GAZİ ÜNİVERSİTESİ MÜHENDSİLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

SELİN CANSU AKBAŞ

191180005

Araştırma Ödevi

PROF. DR. M. ALİ AKÇAYOL

Bilgisayar Mimarisi BM 311

Multicore ve Çok İşlemcili Sistemlerde Cache Coherence İçin Kullanılan Protokoller

İÇİNDEKİLER

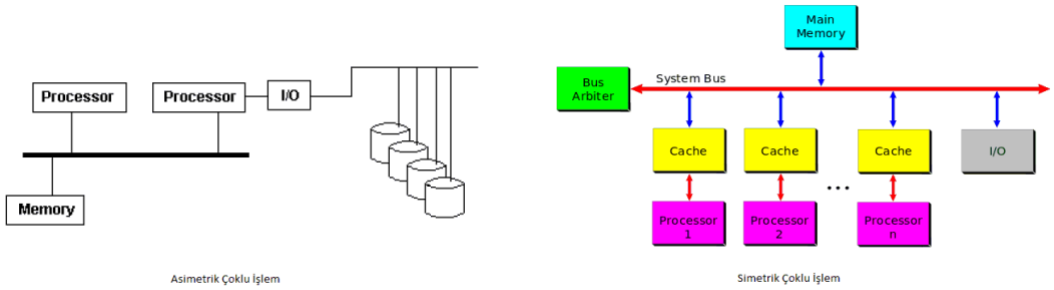
1. ÖZET

Önbellek tutarlılık mekanizmaları, yaygın iş parçacığı düzeyinde paralellik yoluyla üstel performans artışını sürdürme hedefine ulaşmada kilit bir bileşendir. Tutarlılık, bir sistem içindeki tüm işlemcilerin veya veri yolu yöneticilerinin aynı paylaşılan bellek görünümüne sahip olmasını sağlamak anlamına gelir. Cache Coherence olayını özetle böyle açıklayabiliriz. Kısaca veriler arasında tutarlılığı sağlar. İşlemciler mantıksal olarak aynı belleğe erişse de, çip üzerindeki önbellek hiyerarşileri, işlemciler tarafından yapılan bellek referanslarının çoğunda hızlı performans elde etmek için çok önemlidir. Bu nedenle, paylaşılan bellek çok işlemcilerinin temel bir sorunu, çeşitli önbellek hiyerarşileriyle tutarlı bir bellek görünümü sağlamaktır. Önbellek tutarlılık mekanizmaları yalnızca paylaşılan bellekli bir çok işlemcideki iletişimi yönetmekle kalmaz, aynı zamanda tipik olarak bellek sisteminin verileri işlemciler, önbellekler ve bellek arasında nasıl aktardığını da belirler. Önbellek tutarlılığı probleminin en bilindik 4 çözüm protokolü vardır. Bunları, MSI Protokolü, MOSI Protokolü, MESI Protokolü, MOESI Protokolü olarak sıralayabiliriz. Bu protokoller modified, shared, exculusive, owned, invalid başlıkları ile incelenir. Ayrıca gelecekteki iş yükleri, önbellek tutarlı bellek sisteminin performansına bağlı olacaktır ve bu alanda devam eden yenilik, bilgisayar tasarımında ilerlemeye ve gelişmeye uygundur diyebiliriz.

2. CACHE COHERENCE

2.1. Çok İşlemcili Mimariler

Bilgisayar sistemlerindeki performansı iyileştirmek bazı durumlarda erişilebilirlik özelliğini arttırmak amacıyla çok sayıda işlevsel birim (ALU,PC,Memory,Floating-Point,vb.) paralel olarak çalıştırılır. Çok işlemcili mimarilerde aynı bilgisayar sistemi içerisinde 2 ya da daha fazla CPU kullanılarak performansı ve erişilebilirliği arttırmak amaçlanır. Performans artışı işlem havuzundaki işlemlerin farklı işlem ünitelerine dağıtılmasıyla gerçekleştirilir. Kısaca özetleyecek olursak çok işlemcili mimariye sahip bilgisayar sistemleri aynı ana bellek ve çevre birimlerini paylaşabilen iki veya daha fazla işlem birimi bulunduran bu CPU'lar için farklı cache belleklerin kullanılabildiği bilgisayar sistemleridir ve birden fazla işlemci aynı işin farklı kısımlarını gerçekleştirebilir. Çoklu işlemcili mimariler çoklu işlem mantığı üzerinden çalışır ve çoklu işlem simetrik ve asimetrik olarak iki başlıkta incelenir [1].



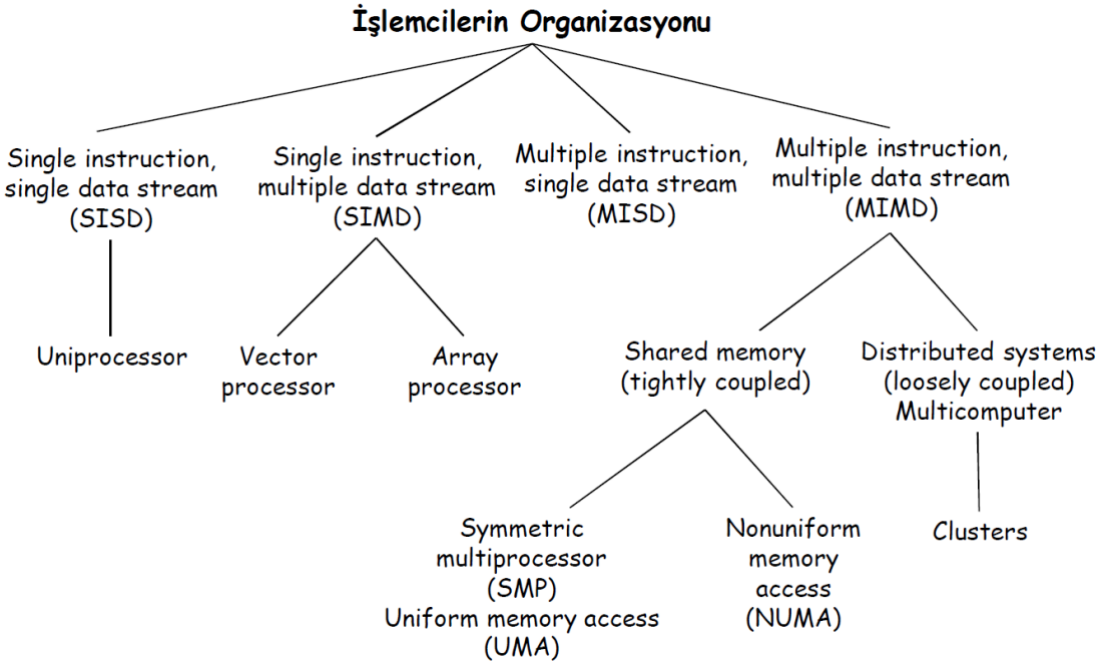
Şekil 2.1. Asimetrik ve Simetrik Çoklu İşlem Şekilleri

Simetrik çoklu işlem (SMP): İki veya daha fazla özdeş işlemcinin tek bir paylaşılan ana belleğe bağlandığı, tüm giriş ve çıkış aygıtlarına tam erişime sahip olduğu ve işleyen tek bir işletim sistemi örneği tarafından denetlendiği çok işlemci yapısıdır. Çok işlemcili bir bilgisayar donanımı ve yazılım mimarisi açısından tüm işlemcileri eşit, özel amaçlı olarak hiçbir şey ayırmadan kullanır.

Asimetrik çoklu işlem (AMP): Tüm CPU'lar eşit muamele görmez. Örneğin ilk CPU; donanım veya işletim sistemi düzeyinde bir sistem CPU'nun sistem kodunu çalıştırmasına ya da CPU'nun I/O işlemlerini gerçekleştirmesine izin verebilir, ikinci herhangi bir CPU'nun hem işletim sistemi kodunu çalıştırmasına hemde I / O işlemlerini gerçekleştirmesine izin verir. Böylece işlemci rolleri açısından simetrik olur, ancak bazı veya tüm çevre birimlerini belli CPU'lara bağlar; böylece çevre birimlere göre asimetrik olurlar.

2.2. Çok İşlemcili Mimarilerin Sınıflandırılması

Mikro işlemciler Şekil 2.2.'de de görüldüğü gibi farklı başlıklarda komut ve veri yönetimlerine göre ayrılmaktadırlar.



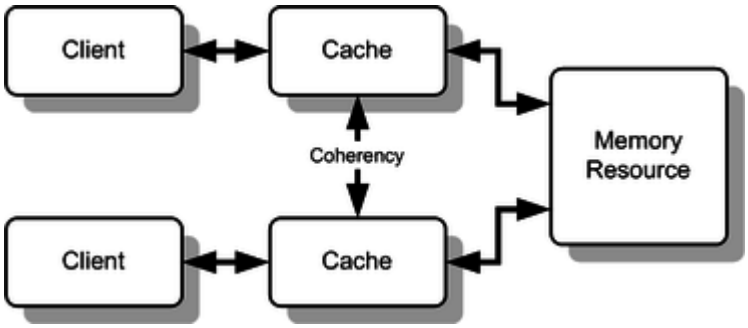
Şekil 2.2. İşlemcilerin Organizasyon Şeması

2.3. Cache Coherence Nedir?

Çok işlemcili sistemlerde aynı anda birden fazla işlemcinin çalışmasından dolayı işlemcilerin aynı veriyi aynı anda işleme olasılıkları vardır. Bu nedenle; işlemcilerden biri veriyi işlerse diğer işlemcinin o anda içeriği değişmiş veriye ulaşması riski vardır. Paylaşılan bellekteki verilerin değişiminden tüm işlemcilerin haberdar olması gerekir. Bunun için çok işlemcili sistemlerde hafızadan alınan veriler üzerinde herhangi bir değişiklik yapıldığında, yanlış veya hatalı sonuçları engellemek için bu değişikliklerin diğer işlemcilere bildirilmesi gerekir. Böylece cache ve memory arasındaki veriler hakkında bir tutarlılık sağlanır. Cache Coherence olayını tam olarak böyle açıklayabiliriz. Veriler arasında tutarlılığı sağlayan protokoldür [1].

Hafızadan bilgi okumuş/yazmış işlemciler yaptıkları bu değişiklikleri diğer işlemcilere bildirmesi gerekir. Bu bildirimler Cache Coherence trafiği oluştururlar. Bu trafiğin yoğunluğu sistemdeki işlemci sayısına bağlıdır. Trafik yoğunluğu sistemde kullanılan işlemci sayısının karesiyle doğru orantılıdır. Ancak trafik yoğunluğunun artması sistemlerin performansını kötü şekilde etkileyecektir.

Herhangi bir işlemci sistem belleğinin bir bloğunu saklarsa o kısım alınmış olarak işaretlenir. Alınan veri değişmiş veya değişeceği kesin ise; veriyi kullanan işlemci tarafından diğer işlemcilerin bu veriye ulaşmaması için ve yanlış veya hatalı işlemler yapılmaması için hafızadan alınan bu veriler işaretli olarak tutulur [2].



Şekil 2.3. Paylaşılan bir kaynak görevi gören bazı belleklerin birden çok önbelleğini gösteren şekil

Şekil 2.3.'teki resimde, her iki istemcinin de önceki bir okumadan belirli bir bellek bloğunun önbelleğe alınmış bir kopyasına sahip olduğunu düşünün. Alttaki istemcinin bu bellek bloğunu güncellediğini / değiştirdiğini, üstteki istemcinin değişiklik bildirimi yapılmadan geçersiz bir bellek önbelleğiyle bırakılabileceğini varsayalım. Önbellek tutarlılığı, birden çok önbellekteki veri değerlerinin tutarlı bir görünümünü koruyarak bu tür çakışmaları yönetmeyi amaçlar [3].

Cache coherence, eski bir veri varlığını önlemek için birden çok işlemcinin doğru talimatlara / verilere eriştiğinden emin olur. Bir özel önbellekteki verilerin değiştirilmesi, diğer özel önbellek bloklarında bulunan aynı verilerin tutarsızlıklarına yol açar. Bu da önbellek tutarsızlığına yol açar [4].

İçinde paylaşılan hafıza her işlemci için ayrı bir önbelleğe sahip çok işlemcili sistemde, paylaşılan verilerin birçok kopyasına sahip olmak mümkündür. Ana bellekte bir kopya ve bunu isteyen her işlemcinin yerel önbelleğinde bir kopya bulunur. Verilerin kopyalarından biri değiştirildiğinde, diğer kopyalar bu değişikliği yansıtmalıdır. Önbellek tutarlılığı, paylaşılan verilerin değerlerindeki değişikliklerin sistem genelinde zamanında yayılmasını sağlayan disiplindir [5].

2.4. Cache Coherence Protokolleri

Coherence protokolleri, çok işlemcili sistemlerde önbellek tutarlılığını uygular. Amaç, iki istemcinin aynı paylaşılan veriler için asla farklı değerler görmemesi gerektiğidir. Protokol, tutarlılık için temel gereksinimleri uygulamalıdır. Hedef sistem veya uygulama için özel olarak hazırlanabilir.

Protokoller snoopy veya dizin tabanlı olarak da sınıflandırılabilir. Tipik olarak, ilk sistemler, bir dizinin paylaşılan verileri ve paylaşanları izleyeceği dizin tabanlı protokoller kullanırdı. Snoopy protokollerinde, işlem istekleri (okuma, yazma veya yükseltme) tüm işlemcilere gönderilir. Tüm işlemciler isteği takip eder ve uygun şekilde yanıt verir.

Snoopy protokollerinde yazma yayılımı aşağıdaki yöntemlerden biriyle uygulanabilir:

Write-invalidate: Önbelleğin kopyasına sahip olduğu bir konuma yazma işlemi gözlemlendiğinde, önbellek denetleyicisi gözetlenen bellek konumunun kendi kopyasını geçersiz kılar ve bu da bir sonraki erişiminde yeni değerin ana belleğinden okumaya zorlar.

Write-update: Önbelleğin kopyasının bulunduğu bir konuma yazma işlemi gözlemlendiğinde, önbellek denetleyicisi gözetlenen bellek konumunun kendi kopyasını yeni verilerle günceller.

Protokol tasarımı, paylaşılan verilerin herhangi bir kopyası değiştirildiğinde, diğer tüm kopyaların değişikliği yansıtacak şekilde "güncellenmesi" gerektiğini belirtiyorsa, bu bir write-update protokolüdür. Tasarım, herhangi bir işlemci tarafından önbelleğe alınmış bir kopyaya yazmanın diğer işlemcilerin önbelleğe alınmış kopyalarını atmasını veya geçersiz kılmasını gerektirdiğini belirtiyorsa, bu bir write-invalidate protokolüdür [3].

Çok fazla protokol bulunmasına rağmen şu anda kullanılan ana protokoller R-MESİ tipi / MESİF protokolleri ve HRT-ST-MESİ (MOESİ tipi) veya bunların bir alt kümesi veya uzantısıdır.

MESI ve MOESI en popüler protokollerdir. Moesı'nin MESI protokolünün bir uzantısı olduğu ve bu nedenle daha sofistike ve daha performanslı olduğu yaygın bir görüştür. Bu, yalnızca standart MESI ile karşılaştırıldığında, yani "müdahaleyi paylaşmayan" MESI ile karşılaştırıldığında geçerlidir. MESI Illinois benzeri veya eşdeğeri 5 durumlu protokoller MERSI / MESIF gibi "paylaşım müdahalesi" olan MESI, MOESI protokolünden çok daha performanslıdır.

En gelişmiş sistemler yalnızca R-MESI / MESIF protokolünü veya daha eksiksiz RT-MESI, HRT-ST-MESI ve POWER4 IBM MESI ve MOESI protokollerinin gelişmiş bir birleşimi olan protokollerdir [6].

MSI Protokol: Bu protokol, çok işlemcili sistemde kullanılan temel bir önbellek tutarlılık protokolüdür. Protokol adının harfleri, bir önbelleğin olabileceği olası durumları tanımlar. Modified, Shared, Invalid.

MOSI Protokol: Bu protokol MSI protokolünün bir uzantısıdır. Modified, Owned, Shared, Invalid.

MESI Protokol: En yaygın kullanılan önbellek tutarlılık protokolüdür. Modified, Exclusive, Shared, Invalid.

MOESI Protokol: Bu protokol, diğer protokollerde yaygın olarak kullanılan tüm olası durumları kapsayan tam önbellek tutarlılık protokolüdür. Modified, Owned, Exclusive, Shared, Invalid [7].

2.5. Cache Coherence Sağlamak İçin Olan Gereklilikler

Çok işlemcili bir sistemde, birden fazla işlemcinin X bellek konumunun bir kopyasını önbelleğe aldığını göz önünde bulunduralım. Önbellek tutarlılığını sağlamak için aşağıdaki koşullar gereklidir:

1. Bir işlemci tarafından yapılan bir okumada P aynı işlemcinin yazmasını izleyen bir konuma X P'den X'e, yazma ile P tarafından yapılan okuma talimatları arasında başka bir işlemcinin X'e yazması olmadan, X her zaman tarafından yazılan değeri döndürmelidir.

2. Bir işlemci tarafından yapılan bir okumada P1 başka bir işlemcinin yazmasını izleyen X konumuna P2 -e X, iki erişim arasında meydana gelen herhangi bir işlemci tarafından yapılan başka hiçbir X'e yazma yapılmadan ve okuma ve yazma yeterince ayrılmışken, X her zaman P2 tarafından yazılan değeri döndürmelidir. Bu koşul, tutarlı bellek görünümü kavramını tanımlar. Yazmaların paylaşılan bellek konumuna yayılması, tüm önbelleklerin belleğin tutarlı bir görünümüne sahip olmasını sağlar. İşlemci P1, P2 ile yazdıktan sonra bile eski X değerini okursa, belleğin tutarsız olduğunu söyleyebiliriz [3].

Yukarıdaki koşullar, önbellek tutarlılığı için gereken Write Propagation kriterlerini karşılar. Ancak, Transaction Serialization koşulunu karşılamadıkları için bunlar yeterli değildir. Bunu daha iyi göstermek için aşağıdaki örneği düşünebiliriz:

Çok işlemcili bir sistem, tümü başlangıç değeri 0 olan paylaşılan bir değişkenin önbelleğe alınmış kopyalarını içeren dört işlemciden (P1, P2, P3 ve P4) oluşur. İşlemci P1 değerini değiştirir S (önbelleğe alınmış kopyasında) 10'a, ardından işlemci P2 değerini değiştirir S kendi önbelleğe alınmış kopyasında 20 olarak kaydedilir. Yalnızca write propagationı sağlarsak, P3 ve P4, P1 ve P2 tarafından S'de yapılan değişiklikleri kesinlikle görecektir. Bununla birlikte, P3, P2 tarafından yapılan değişikliği gördükten sonra P1 tarafından yapılan değişikliği görebilir ve bu nedenle bir okumada 10'u S'ye döndürebilir. P4 öte yandan, P1 ve P2 tarafından yapıldıkları

sırayla yapılan değişiklikleri görebilir ve dolayısıyla bir okumada 20'yi S'ye döndürebilir. P4 artık belleğin tutarsız bir görünümüne sahip olur.

Bu nedenle, Transaction Serialization'ı sağlamak ve dolayısıyla Önbellek Tutarlılığını sağlamak için, bu bölümde bahsedilen önceki iki gereklilikle birlikte aşağıdaki koşulun karşılanması gerekir:

- Aynı konuma yazmaların sıralanması gerekir. Başka bir deyişle, konum X, herhangi iki işlemciden bu sırayla iki farklı A ve B değeri aldıysa, işlemciler konum X'i asla B olarak okuyamaz ve ardından A olarak okuyamaz. Konum X, bu sırada A ve B değerleriyle görülmelidir.

Aynı bellek konumuna yapılan tüm yazmalar sıralı bir sırada gerçekleştirilirse, çok işlemcili önbellek tutarlıdır. Aynı verilerin birden çok kopyası aynı anda farklı önbellekte bulunabilir ve işlemcilerin kendi kopyalarını özgürce güncellemelerine izin verilirse, tutarsız bir bellek görünümü ortaya çıkabilir [3].

3. SONUÇ

Bu raporda çok çekirdekli yongayla oluşturulmuş sistemler için önbellek tutarlılık protokolleri araştırıldı. İş yükünü her çekirdeğe dağıtarak uygulama performansını hızlandırmak için çok çekirdekli mimariler ortaya çıktı. Çok çekirdekli mimariler, tek çekirdekli işlemcilerin maksimum performansa ulaşması için yeni bir iş trendi olarak ortaya çıkıyor diyebiliriz. Ayrıca bu mimarilerin, OpenMP (Açık Çoklu İşleme) ve Pthreads (POSIX İş Parçacıkları) kullanılarak iş parçacığı düzeyinde paralellikten yararlanmak için de tasarlandığını görüyoruz. Ancak, çok işlemcili bir sistemde paylaşılan önbelleklerin kullanılması veri tutarsızlığı sorunlarına da neden olabiliyor.

4. KAYNAKÇA

1. http://kursatcakal.azurewebsites.net/Makale/Detay/74
2. http://www.mehmetduran.com/Blog/Makale/Cache-Coherence-Kavrami/194
3. https://en.m.wikipedia.org/wiki/Cache_coherence
4. https://www.educative.io/answers/what-is-cache-coherence
5. https://web.archive.org/web/20140811023120/http://sc.tamu.edu/systems/eos/nehalem.pdf
6. Cache coherency protocols (examples) - Wikipedia
7. https://www.geeksforgeeks.org/cache-coherence-protocols-in-multiprocessor-system/