistency protocol) ve komut kümesi mimarisi (instruction set architecture-ISA) üzerinde yapılan küçük değişikliklerle ve güncelleme tamponu kullanımıyla basit bir Hareket Tabanlı Bellek oluşturulmuştur.

Daha sonra ISA ya da cep üzerinde çok daha ufak değişikliklerle oluşturulmuş Hareket Tabanlı Bellekler de gerçeklemiştir.

Hareketlerin soyutlanması:

Hareketler için ISA üzerindeki değişikliklerle bir kaç komut eklenmiştir:

- STR (start transaction) : hareket başlangıcı
- ETR (end transaction) : hareket sonu
- TLD : hareket için load komutunun özel bir versiyonu
- TST: hareket için store komutunun özel bir versiyonu
- ABR(abort) : hareketin kesilmesi için
- VLD (validation): geçerlilik

Tüm komutlar her sistem için kullanılmak zorunda değildir. Hiç STR kullanmadan hareket içindeki ilk TLD/TSTyi baz alan yapılar vardır.

ABR komutu yürütülen program içinde performans gibi bir problemi çözmek için biliçli olarak kesilecek bir hareket için kullanılabilir. VLD çakışmaları önceden yakalamak için kullanılabilir.

Veri-versiyonlama :

Veri-versiyonlama güncelleme tamponu tutularak gerçekleştirilir. Güncelleme tamponu veri cebinde ya da donanım tampon alanında tutulur.

Çakışma Sezme ve Önleme :

Çakışmaları sezmek ve çözüm bulmak için MESI cep tutarlılık protokolü kullanılır. Bu protokol için fazladan iki bit eklenmesi yeterlidir. MESI protokolü, ilgili blokla (cache line) ilgili şu bilgileri tutar:

- M (modified) : cepteki veri güncellenmiş ve bellektekinden farklı. Bellek güncellemesi gerekir.
- E (exclusive): cepteki veri güncellenmemiş, bellektekiyle aynı
- S (shared): paylaşılan bellek alanı
- I (invalid): cepteki veri geçersiz, belleğe yazılmamalı

Bu bilgiler ışığında hareket tamamlanır ya da kesilirse, sistemin nasıl davranacağı açıktır.

Bilinen Sorunlar/Zayıf Yönler :

- Hareketler için cepin kapasitesi gibi bir boyur sınırı vardır. (bounded) Bu sorunu aşmak için Sınırsız Boyutlu TM[2] (unbounded TM) yapısı önerilmiştir. UTM ile hareketlerin boyutu sanal bellek kapasitesine kadar çıkabilmektedir.

Yazılımsal Hareket Tabanlı Bellek (Software TM-STM)

STMler küme bellek (heap) üzerinde her hareket için ayrı bir görüntü (view) alanı oluşturmak zorundadır. Bu görüntü, hareketin yürütülmesi sırasında oluşan değişiklikleri görüp kullanabilmesini sağlamalı ve hareketin tamamlanması durumunda bellek güncellemesine izin verirken kesilmesi durumunda tüm değişiklikleri geri alabilmelidir. Ayrıca çakışma durumlarını sezmek ve bir çözüm üretebilmek için de mekanizmalar geliştirilmelidir.

STMler gerek hareket boyu kısıtlaması olmaması gerek de programlama konusunda getirdiği esneklikler nedeniyle HTMlere tercih edilir.

Hareketlerin soyutlanması:

Soyutlama için iki farklı yaklaşım vardır:

1. Hareketten etkilenecek veriler için ayrı bir veri yapısı tutma yaklaşımı : Bu yaklaşımda kullanılan nesnelere başlıklar(header) eklenir ve bu başlıklarda ilgili hareket numarası ve erişim şekli (okuma/yazma olarak) belirtilir.
2. Veri yapısını değiştirmeden harekete özel metadataların STM tarafından kotarılması.

Kullanıcı için kritik bölümün *atomic* ile belirtilmesi yeterlidir. STMler diller, derleyiciler ve kütüphaneler yardımıyla bu atomik yapıyı destekler.(Haskell derleyicisi, Atomus, Argus ve C# dilleri..)

```
atomic {
   hist[index]++;
}
(a)

atomic {
   ...
   OpenForReading(tr, TMW_FOR(index));
   OpenForWriting(tr, TMW_FOR(hist));
   int *addr = &hist[STMRead(tr, &index)];
   STMWrite(tr, addr, STMRead(tr, addr) + 1);
   ...
}
(b)
```

Şekil3.

Örneğin *Şekil3* için *a* şıkkında kullanıcı *atomic* yapısı ile kritik bölümü belirtmiştir. *b* şıkkı ise derleyicinin yapması gereken izdüşümü(mapping) göstermektedir.

Veri-versiyonlama :

Veri versiyonlamak için güncelleme tamponu kullanılıyorsa, hareket içinde güncellenen her veriye özel bir gölge kopya (private shadow copy) tutar, aynı zamanda okuma işlemleri de bu özel gölge kopyaya başvurur, bir arama yapılır ve

okumak istediği adresina daha önce bu hareket tarafından değiştirilmesi durumunu es geçmemiş olur.

Güncelleme tamponu kullanımında okuma işlemleri için gölge kopyanın aranması işleminin yaratacağı zaman kaybı çeşitli yöntemlerle azaltılmış olsa da hala problem olabilir. Bu durumda, güncelleme tamponu yerine geri-al günlüğü kullanan sistemler daha elverişli olabilir. Veri güncelleneceği zaman doğrudan küme bellek (heap) üzerinde değişiklik yapılır ve bu değişiklik günlüğe kaydedilir. Hareketin kesilmesi durumunda bu günlük kullanılarak küme bellek eski haline getirilir. Geri-al günlüğü kullanımının da yarış durumu (race-condition) ortaya çıkarma riski vardır.

Çakışma Sezme ve Önleme :

Hareket verisi kilidi (Lock transactional data), Engelsiz(nonblocking) ve karma (hybrid) gibi yaklaşımlar vardır.

Engelsizlik yaklaşımında hareketin veri yapısının yazma kümesindeki günceleme bilgisini kontrol edilerek, güncelleme gerekiyorsa belleğe yansıtılır. Herhangi bir kilit uygulaması söz konusu değildir, ölümcül kilitlenmeyle karşılaşılmaz.

Hareket tabanlı bellekler üzerindeki çalışmalar arttıkça, ölümcül kilitlenmeleri önlemenin tek yolunun engelsiz (nonblocking) çalışma olmadığı görülmüş ve kilit yapısını kullanan sistemler önerilmiştir. [3] Hareket verisi kilidi kullanımında, hareketin paylaşılan bellek alanı üzerinde etkilediği kısımlar kilitlenir, böylece başka hareketlerin erişimine izin verilmez. Kilitleme iki türlü yapılır: tüm hareket süresince kilitleme ya da hareketin tamamlanması sırasında (commit-time) kilitleme. Hareket verisi kilidi kullanımının STMler üzerinde performans açısından yararı görülmüştür.

Bilinen Sorunlar/Zayıf Yönler :

- Büyük boyuttaki hareketler sistemi çok yorarak hantallaşmasına neden olmaktadır.

Bütünleşik Hareket Tabanlı Bellek

Hareket Tabanı Bellek uygulamalarını yazılım ve donanımsal olarak desteklemek üzere pek çok sistem geliştirilmiştir. *An Integrated Hardware-Software Approach to Flexible Transactional Memory*[4] adlı makalede anlatılan sistem yazılımsal ve donanımsal sistemlerin üstün yönlerini birleştirerek optimum bir çözüm sunmuştur, bu sistem kısaca *Bütünleşik Hareket Tabanlı Bellek (BHTB)* olarak anlatılacaktır.

BTHB, temelde yazılımsal hareket tabanlı bellek olmakla birlikte daha hızlı çalışması için donanımla desteklenmiş bir sistemdir. Donanım, sınırlı boyutlu hareketler (bounded transactions) için çakışma sezmeyi ve veri soyutlamayı hızlandırma ve sistem üzerindeki yükünü azaltma desteği sağlar. Yazılım ise veri-versiyonlama, çakışma sezme politikalarının esnekleştirilmesinde ve sınırsız boyutlu hareket (unbounded transactions) desteğinde kullanılmaktadır.

Donanım Desteği

Donanım desteği içerdiği mekanizmalar ile:

1. güncelleme uyarısı (alert-on-update) ile çakışma denetiminde hızlı davranmaya
2. programlanabilir veri soyutlama (programmable data isolation) desteği ile de eş zamanlı olarak pek çok hareketin okuma yazma işlemini tamamlama/kesilme durumlarını hızlı sonuçlandırmasına

yardımcı olur.

Güncelleme uyarısı (alert-on-update)

Çakışma sezmek için her işçik uyarı denetimcisine (alert handler) kaydolur ve cepte ilgilendiği bloğu AOU (alert-on-update) olarak işaretler. Cep denetleyicisi işaretlenmiş bloğu boşaltmak istediğinde ya da yeni bir yazma yapılacağı zaman ilgili işçiğe uyarı gönderilir.

Gönderilen uyarıda, uyarıya yol açan sebep ve ilgili bloğun adresi gönderilir. Ayrıca denetimciye kaydolmak, blokları işaretlemek, işareti kaldırmak gibi işlemler için yeni komutların da eklenmesi gerekir. Cep işaretlemek için 1 bit yeterlidir.

Registers	
%aou_handlerPC:	address of the handler to be called on a user-space alert
%aou_oldPC:	program counter immediately prior to the call to %aou_handlerPC
%aou_alertAddress:	address of the line whose status change caused the alert
%aou_alertType:	remote_write, lost_alert, or capacity/conflict eviction
interrupt vector table	one extra entry to hold the address of the handler for kernel-mode alerts
Instructions	
set_handler %r	move %r into %aou_handlerPC
clear_handler	clear %aou_handlerPC and flash-clear the alert bits for all cache lines
aload [%r1], %r2	load the word at address %r1 into register %r2, and set the alert bit(s) for the corresponding cache line
arelease %r	unset the alert bit for the cache line that corresponds to the address in register %r
arelease_all	flash-clear alert bits on all cache lines
Cache	
one extra bit per line, orthogonal to the usual state bits	

Tablo1: Güncelleme uyarısı için geren donanım listesi

*Tablo1*de gereken donanım sıralanmıştır. *Aload* komutu cep bloğunu işaretlenmek için kullanılır. Bunu yanısıra her hareket *Aload* komutunu yürüterek işaretlediği ve güncel durumunu öğrendiği sözcüğün (word) durumunun değişmesi durumunda uyarılır.

Programlanabilir Veri Soyutlama (programmable data isolation)

Çoğu donanımsal hareket tabanlı bellkete olduğu gibi BHTBde de hareketlerin bir güncelleme tamponunda tutulup hareket tamamlandıktan sonra belleği güncellemesi ve diğer hareketlere görünür kılınması desteklenmektedir. Ancak farklı olarak, güncelleme tamponu işlemciye ana cepten daha yakın olan ikinci bir cepte tutulur.

Geleneksel MESI tutarlılık protokolü kullanılmaktadır ancak donanımsal değişikliklerden dolayı TMESI olarak adlandırılmıştır. Eklenen T, okumak için ilgili adrese erişen harekete potansiyel değişiklik konusunda uyarır. *Tablo2*de TMESI için gerekli donanımın listesi verilmiştir.

Registers	
`%t_in_flight:`	a bit to indicate that a transaction is currently executing
Instructions	
`begin_t`	set the `%t_in_flight` register to indicate the start of a transaction
`tstore [%r1], %r2`	write the value in register `%r2` to the word at address `%r1`; isolate the line (*TMI* state)
`tload [%r1], %r2`	read the word at address `%r1`, place the value in register `%r2`, and tag the line as transactional
`abort`	discard all isolated (*TMI* or *TI*) lines; clear all transactional tags and reset the `%t_in_flight` register
`cas-commit [%r1], %r2, %r3`	compare `%r2` to the word at address `%r1`; if they match, commit all isolated writes (*TMI* lines) and store `%r3` to the word; otherwise discard all isolated writes; in either case, clear all transactional tags, discard all isolated reads (*TI* lines), and reset the `%t_in_flight` register
Cache	
two extra stable states, TMI and TI, for isolated reads and writes; transactional tag for the MES states	

Tablo2 : programlanabilir veri soyutlama için gereken donanım listesi

Şüpheli okuma ve yazma işlemleri *Tload* ve *Tstore* komutları ile gerçekleştirilir. Hareketin yürütülüp yürütülmediğini gösteren bir saklayıcı (%t_in_flight) mevcuttur ve okuma/yazma işleminin şüpheli olup olmadığı bu saklayıcının değerine göre kararlaştırılır. Şüpheli okuma/yazma komutları kullanıldığı zaman *TI* ve *TMI* denilen iki tutarlılık durumu devreye girer ve yazılım politikası gereği seçilen çakışma sezme yöntemi ile okuma-yazma ve yazma-yazma çakışmaları kontrol edilir.

TMI, tampon alandaki şüpheli yazma işlemlerine hizmet eder, yani Tstore komutundan sonra blok TMI moduna geçer. Eğer hareket tamamlanırsa M, kesilirse I olarak güncellenir. Uyuşuk sezme yöntemi kullanıldığında, okuma-yazma çakışması durumunun problem olmaması için okuma işlemini yapan hareketin yazma işlemini yapan hareketten daha önce tamamlanması gerekmektedir, ve bu sistemde bu yazılımın sorumluluğundadır. Yazma-yazma çakışması durumunda zaten sadece bir hareket tamamlanacaktır, diğer(ler)i kesilir ve daha sonra tekrar yürütülür.

TI, o anki hareket tarafından okunan ve diğer hareketlerden tarafından üzerine yazılmış olabilecek verilerin kullanılmasına olanak sağlar. I modunda olan bir cep bloğu *Tload* komutu kullanımıyla, M S veya E modunda olan bir cep bloğu da başka bir hareket tarafından yazma işlemi yürütüldüğünde TI moduna geçer. Hareket tamamlansa da kesilse de TI modundaki bir cep bloğu I olarak işaretlenecektir, çünkü şüpheli bir durum oluşmuştur. TI sabırsız çakışma sezme yöntemi kullanıldığında anlamlıdır, eğer yazılım uyuşuk sezme yöntemine karar verdiyse TI kullanımı gereksiz olacaktır.

CSA-Commit komutu bir karşılaştır-ve-takasla uygulamasıdır. Eğer CSA başarıyla sonuçlanırsa şüpheli yazma (TMI) olarak işaretlenen cep bloğu M olarak işaretlenir, böylece yapılan değişiklik herkese görünür hale gelir. Eğer CSA başarısızlıkla sonuçlanırsa, TMI geçersiz kılınır ve yazılım hareketi kesmek üzere dallanır.

Cep Etiketi Kodlaması

Normalde MESI protokolü ile M,S,E,I durumları varken TMESI ile AOU ve harekete bağlı TI, TMI , TS,TM,TE durumları ve CAS karşılaştırmasının sonucu da eklenince durum sayısı dolayısıyla ifade etmek kullanılması gereken bit sayısı artıyor. Bu durumda hareketin tamamlanma/kesilme durumlarını daha hızlı kotarabilmek için bir kodlama sistemi kullanılmıştır. Kodlama sonucu oluşan *Tablo3* aşağıda verilmiştir.

T	A	MESI	C/A	M/I	State
0	—	00	—	—	I
0	—	11	0	1	I
0	—	01	—	—	S
0	—	10	—	—	E
0	—	11	1	—	M
0	—	11	0	0	M
1	—	00	—	—	TI
1	—	01	—	—	TS
1	—	10	—	—	TE
1	—	11	—	0	TM
1	—	11	—	1	TMI

T	Line is (1) / is not (0) transactional
A	Line is (1) / is not (0) alert-on-update
MESI	2 bits: I (00), S (01), E (10), or M (11)
C/A	Most recent txn committed (1) or aborted (0)
M/I	Line is/was in TMI (1)

Tablo3

Yazılım Desteği

BHTB, Rochester Software Transactional Memory[8] (RSTM) tabanlı geliştirilmiştir ve RSTMnin tüm kütüphaneleri ile tamamen uyumludur.

Bütünleşik Hareket Tabanlı Bellek, donanımsal destekleyicileri olan güncelleme uyarısı ve programlanabilir veri soyutlama sistemlerini kullanırken hareketlerin hızlı işlenmesini sağlar. Bunun yanısıra eğer bir hareketin tamamlanması için gerekli süre uzarsa ya da hareket ALoad veya Tstore kapasitesini aşarsa, hareket baştan başlatılır, yazılım "taşma modu"nda ve zaman/boyut sınırlaması omaksızın çalıştırılır.

BTHB Uygulama Programı Arabirimi

Hareketler BEGIN_TRANSACTION ve END_TRANSACTION makroları arasında belirlenir. BEGIN_TRANSACTION ile uyarı denetimcisi kurulur ve END_TRANSACTION ile CAS-Commit yürüterek hareket tamamlanır ya da kesilir.

Hareketler sistemde genel olarak kullanılan hareket nesnelerinden türetilirler ve bu sayede harekete özgü alanlara güvenli erişimi garanti ederler. *open_RO* ve *open_RW* çağrıları sayesinde programcı hareket alanlarına ulaşım isteğinde bulunur ve sonuç olarak o harekete bir işaretçi elde eder.

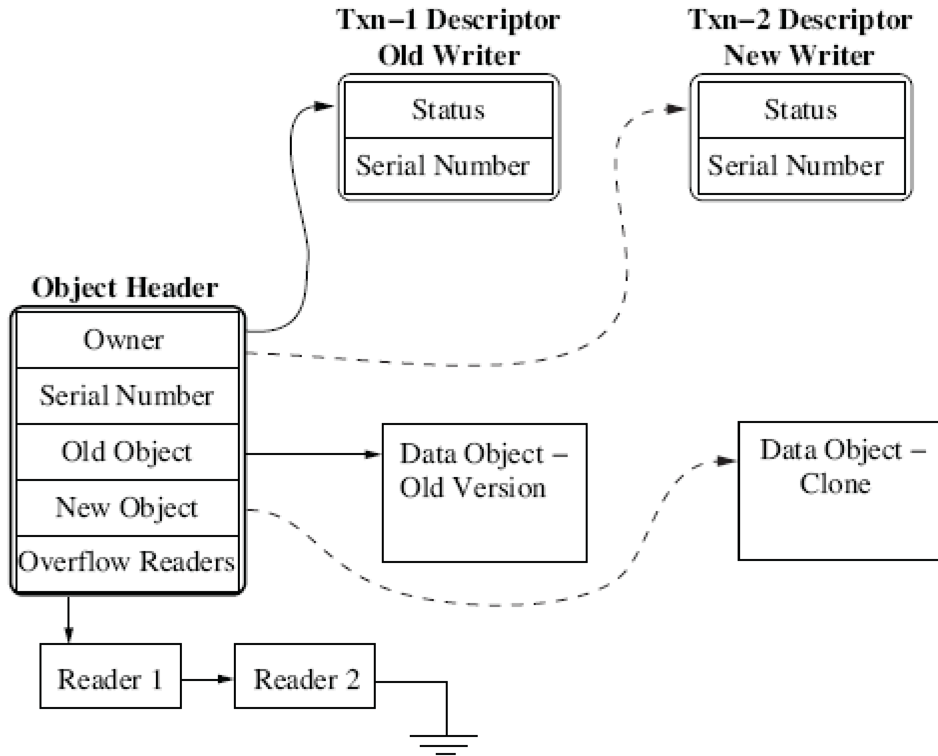
Metadata

Her hareket bir tanımlayıcı (descriptor) ile gösterilir. Bu tanımlayıcı bir seri numarası ve hareketin durumunu *Aktif, Tamamlandı* ya da *Kesildi* şeklinde belirten sözcükten oluşur.

Her hareket nesnesi bir başlık içerir ve bu başlıkta :

1. sahip harekete bir işaretçi
2. sahibinin seri numarası
3. eski veriye işaretçi
4. yeni/şüpheli verilere işaretçi
5. nesneyi okuyan hareketler listesine işaretçi

Tanımlayıcı ve nesne başlıklarının yapısı aşağıdaki şekil4te görüldüğü gibidir.



Şekil 4 – BHTB metadata yapısı

open_RO ve open_RW çağrıları ile bu tanımlayıcı ve başlık bilgilerinde yapılan güncellemeler sayesinde veri bilgisi güncellenir ya da eski haline geri döndürülür.

Bütünleşik Hareket Tabanlı Bellek Sistemi

Donanım ve yazılım desteği sonunda elde edilen Bütünleşik Hareket Tabanlı bellek sistemi genel olarak şu özellikleri sağlamaktadır:

1. Engelsiz (nonblocking) çalışır.
2. Özel hareket veri nesneleri içerir.
3. Çakışma sezme için sabırsız/uyuşuk arasında geçiş yapma esnekliğine sahip, aynı zamanda donanımdan gelen uyarıları dinleyebilir.
4. Hareket boyu için sınırsız kapasite kullanabilir.
5. Tamamlanma/kesilme durumlarını hızlı kotarır.
6. Güncelleme tamponu olarak CPUya daha yakın ikinci bir cep ve hareket nesnelerinin kopyasını kullanabilen.
7. Donanım özellikleri kullanarak çalışmaya başlaması öngörülmüş fakat donanımın yetersiz kaldığı durumları tespit edip yazılımsal destekleri kullanmaya başlayabilir.

Kaynakça

1. Harris T., Cristal A., Unsal O.S., Ayguade E., Gagliardi F., Smith B., Valero M., "*Transactional Memory: An Overview*" , IEEE Computer Society publications, May-June 2007.
2. Ananian C. S., Asanovic K. , Kuszmaul B. C., Leiserson C. E. , Lie S.,"*Unbounded transactional Memory*", Proceedings of the 11th Int'l Symposium on High-Performance Computer Architecture, IEEE,2005.
3. Dice D., Shavit N.,"*Understanding tradeoffs in Software Transactional memory*", International Symposium on Code Generation and Optimization, IEEE,2007
4. Shriraman A., Spear M. F., Hossain H. , Marathe V. J., Dwarkadas S. , Scott M. L., *"An Integrated Hardware-Software Approach to Flexible Transactional Memory"*, Department of Computer Science, University of Rochester, 2006
5. Blundell C., Lewis E. C. , Martin M. M. K., "*Subtleties of Transactional Memory Atomicity Semantics*",IEEE Computer Architecture Letters VOL. 5, NO. 2, 2006
6. "*Software Transactional Memory*", http://en.wikipedia.org/wiki/Software_transactional_memory
7. "*MESI protocol*", http://en.wikipedia.org/wiki/MESI_protocol
8. "*Rochester Software Transactional Memory*", http://www.cs.rochester.edu/research/synchronization/rstm/