CHIP MULTIPROCESSOR (CMP) ARCHİTECTURES

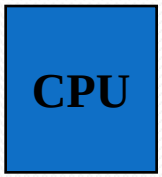
ÇOK İŞLEMCİLİ KIRMIK MİMARİLERİ

İNCİ CABAR

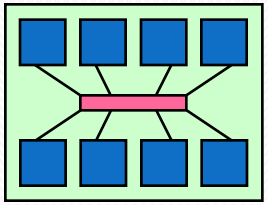
704061016

Giriş

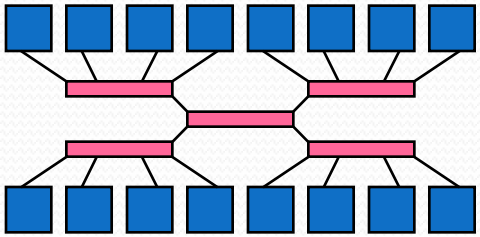
- *Çok işlemcili kırmıklar (CMP)* tek bir kırmık üzerinde birden fazla çekirdek bulundulurlar. Bu mimariler işlemci çekirdekleri arasında hızlı iletişim sağlarlar. Bu hızlı iletişim paralel programların performansını arttırır.



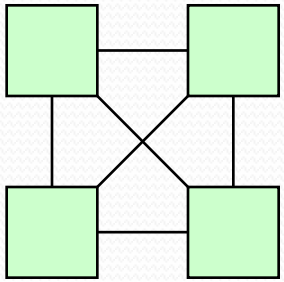
Uniprocessor



Chip Multiprocessor



Multiprocessor

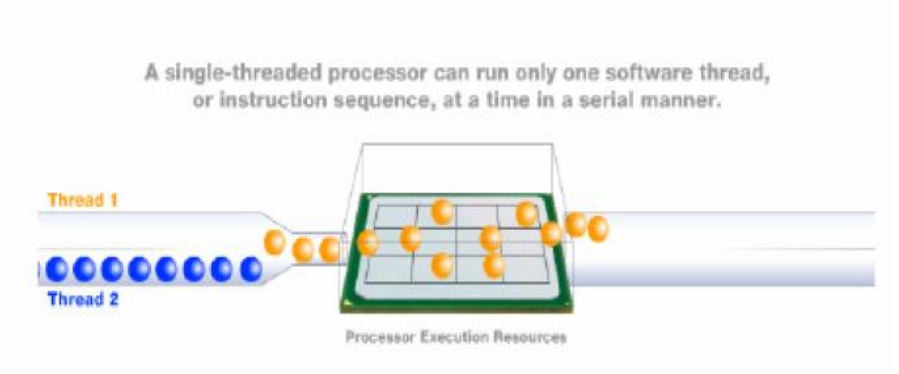


Multiple CMPs

Çok İşlemcili Kırmıklarda Önemli Tasarım Kriterleri

- **Üretilen işe karşı tek-iplikçik performansı (Throughput vs. single-thread performance):**

  Günümüzde programların çoğu tek-iplikçiklidir ve bu CMP'lerde (çok işlemcili kırmıklar) daha fazla bir hız sağlamaz, hatta bellek paylaşımı nedeniyle bir yavaşlama bile görülebilir. Fakat değişik tek-iplikçikli programları aynı anda çalıştırarak üretilen işi (throughput) arttırabiliriz.



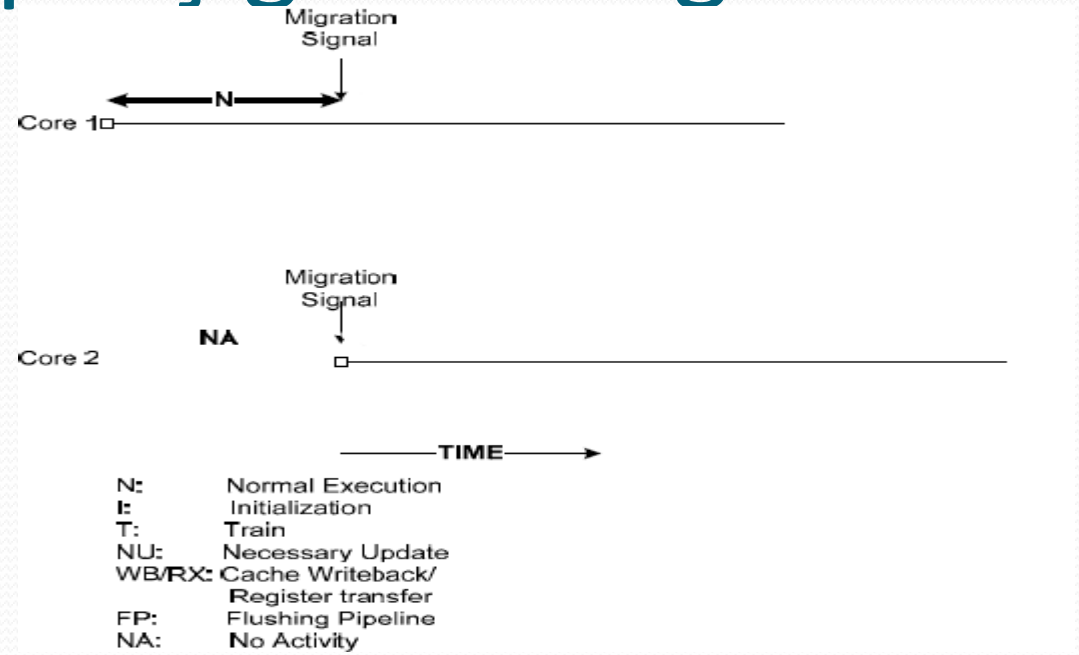
- **Multithreading derecesi ve Multiprocessing derecesi**

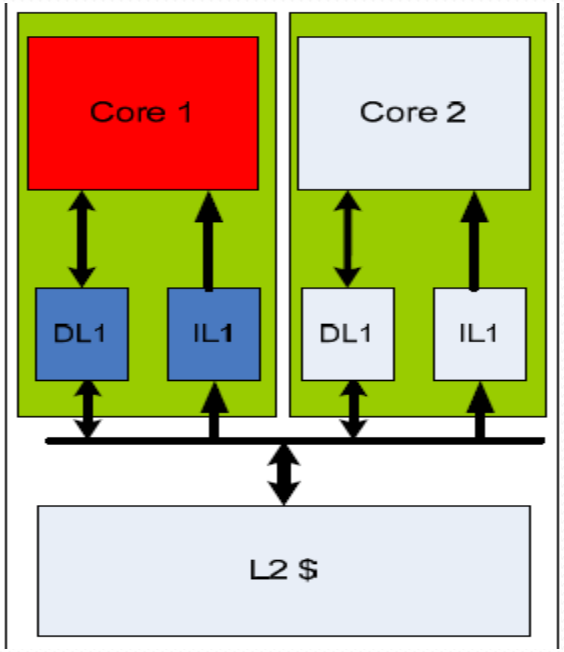
  CMP'den daha iyi yaralanabilmek için uygulamada kaba taneli paralelliği (coarse grained parallelism) kullanmak gerekiyor. Böyle bir paralelliğe İplikçik Düzeyinde Paralellik (Thread Level Parallelism (TLP) örnek verilebilir. Bu paralellikte iki teknik kullanılır: multithreading ve multiprocessing . Multithreading iplikçikler tek bir çekirdekte aynı anda çalışırken, multiprocessing iplikçikleri farklı çekirdeklerde çalışır.

- **Bellek hiyerarşisindeki düzey sayısı**

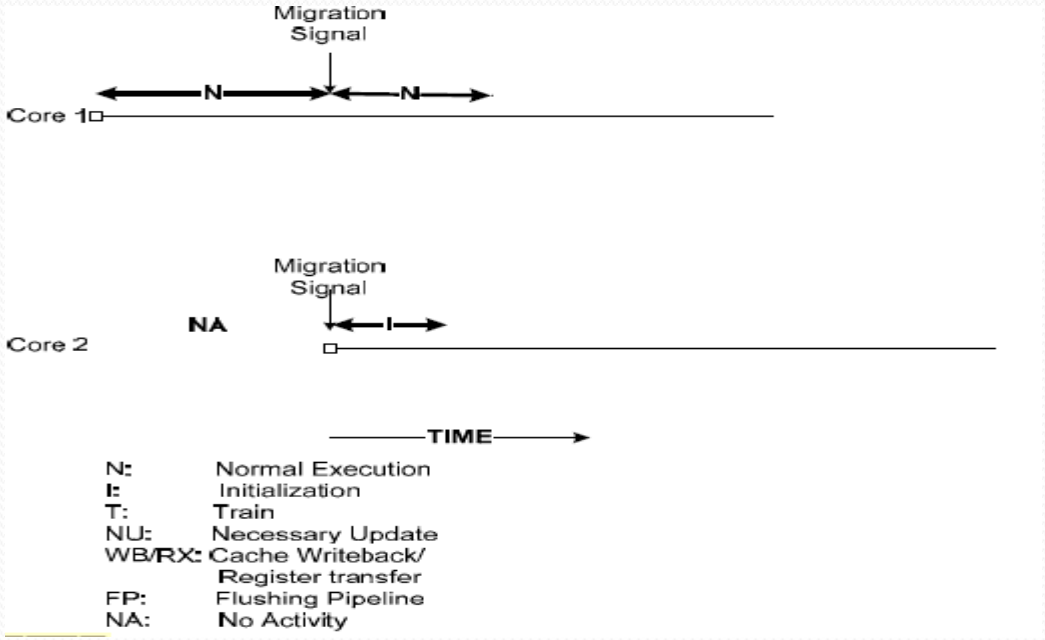
  Günümüzde en revaçta olan tasarım seçeneği küçük özel L1 ön bellekler ve daha büyük paylaşımlı kırmık-içi L2 önbelleklerdir. Fakat bu tek tasarım seçeneği değildir. Örneğin IBM POWER 5'te 3 düzeyli kırmık-dışı L3 önbellekler vardır, bunların kontrolu kırmık üzerinden yapılır.

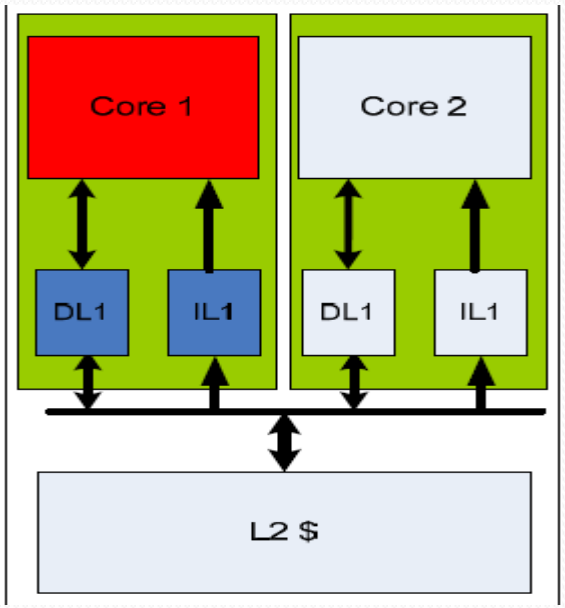
P1 çekirdeğinden P2 çekirdeğine İplikçiğin İzlediğiEvreler



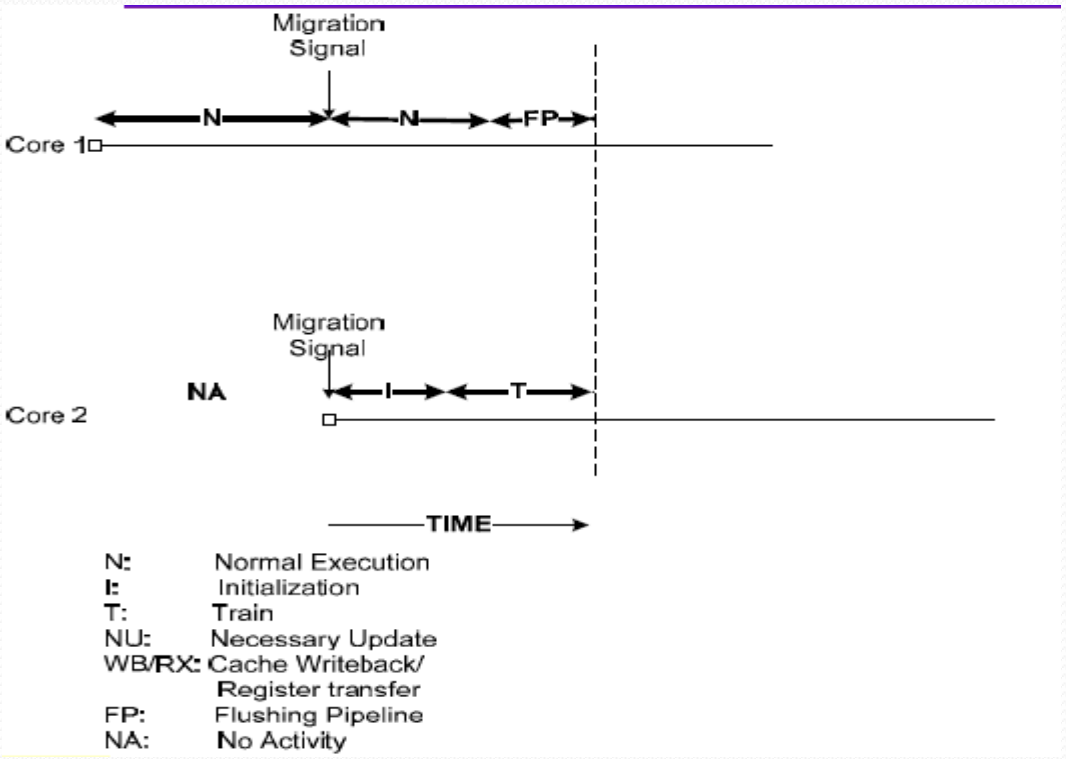


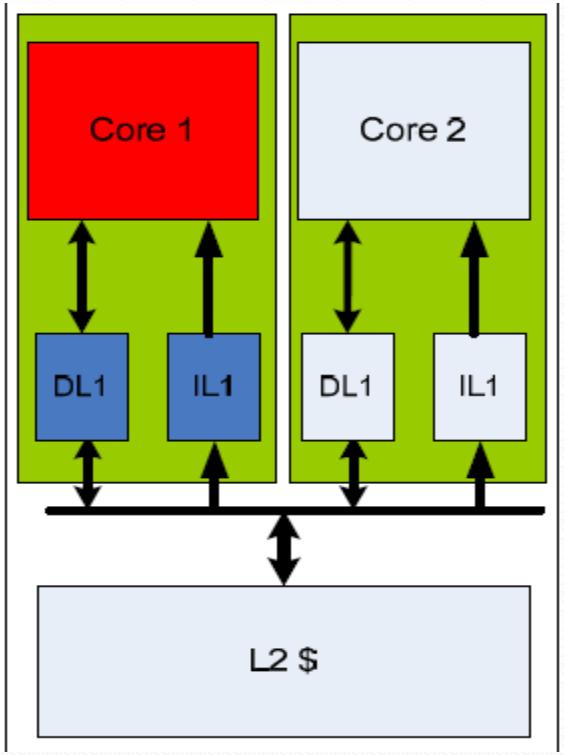
L2 paylaşımlı bellektir, L1 ve L2 write-back ,write-allocate
P1 deki iplikçik normal yürütülürken belli sıcaklık değerine erişince P2 ye yürütmeyi aktarmak için iki çekirdeğe de migration sinyali gönderilir.P2 aktive olur ve kaynaklarını açar.





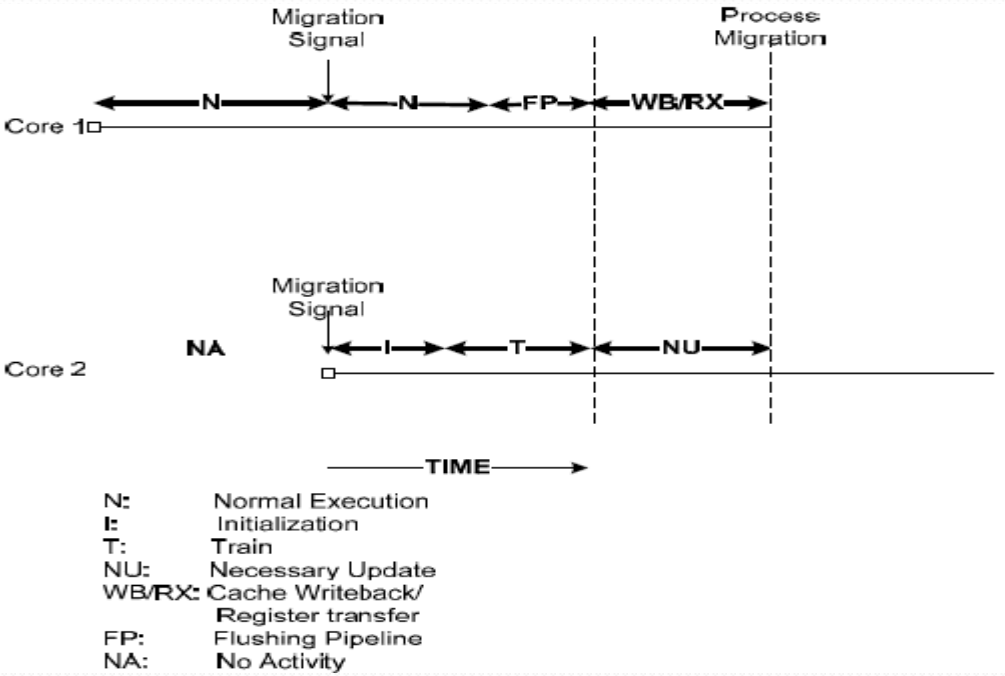
P2 başlangıç durumuna geçer.İnitialization phase .
P1 normal yürütmesine devam eder

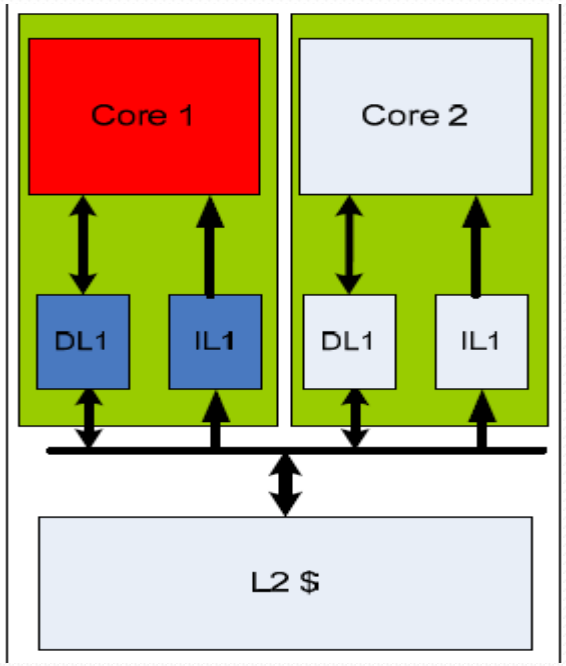




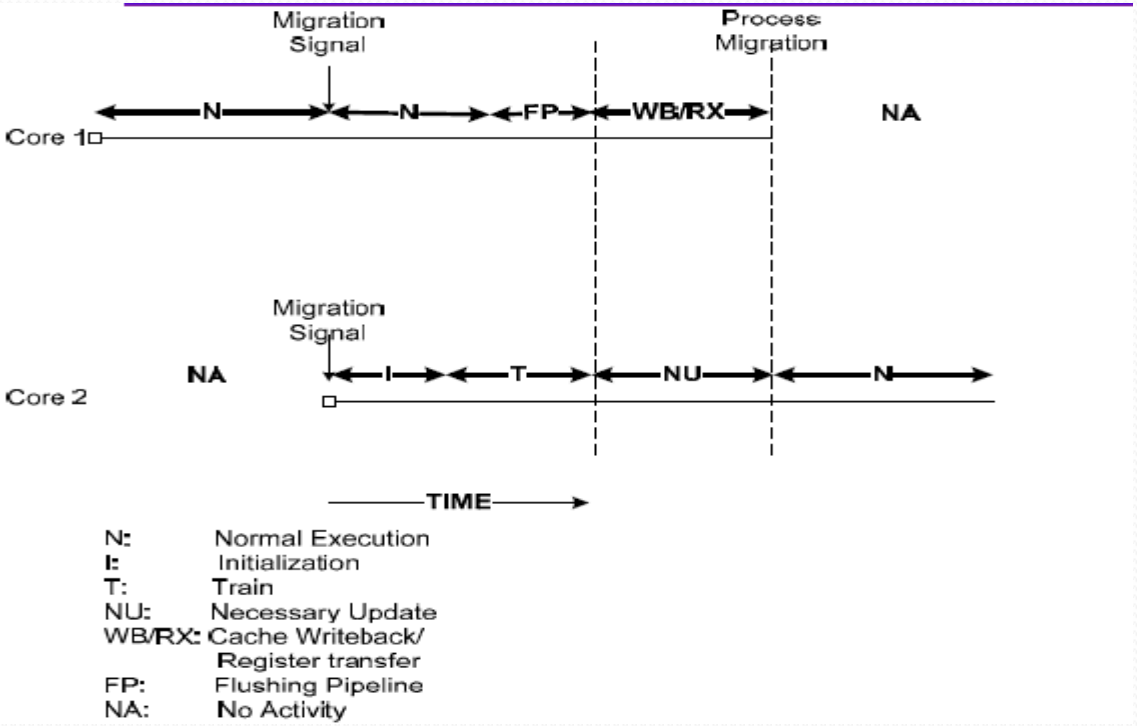
Train phase:P2 ,P1 deki komutların yürütülmesi sırasındaki elde edilen sonuçlar üzerine eğitilir.

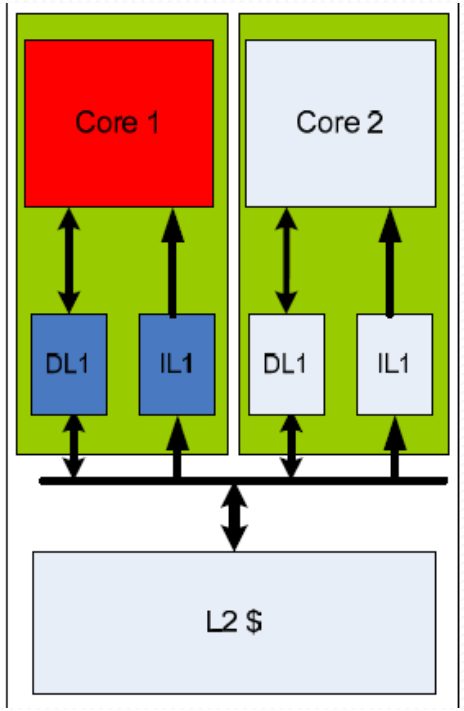
Flushing Pipeline: sona doğru p1 deki işhattı boşaltılır.



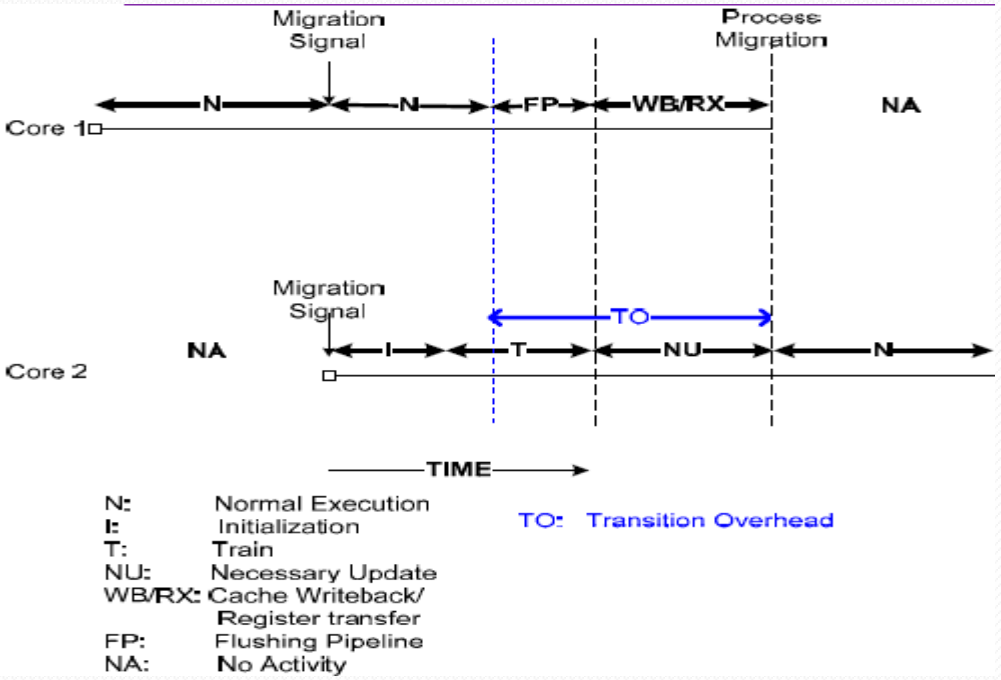


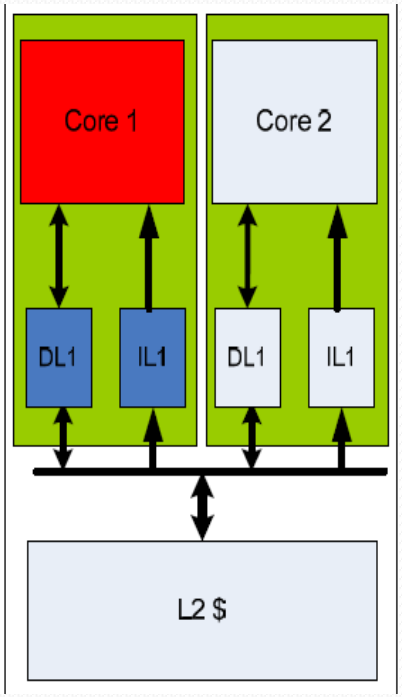
Necessary update:Doğruluk içinsatae essentialları güncellenir.Bu sırada Register P1 den P2 ye transfer edilir ve cache writeback ;kirli önbellek blokları özel önbelleğe aktarılır.



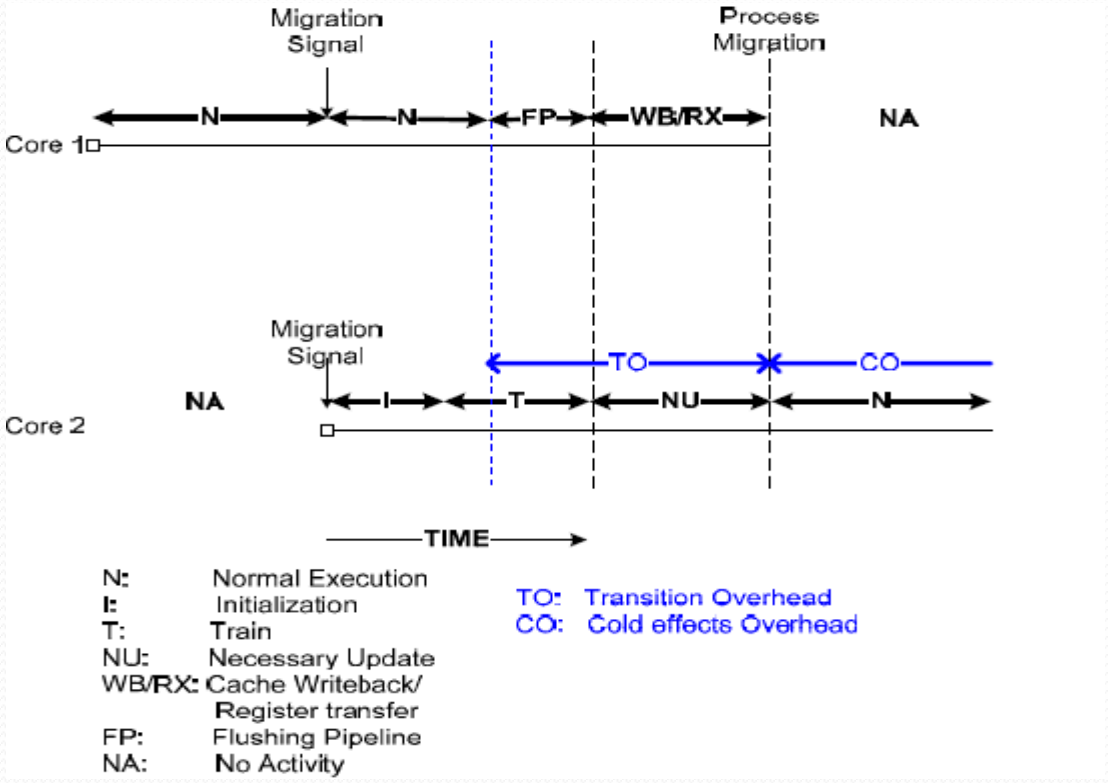


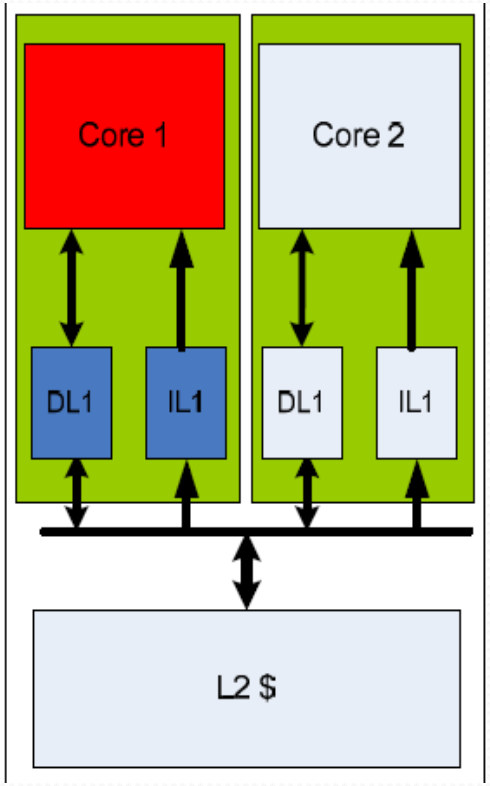
P1 inactive,P2 normal yürütmede



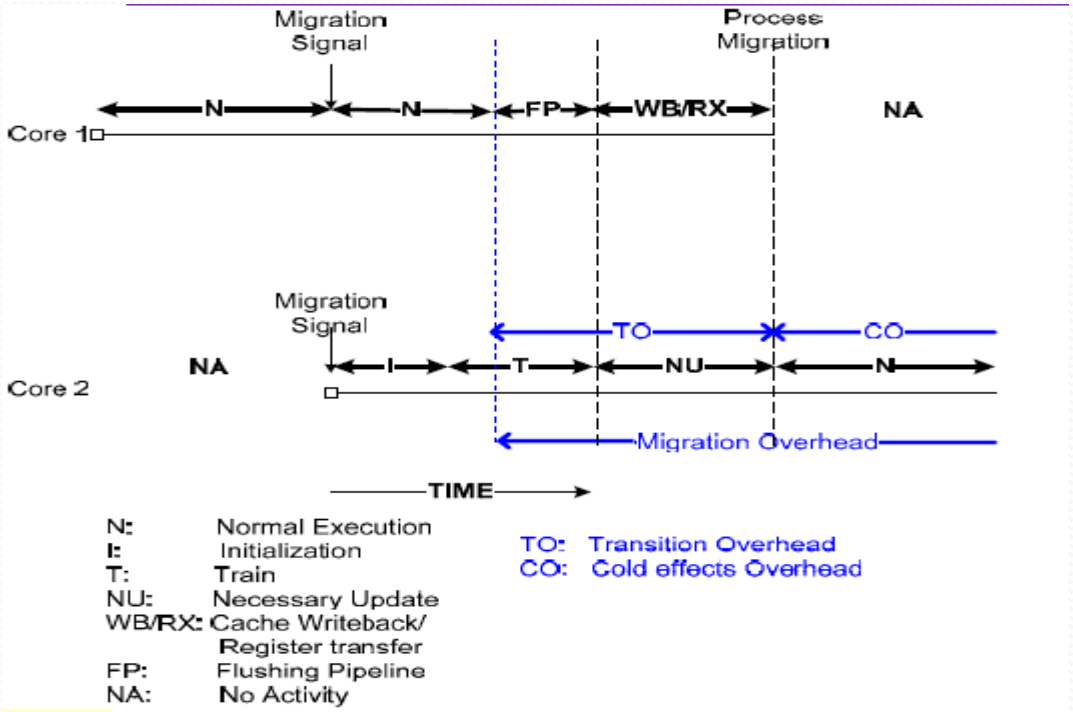


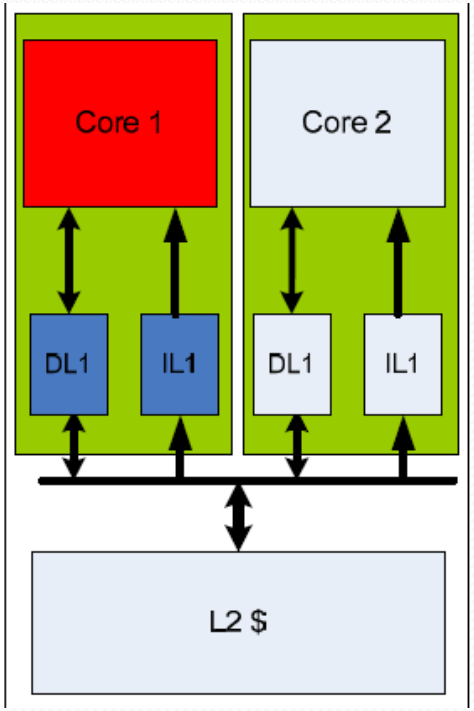
Transition overhead:İşhattı boşaltılması ve necessary update aşamasında olur.Performans düşer





Cold effects overhead: Cache misses ve branch mipredictions

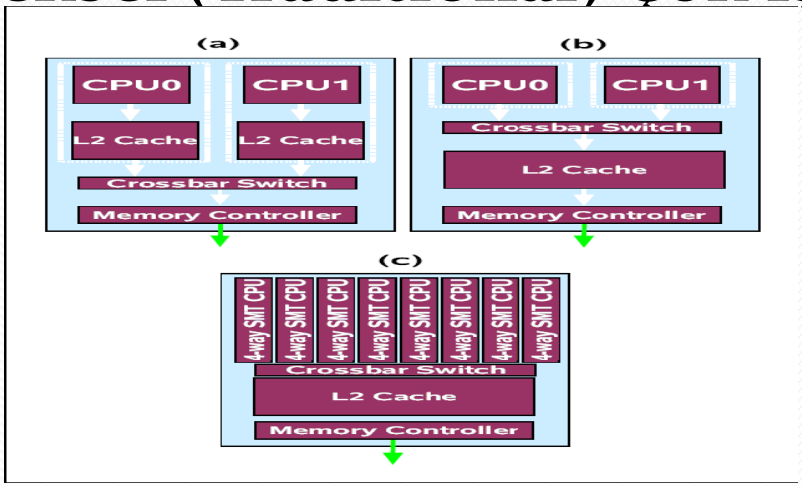




Migration Overhead:TO+CO

Çok İşlemcili Kırmık Çeşitleri

- **Homojen Çok İşlemcili Kırmıklar**
- **Geleneksel (Traditional) Çok İşlemcili Kırmıklar**



a) sadece kırmık-içi bellek denetimcisini paylaşırken

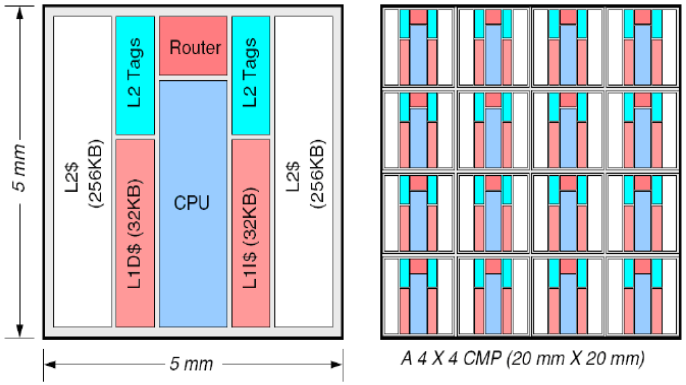
b) deki ikinci nesil CMP'lerde L1 ve L2 arasına çapraz bağlantı eklenmiştir , burada çekirdekler L2 önbelleğini paylaşırlar ve çekirdekler arası kırmık-içi haberleşme mümkündür.

c) de üçüncü nesil CMP'ler görülmektedir ve özellikle iyi bir tek iplikçik performansı sağlamak amacıyla ticari sunucularda kullanır.Yüksek TLP ,düşük ILP değerlerine sahiptir. işlemcisi ULTRASPARC T1,Piranha

Çok İşlemcili Kırmık Çeşitleri

- **Döşenmiş (Tiled) Çok İşlemcili Kırmıklar**

  Burada işlemci çekirdeği , L1 önbelleği, L2 önbelleği ve iletişim yönlendiricisi bir döşe üzerine konulmuştur. Bu döşeler kırmık boyunca yinelenip daha sonra anahtarlanmış ağla bağlanırlar.



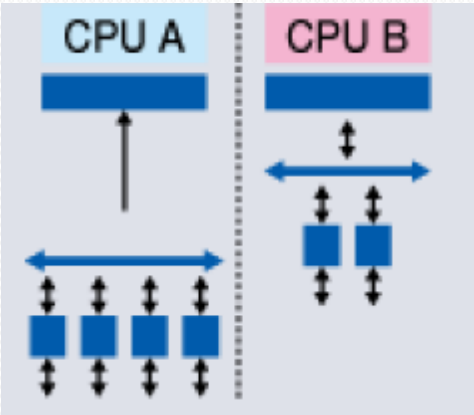
- **Birleştirilmiş Çekirdek (Conjoined Core) Çok İşlemcili Kırmıklar**

  Çekirdeklerdeki kaynaklar paylaşılır. Amaç performansın düşmesini minimumda tutarken, daha az alan kullanmaktır.

Çok İşlemcili Kırmık Çeşitleri

- **Heterojen Çok İşlemcili Kırmıklar**

  Burada CMP'ler farklı performans özellikli ,farklı çekirdeklere sahiptir. Daha az güç tüketimi, ya daha yüksek iş üretimi (throughput) veya her ikisi



Uygulama Düzeyinde Çok İşlemci Programlanması

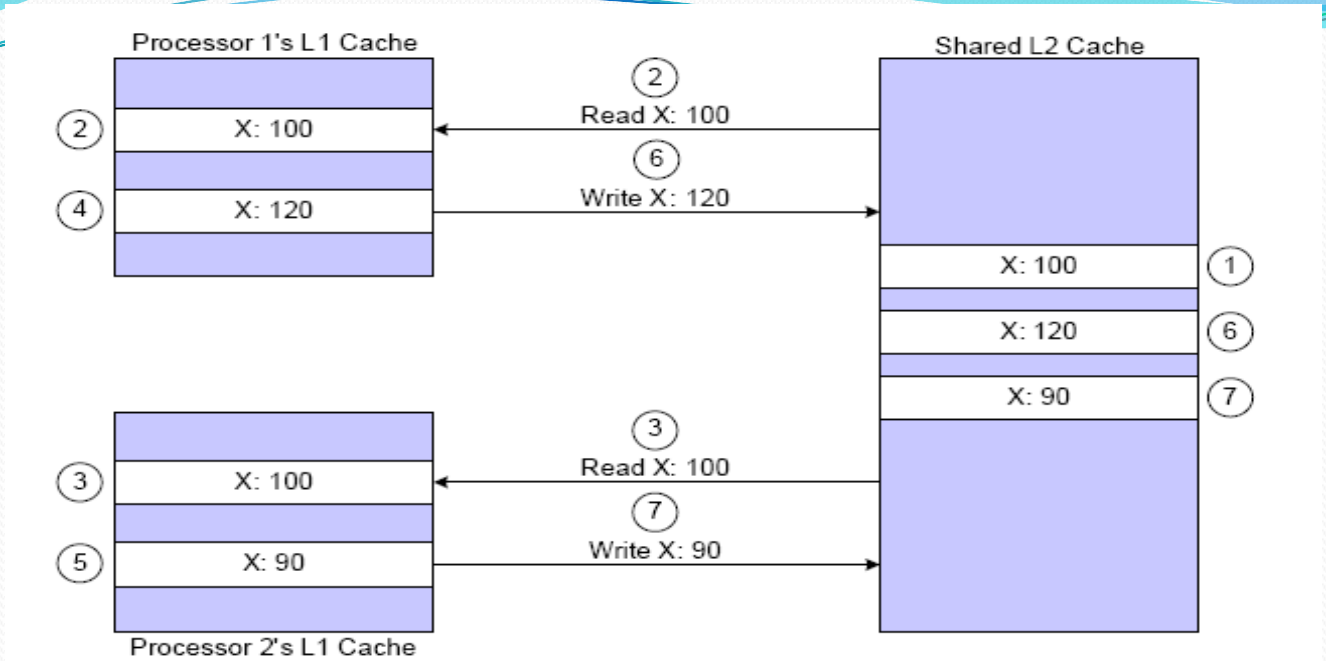
- Birçok iletişim programı senkronizasyon ve iletişimi işletim sistemine dayanır. Uygulamada paylaşılan değişkenler kullanılır. Kodun bu bölümlerine krtitik bölgeler denir ve kilitlerle korunurlar. Programların doğru çalışmalarını sağlamak için programcı kodun büyük bölgelerini korumak için kilitler koyar. Bunlar performansı düşürebilir. Öte yandan sadece gerekli yerlere kilit koymak doğru kodu üretmeyi engeller. Günümüzde, iletişim yazılımı paralel kütüphaneler kullanılarak geliştirilir. Bunlardan en çok kullanılanları OpenMP ve MPI dır. OpenMp ,paylaşımlı bellek kullanarak paralel yazılımı mümkün kılar. MPI ise mesaj geçmeyi (message passing) kullanır.

Sistem Yazılımı Düzeyinde İşlemcinin Çalışması

- İşletim sistemi çoğu bilgisayarda sistem düzeyinde yazılımın asıl parçasıdır, en önemli görevlerinden biri de farklı prosesleri yönetmektir. Bunun sonucunda işlemcide yürütülen kodun büyük bir bölümü işletim sistemi kodudur. Prosesler arası iletişim (interprocess communication) işletim sistemi tarafından sağlanır ve bazı proseslerin erişimin olduğu bellek ayırmayı kullanır. Ayrıca, bu paylaşımlı belleğe erişecek proseslerin sırasının kontrolu, söz konusudur. Bu senkronizasyon görevi semaforlar kullanılarak gerçekleşebilir. Paylaşımlı bellekte, çok sistemli işlemciler için yazılım senkronizasyonunu destekleyen ölçeklenebilir senkronizasyon algoritmaları vardır. Burada kullanılan bazı komutlar; test and set, fetch and store, fetch and add ve compare and swap komutlarıdır. Fakat senkronizasyon akıllıca yapılamazsa bu operasyonlar arabağlaşım çekişmesine (interconnect contention) neden olabilir. Bu problem de işlemci sayısı arttıkça kötüleşebilir.

Çok işlemcili Kırmıklarda Önbellek Uyumluluğu ve Tutarlılığı

- Bellek sisteminin bir kısmını paylaşmak CMP de önbellek uyumluluğu problemine neden olur. Bu problem, kendini iki veya daha çok işlemci önbelleklerindeki aynı veri değerini kullanıp, güncelledikleri̇nde ortaya çıkar ve programın yürütülmesi sırasında hatalar oluşur.
- CMP'lerde önbellek uyumluluğu problemi daha çok farklı işlemcilerin L1 önbellekleri ile, paylaşımlı L2 önbelleği arasındadır. Ayrıca, eğer birden fazla CMP kırmığı aynı belleğe bağlıysa, bellek ile farklı kırmıklardaki L2 önbellekler arasında da önbellek uyumluluğu incelenmelidir



Şekilde önce L2 100 değerindeki X bloğunun bir kopyasının olduğu tek önbellektir.1 no'lu işlemci bu değeri okuyup kendi önbelleğinde saklar. Daha sonra 2 no'lu işlemci bu değeri okuyup kendi önbelleğinde saklar. 1 no'lu işlemci önbellek hattına 120 yazar. Bu güncellemeyi 2 no'lu işlemci ve L2 önbelleği göremez. 2 no'lu işlemci önbellek hattına 90 yazar. Bir süre sonra blok 1 no'lu işlemcinin önbelleği ile değiştirilir. Daha sonra L2 önbelleğine yazılıp değer 120 olarak güncellenir. Ardından 2 no'lu işlem X'in bir kopyasını geri yazar. L2 önbelleğindeki değer 90 olmuştur ve 1 no'lu işlemci tarafından yapılan değişiklik kaybolur.

Önbellek Uyumluluğu Protokolleri

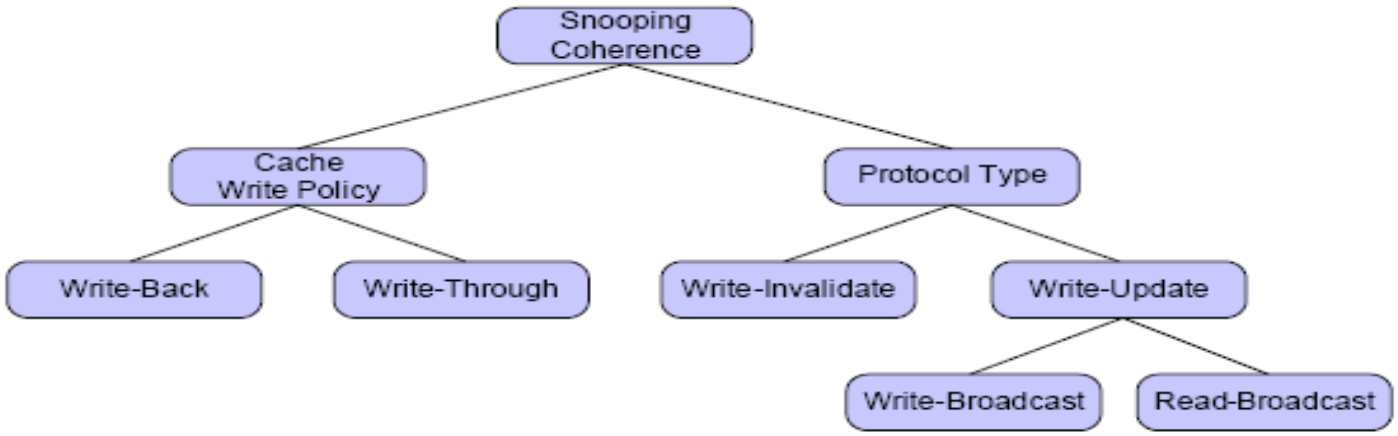
- Önbellekte uyumsuzluk olmaması için paylaşılan değerler önbellek uyumluluk protokolleri ile kontrol edilir. Önbellek uyumluluğu protokolleri yazılım veya donamım temelli olabilir. Donanımla yapılanlar çok hızlıdır ve program saydamlığı sağlar. Bunlar trafik gözetleme ve dizin protokolleridir.Bunlara geçmeden önce yazılımla yapılanlardan bahsedeceğim.

- Verimlilik düşüncesiyle her bir işlemciye bir tane özel önbellek bağlanması istenir. Bu yöntem sadece paylaşılmayan ve yalnız okunabilen verilerin önbelleğe yazılmasıdır. Düzenleyici verileri önbelleklenebilir ve önbelleklenemez olarak ayırır. Önbelleklenebilir veriler önbelleklere yazılırken önbelleklenemez veriler bellekte kalır. Bu da önbelleklere yazılan veri tiplerini kısıtlar. Ayrıca fazladan yazılım gerektirir.

- Bir diğer yöntem, yazılabilir verilerin en azından bir önbellekte bulunmasıdır. Bu yöntem derleyici de merkezselleştirimiş küresel çizelge kullanır ve bellek bloklarının durumu(read only-RO veya read/write –RW) buraya yazılır.RO'laın tüm önbelleklerde kopyaları vardır, RW bloğunun ise tek kopyası bulunur .Böylece veriler önbellekte RW bloğuyla güncellenirse diğer önbellekler etkilenmez .

Trafik Gözetleme (Snooping) Önbellek Protokolü

- Trafik gözetleme protokolü işlemcilerin diğer işlemcilerin bellek hareketlerini izleyebildikleri paylaşımlı ortama dayanır. Bu protokol hızlıdır .Öte yandan paylaşılan veri yolundaki trafik fazladır.



- Şekilde bu protokol için verilmesi gereken bazı seçimler vardır. İlk seçim önbelleğin yazma ilkesiyle ilgilidir, write-back ya da write-through olabilir. Daha sonra önbellek protokol tipi seçilir. Olası seçimler write-invalidate ve write-updatetir. Write-invalidate protokolde yazmalar yerel(local) olarak yürütülür. Ve tüm diğer blok kopyaları geçersiz(invalidate) kılınır. Write-update protokolü bloğun yazıldığı önbelleklerde saklanan veriyi günceller(update). İki protokolde aynı zamanda sadece tek önbelleğe bloğa yazma izni verilir. Bir veri yazılırken ya da daha sonra bir önbellek veriyolundaki okumayı görünce , bu güncellemenin direkt olarak yürütülmesi de sözkonusu olabilir. Buna sırasıyla write-broadcast ve read-broadcast denir.

- Trafik gözetleme protokollerinin tercih edilme nedeni, dizin protokollerine göre uygulamasının kolay olmasıdır. Fakat bloğu paylaşımlı L2 den getirmeden önce bloğun diğer işlemci önbelleklerinde olup olmadığını kontrol edilmelidir. Bu daha çok tüm önbelleklerin isteğini yayınlamak(broadcast) ile olur ve iki tane sorun yaratır. İlki bu yayın fazla arabağlantı band genişliği kullanır. Ayrıca bir istek, gerekli önbellek bloğu paylaşılmasa da yayınlanır(broadcast) çünkü önbellek, diğer önbellekleri kontrol ederken kontrol ettiği önbelleğin paylaşımlı olup olmadığını bilmez. Dolayısıyla bellek isteklerinin gecikmesi artar. Trafik gözetleme protokolünü geliştirmek için iki yol vardır;

  •Protokolü, yayın sayısını azaltarak geliştirebiliriz. Cantin bu yaklaşımı kaba-taneli uyumluluk izleme (Coarse-grain coherence tracking) tekniğinden alır. Burada her çekirdek, adres alanının (space) toplam uyumluluk bilgisini içerir. Eğer bu adres alanında başka işlemcinin blokları yoksa, yayın bilgisine gerek yoktur.

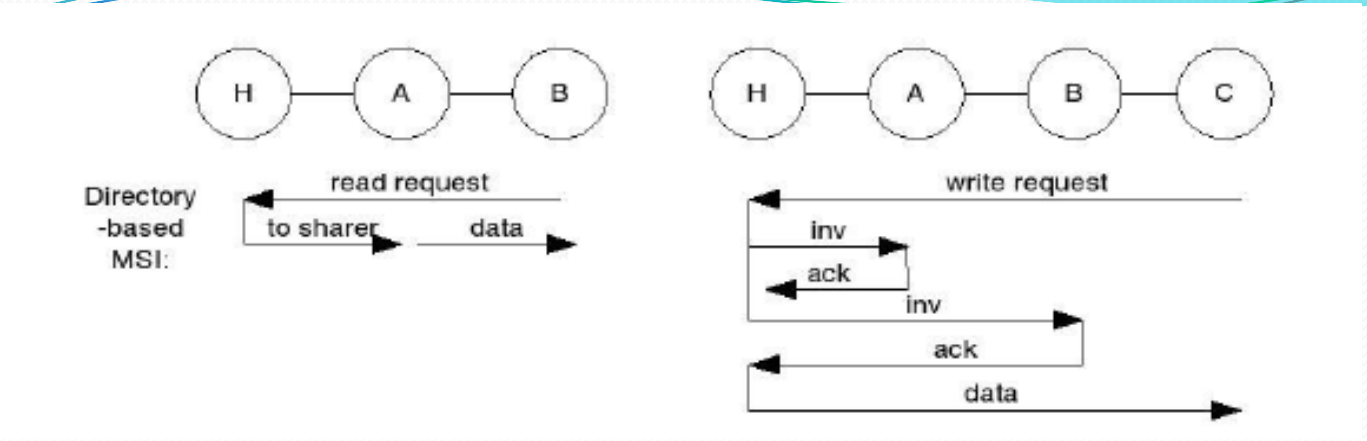
  •Protokol daha güçlü arabağlantılar kullanılacak şekilde güncellenir. Örneğin protokoller halka(ring) arabağlantı ile çalışabilirler.

Dizin(Directory) Protokolleri

- Dizin protokolleri blokların nerede tutulduğu ve durumları hakkında bilgi depolar. Dizinler merkezi ya da dağıtık olabilir. Merkezi olanda tüm blokları tutan bir dizin vardır. Dağıtık ise pek çok dizin kullanır, bu da ölçeklenilebilirliği geliştirir. Dizin trafik gözetlemeye göre daha az mesaj üretir fakat gecikme daha fazladır . Ayrıca dizinin uygulaması daha zordur.

- Bu protokolde aynı blok için aynı zamanda iki işlemci yarışıyorsa, yarış mesajlar dizine ulaşınca sona erer çünkü verieln önbellek bloğu için bir tane dizin vardır. Fakat, kaybedene Negative Acknowledge ment (NACK) mesajı gönderilir. Bazen , bazı protokol eylemleri için Acknowledgement (ACK) mesajı gerekir.

- Bu protokol hangi çekirdeklerin hangi önbellek bloklarında , önbellek kopyaları olduğunu tutar. Bu durumda bellek erişim sıraları dizin tarafından takip edilir.



a b

- Şekil a)da B işlemcisi, A da olan bir blok için okuma isteğinde bulunur. B'nin isteği H (home node) dizinine gönderilir. H, A'ya veriyi B'ye sunması yönünde komut verir ve transaction sona erer. Şekil b ise C işlemcisinin ,A ve B işlemcilerinin önbelleklerinde olan bir önbellek bloğuna yazmak istediği zamanki protokol eylemlerini göstermektedir. C, H dizinine istekte bulunur. H, A ve B ye geçersizlik mesajları gönderir ; A ve B den ACK mesajlarını alınca veriyi C işlemcisine gönderir.

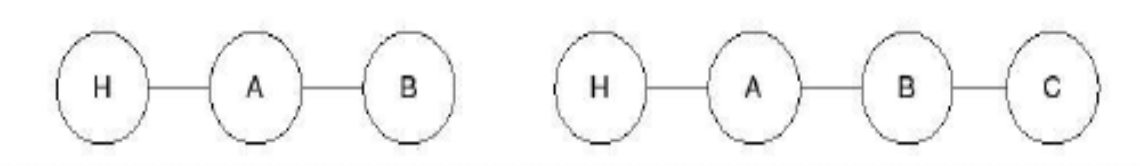
- Şimdi iki dizin protokolünü görelim:

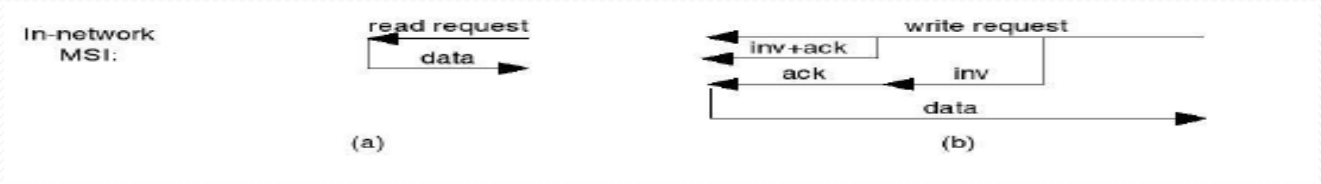
• In-Network Önbellek Uyumluluğu; burada dizin protokolleri istekleri , arabağlantı ağını geçerken optimize edilir.

• Stenstr¨om ' un dizin protokolü. Burada protokol eylemleri hızlandırılır.

In-Network Önbellek Uyumluluğu

- Bu teknikte dizin protokolünün performansı, protokol eylemlerinin gecikmesi azaltılarak, arttırılır.
- Şekilde bu gecikme azalması görülmektedir. Burada protokol ve dizinler ağ routerları(yol atayıcıları) içine gömülüdür.





- Şekil a'da okuma isteği için arabağlant ağı geçilip veri direkt olarak temin edilir, gecikme azalır. Şekil b'de İn-network protokolü protokol eylemlerini, yazma isteğini tüm paylaşanlardan geçirerek, optimize eder. A ve B geçersiz kılma mesajlarını C'nin isteğini görünce işleme koyarlar. Sonra H; tüm ACK mesajlarını ve isteği alınca, veriyi sunar.

The Stenstr¨om Dizin Protokolü

- Diğer protokollerden şu şekilde ayrılır. Eğer A önbelleği X önbellek bloğuna sahipse, A önbelleği hangi diğer önbelleklerde X in kopyası olduğunu bilir. Ayrıca X in kopyası olan tüm önbelleklerde, A önbelleğinin X'e sahip olduğunu bilir. Yani X bloğunu okumak isteyince A'ya direkt olarak başvururlar.

- Burada , özel L1 leri ve paylaşımlı L2 önbelleği olan bir hiyerarşiyi ele alacağım.

- 6 tane olası protokol durumu vardır fakat Global Read modda bunlardan üçü kullanılır.

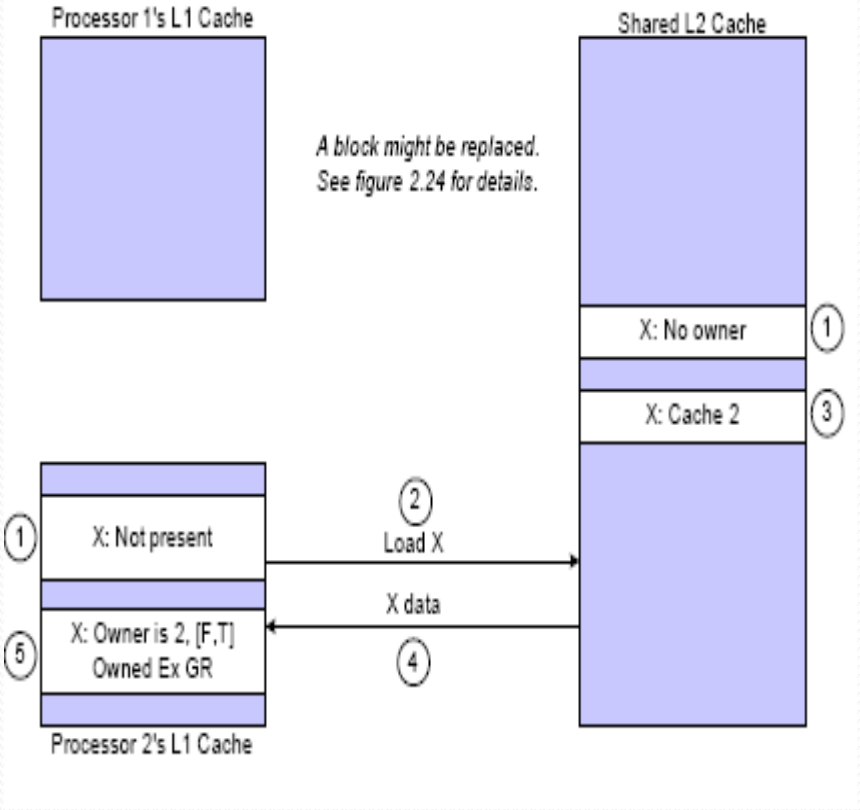
İnvalid: Önbellek bloğu geçerli değildir. Sahibin bilgisi mevcutsa,okumalar bloğa sahip olana yönlendirilir. Y azmalarda, yazma yürütülmeden önce sahiplik elde edilir.

Owned Exclusively(Yalnız sahiplik) Global Read :Bönbellek bloğa sahiptir ve bloğun tek kopyası vardır.Okuma ve yazma ertelemesiz yürütülür.

• Owned NonExclusively Global Read : Önbellek bloğa sahiptir fakat en az bir önbellek daha invalid durumda bloğun kopyasını bulundurur.Önbellek hattının diğer kopyaları invalid olduğundan ,okuma ve yazmalar ertelemesiz yürütülür.

- Diğer durumlar şunlardır: UnOwned,
  Owned Exclusively Distributed Write ve
  Owned NonExclusively Distributed Write.
  Bu durumlar sadece distributed write modda geçerlidir.
- **Okuma Vuru Durumu:** Önbellekte, önbellek bloğunun geçerli(valid) biti 1 ise istek vurudur. Bu ancak, Owned Exclusively Global Read ve Owned NonExclusively Global Read durumlarında gerçekleşir. Diğer bir değişle en güncel veridir ve okuma gecikmesiz yürütülür.
- Diğer paylaşanları haberdar etmeyiz, çünkü veriyi değiştirmiyoruz .Ayrıca Global Read modda olduğumuz için önbellekte bloğun tek kopyası vardır.Yani diğer paylaşanlar veriye gerek duyduklarında bu önbelleğe erişirler.

- **Okuma Iska Durumu:** Üç durum sözkonusudur. İlki, önbellek bloğu L1 de mevcut değilse; ikincisi, blok başka bir L1 önbellekteyse, üçüncüsü blok mevcut fakat geçersizse(invalid).Tüm bu durumlar protokol tarafından ele alınır.



X bloğu kopyasının paylaşımlı L2 de olduğu durum

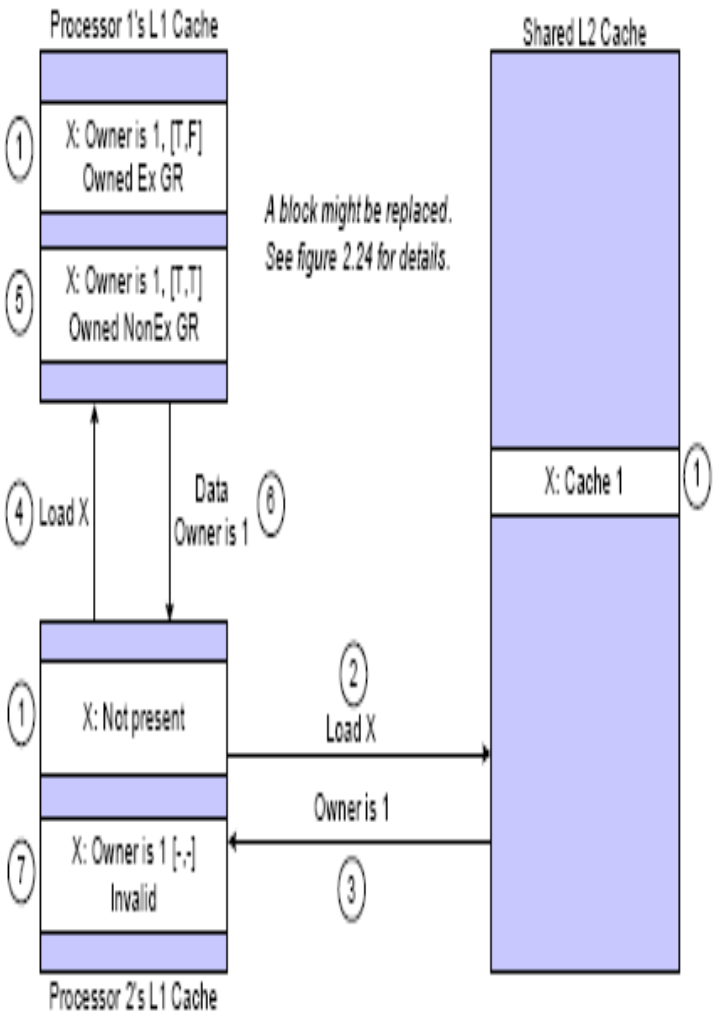
1.X in tek geçerli kopyası L2 de

2. İşlemci 2 nin L1 i,L2 'ye load isteğinde bulunur.

3.L2 önbelleği , İşlemci 2 nin L1 ini, X bloğunun sahibi olarak setler.

4.X bloğu L1 e gönderilir.

5.L1 veriyi depolar ,güncel bayrakları sıfırlar ve durumu Owned Exclusively Global Read e setler.



X in farklı bir L1 önbelleğinde olduğu durum:

1.Başlangıçta,geçerli kopya işlemci 1 in önbelleğinde ve L2, işlemci 1 i X bloğunun sahibi olarak kaydetmiş.X bloğu işlemci 2 nin önbelleğinde yoktur.

2.İşlemci 2 ,X bloğu için L2 den load isteğinde bulunur.

3.L2, işlemci 2ye işlemci 1'in X e sahip olduğunu söyler.İşlemci 2, isteğini işlemci 1 e yönlendirir.

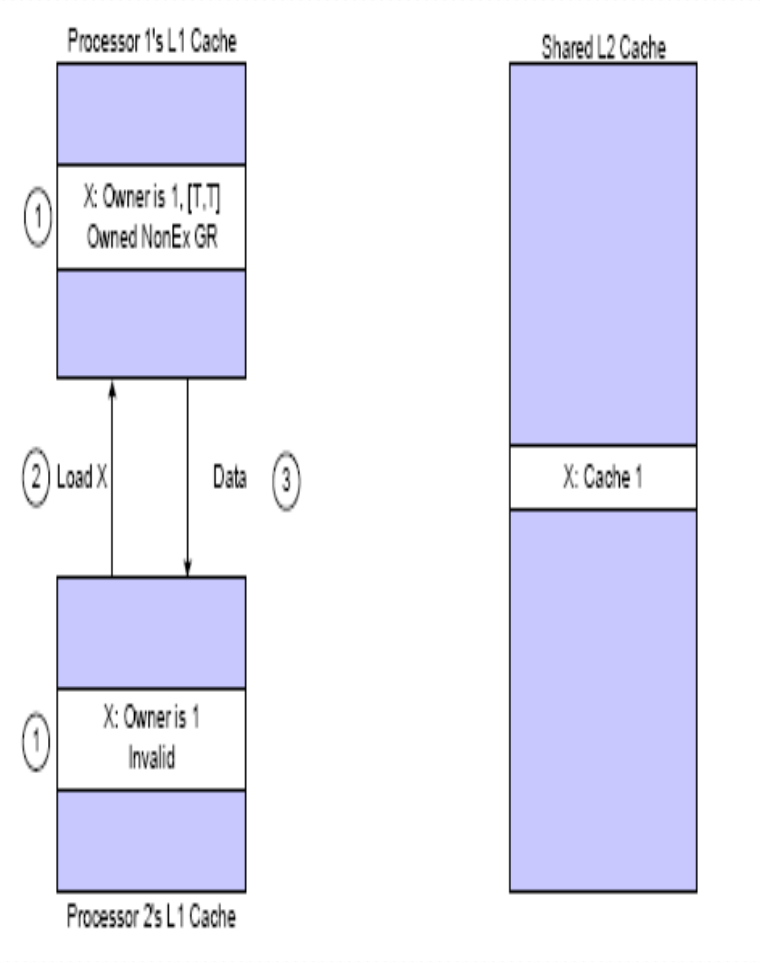
4.İşlemci 2,işlemci 1 e load isteğini gönderir.

5.İşlemci X bloğunun durumunu ,Owned Exclusively Global Read ten, OwnedNonExclusively Global Read e çevirir.Ayrıca işlemci 2 nin mevcut(present) bayrağını setler.Eğer İşlemci 1 in önbelleğindeki X bloğunun durumu Owned NonExclusively Global Read idiyse, X bloğu bu durumda kalır ve sadece mevcut bayrağı setlenir.

Bu durum, 2 den fazla işlemcili sistemlerde eğer diğer işlemcilerden biri daha önceden X bloğunu okuduysa gerçekleşir.

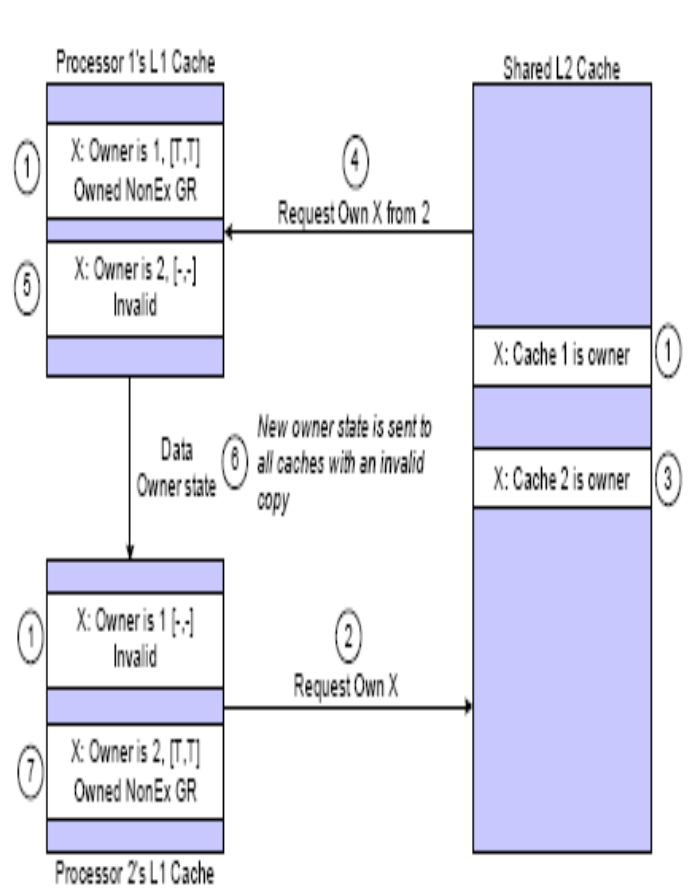
6.İşlemci 1 veriyi ve sahiplik bilgilendirmesini(owner identification) işlemci 2 ye gönderir.Owner identification işlemci 2 nin bir daha hangi işlemciye mesajın gönderildiğini hatırlamasına gerek kalmayacak şekilde eklenir.Diğer bir deyişle buffer avoided olur.

7.İşlemci 2, X i önbelleğinde store eder ve alınan veriyi kullanır.X bloğu durumu invalid yapılır ,böylece sonraki blok okumaları sahip(owner) önbelleğe yönlendirilir yani protokol veriyi göç ettirmez(migrate)



Önbellekte,X bloğunun invalid kopyası varsa:
1.Önce, X bloğu hem işlemci 1 in hem de 2'nin L1 önbelleklerinde mevcuttur.İşlemci 1,X bloğuna sahiptir. Bir kopyadan fazla olduğu için durum Owned NonExclusively Global Readtedir. Tüm sahip olunmayan kopyalar Global Read modta invalid olana kadar İşlemci 2 nin X kopyası invalidtir.
2.İşlemci 2 ,işlemci 1 in sahip(ownwer) olduğunu bilir ve L2 ye erişmeden direkt load isteğini işlemci 1 e bildirir.Bu durumda diğer dizin protokolleri L2 ye ilk olarak erişmek isteyeceklerdir.
3.İşlemci 1 veriyi işlemci 2ye gönderir.L1 lerde protokol durum değişikliği yoktur.

- **Yazma Vuru Durumu:** Üç olası protokol eylemi vardır. İlkinde, blok Owned Exclusively Global Read durumundadır.Bu durumda yazma başka bir blok kopyası yokmuş gibi ertelemesiz yürütülür. İkincisi blok, Owned NonExclusively Global Read durumunda olabilir. Burada, bloğun diğer tüm kopyalar ı geçersiz olduğundan ertelemesiz yürütülür. Son olarak önbellek hattının geçersiz olma durumundaysa önbellek bloğa yazmadan önce sahip olmak ister.



Önbellek hattının invalid olduğu olasılıkta,önbellek bloğa yazmadan önce sahip olmak ister.Bu durumun protokol eylemleri aşağıdadır:

1.İlk olarak, işlemci 2 nin önbelleğinde X bloğunun geçersiz kopyası vardır.Geçerli kopya Owned NonExclusively Global Read durumunda işlemci 1 in önbelleğinde mevcut.Ayrıca,L2 işlemci 1 in X bloğunun sahibi olduğunu biliyor.

2.İşlemci 2 X bloğunun sahipliğini L2 den ister.Serileşme noktası olduğundan bu istek dizine gönderilir.

3.L2, işlemci 2yi X bloğunun sahibi olarak setler.

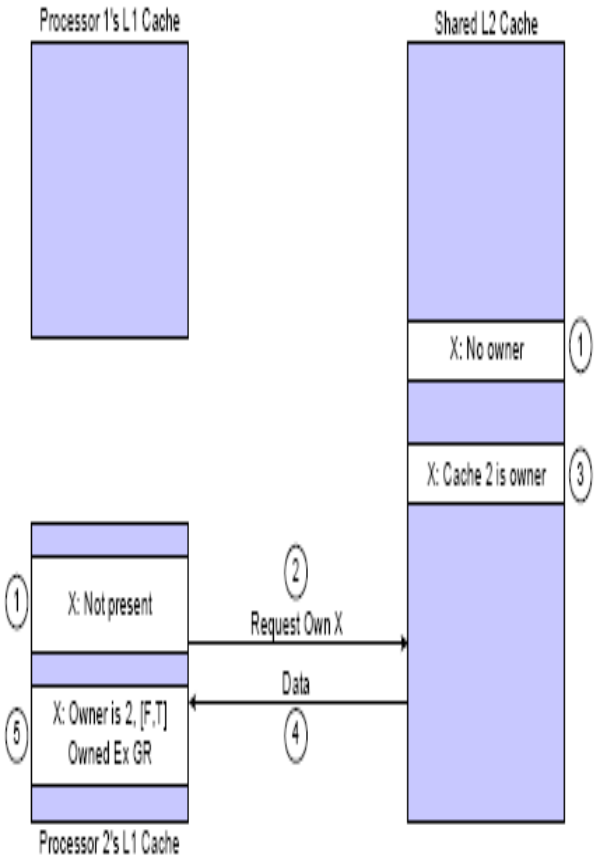
4.L2 işlemci 1 e bu durumu bildirir.

5.İşlemci 1 X in durumunu invalid e setler ve işlemci 2 yi yeni sahip(owner) yapar.

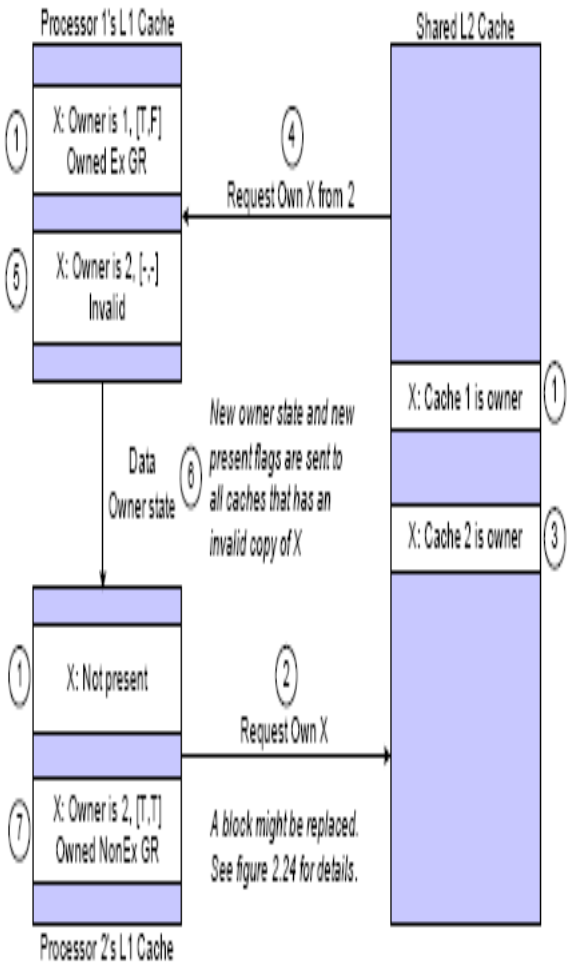
6.İşlemci 1 X bloğu için olan veri ve present bayraklarını işlemci 2 ye gönderir.Ardından, X olan tüm önbellekleri işlemci 2 nin yeni sahip olduğu konusunda bilgilendirir.
Present bayrakları işlemci 1 e hangi işlemcilerin bilgilendirileceğini söyler.

7.İşlemci 2 X in veriyi ve X in present bayraklarını önbelleğinde store eder. Durum Owned NonExclusively Global Read e setlenir. Yazma işlemi yürütülüp, modified biti 1 e setlenir.

- **Yazma Iska Durumu:** Sadece sahibi bloğa yazabilir. Bu yüzden yazmadan önce önbellek bloğun sahipliğini elde etmelidir. İki durum sözkonusu. İlki blok L1'de mevcut, ikicisinde ise L2 den getirilmeli.



X bloğunun L1 önbelleklerde olmadığı durum:
1.İlk başta, x bloğu işlemci 2 nin L1 önbelleğinde ve L2 de mevcut değildir.
2.İşlemci 2, X bloğunun sahipliğini L2 den ister.
3.L2 gerekirse X bloğuna bellekten erişir ve işlemci 2 yi ownwer olarak setler.
4.L2 veriyi işlemci 2 ye gönderir.
5.İşlemci 2, X bloğunu önbelleğinde store eder ve durumu Owned Exclusive Global Read e setler.
Present bayrakları ve işlemci 2 nin bayrağıyla 1 e setlenir.Diğer bayraklarda 0 a setlenir.Yazma yürütülüp modified bit 1 e setlenir.



Blok istenen işlemci nin önbelleğinde değil başka bir L1 de mevcut.

1.Başta, X işlemci 2 nin önbelleğinde yoktur ve işlemci 1 X in sahibidir.

X,Owned Exclusive Global Read durumundadır, fakat protokol eylemleri başka paylaşanlar(sharers) varmış gibi aynıdır.Tek fark, 6 noktasında sahibin bilgisi tüm paylaşanlara yollanır.

2.İşlemci 2,L2 den X in sahipliğini ister.

3.L2, X in sahibini işlemci 2 olarak değiştirir.

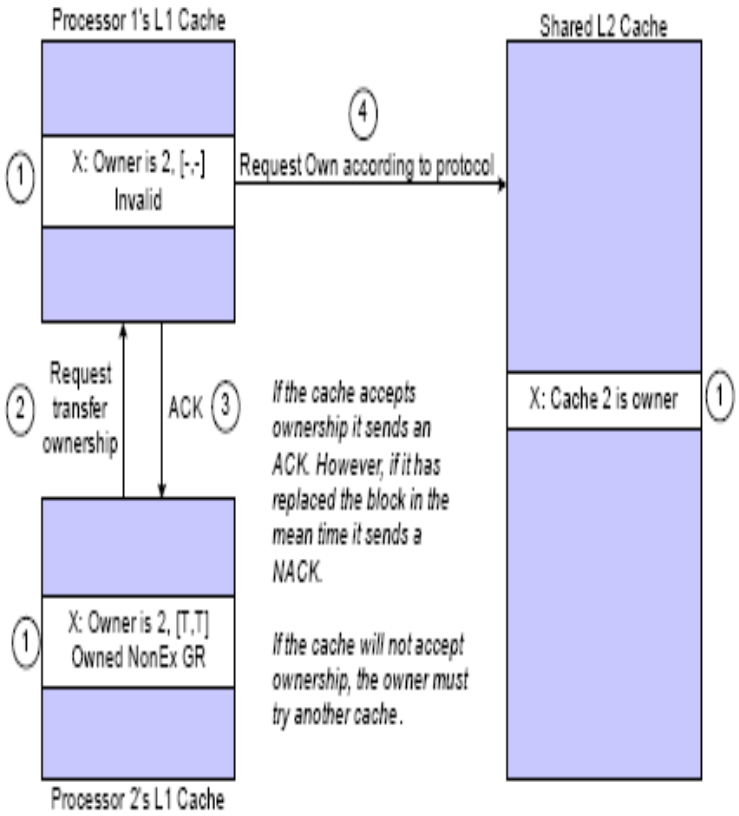
4.L2, işlemci 1 e bu durumu bildirir.

5.İşlemci 1,işlemci 2 yi X in sahibi olarak setler ve durumunu invalid yapar.

6.İşlemci 1 veriyi ve X in present bayraklarını işlemci 2 ye yollar.Sistemde X in kopyalarına sahip başka işlemciler varsa,işlemci 1 yeni sahibin bilgisini bunlara gönderir.

7.İşlemci 2 ,işlemci 1 den aldığı veri ve present bayrakları ile X için önbellek girişini yaratır veya günceller. X in yeni durumu Owned NonExclusively Global Read olur.Yazma yürütülür ,modified bit 1 lenir.

- **Blok Değiştirme:** Üç tane olası protokol eylemi vardır. İlki, eğer blok yalnız sahipleniliyorsa ,diğer deyişle tek kopya ise. Bu durumda L2 ,önbelleği n daha fazla bloğa sahip olmadığı konusunda uyarmalı. Ayrıca blok değiştirilmişse geri yazılmalı. Diğer bir durum bloğun yalnız sahiplenilmediği durumdur. Üçüncü ise bloğun geçersiz olduğu durumdur.



Blok,yalnız sahiplenilmiyorsa. Burada bloğun sahipliği başka bir L1 e trasfer edilir.Yeni sahip present bayraklarından rastgele seçilir.

1.İşlemci 2,X in sahibidir ve işlemci 1 de X in invalid kopyası vardır.

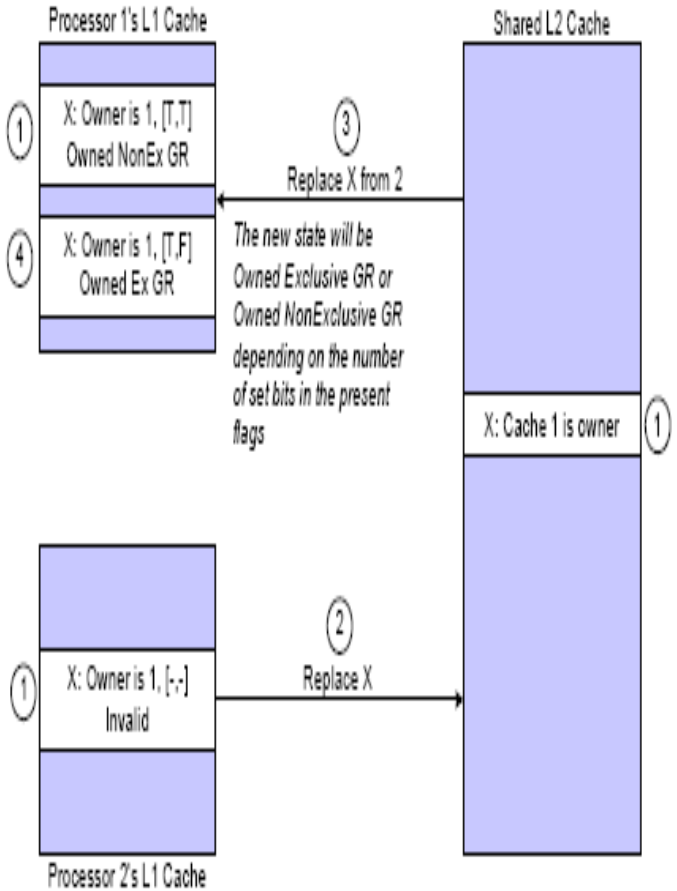
2.İşlemci 2 ,X bloğunu değiştirmek ister ve işlemci 1 in X in yeni sahibi olmasını ister.

3.(a)Eğer işlemci 1 in önbelleğinde hala X bloğu varsa, Acknowledgement (ACK) mesajı ile cevap verir. İşlemci 1 X in sahipliği için alışılmış protokol adımları izlediği için ;işlemci 2 X bloğunu değiştirememiştir.

(b)Eğer işlemci 1in önbelleğinde X yoksa Negative Acknowledgement (NACK) mesaj ile cevap verir.

Daha sonra işlemci 2,işlemci 1 in present bayrağını 0 a setler ve sahipliği farklı bir önbelleğe vermeye çalışır.Eğer başka paylaşan yoksa,işlemci 2 durumu Owned Exclusively Global Read e setler ve yalnız sahiplik durumu için protokol eylemlerini izler.

4.Şekilde işlemci 1 X in sahipliğini kabul eder. Sonra,normal protokolü kullanarak X bloğunun sahipliğini ister.İşlemci 2 veri ve present bayraklarını işlemci 1 e verince, X i değiştirebilir.



Bloğun invalid olduğu durum için protokol eylemleri:

1.İşlemci 1 , X e sahiptir ve işlemci 2nin geçersiz kopyası vardır. İşlemci 2 X i değiştirmek ister.

2.İşlemci 2 L2 yi ,X i değiştireceği konusunda bilgilendirir.Writeback e gerek yoktur çünkü invalid kopya değiştirilemez.Bu istek gönderildiğinde , işlemci 2 X i değiştirmekte özgürdür.

3.L2 işlemci 1 i, işlemci 2 de artık X in kopyası olmadığına dair bilgilendirir.

4.İşlemci 1 present bayraklarında işlemci 2 nin bitini 0 yapar.Eğer X in tek kopyası işlemci 1deyse durum Owned Exclusively

Global Read olarak değişir.

Çok İşlemcili Kırmıklarda Senkronizasyon Mekanizmaları

- Senkronizasyonu lock-based ve lock-free olarak sınıflandırabiliriz. Lock-based algoritması, kilitleri kullanarak işlemcileri senkronize eden bir algoritmadır. Kilitler, bariyerler, ve mutual exclusion, paylaşılan bir değişkene erişip veriyi değiştirmek için kilitlediklerinden lock-based olarak nitelendirilir. Başka bir proses değişkeni kullanmak istediğinde meşgul beklemede ya da dönme modunda (spinning mode) kalır ve kilidin bırakılıp bırakılmadığına bakar ve kilit özgür kalınca diğer proseslerle kilide sahip olmak için yarışır.

- Bu tekniğin uygulaması paylaşılan bellekte sorun çıkmasın diye mutexler ve semaforlar gibi donanım kullanılarak yapılır. Bu metotda ölümcül kilitlenme (deadlock), starvation, öncelik evirmesi(priority inversion) oluşabilir. Deadlocklar, farklı prosesler birbirlerinin semaforları bırakmasını bekledikleri zaman oluşurlar. Starvation kaynakların bir sürecin tamamlanmasındaki yetersizlikleridir. Öncelik evirmesi de yüksek öncelikli iplikçiğin düşük öncelikliyi beklediği zaman oluşur. Meşgul bekleme, proses boş durumda saat çevrimi harcadığı için, faydasızdır.

- Lock-free algoritmalar bu problemi çözmeyi amaçlarlar. Lock-free algoritmalar birden çok iplikçiğin aynı anda okuma ve yazma yapmasına izin veren algoritmalardır. Wait-free algoritmalar ise bir iplikçiğin bir işlemi diğer iplikçiklerin operasyonlarına bakmadan belli sayıda adımda bitirmesini sağlayan algoritmalardır.

Çok İşlemcili Kırmıklarda İletişim

- **Paylaşımlı Bellek Kullanarak İletişim**

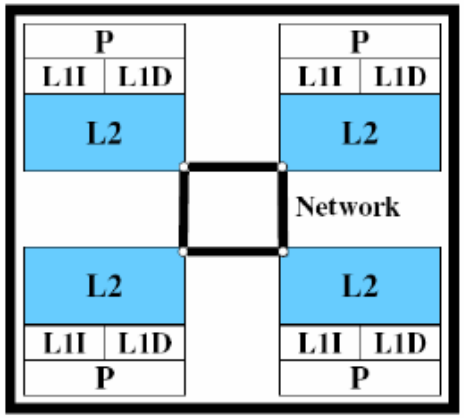
  Paylaşımlı bellek iletişimi CMP'lerde çok verimli olmaktadır. Bunun sebebi hem belleğin hem de işlemci çekirdeklerinin aynı kırmıkta olmasındandır.

  Özel (Private) L1 ve Özel L2 Önbellekler

  Her çekirdekte özel L1 ve L2 önbellekleri vardır, kolay bir yapıdır. Fakat bu önbellekler küçük ve paylaşımlı önbelleklere göre düşük vuru oranlarına sahiptir.

  Yavaş, chip-wide resource utilisation

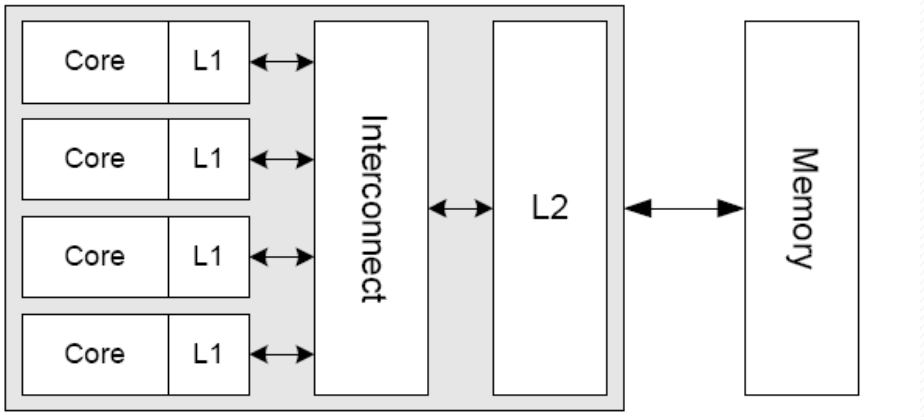
  İmeceli Önbellekleme(Cooperative Caching)



Çok İşlemcili Kırmıklarda İletişim

- **Özel L1 Önbellekler ve Paylaşımlı L2 Önbellek**

  Çekirdekler arasında hızlı bir iletişim ve hızlı bir vuru zamanı sağlar. Paylaşımlı L2 önbelleğin mahsuru uzun önbellek erişim zamanıdır. Fakat iyi kırmık genişliği önbellek kullanımı(chip-wide resource utilisation) ve daha az kırmık-dışı bellek erişimleri bu yaklaşımı çekici kılmaktadır.



Çok İşlemcili Kırmıklarda İletişim

- **Paylaşımlı Lı Önbellek**

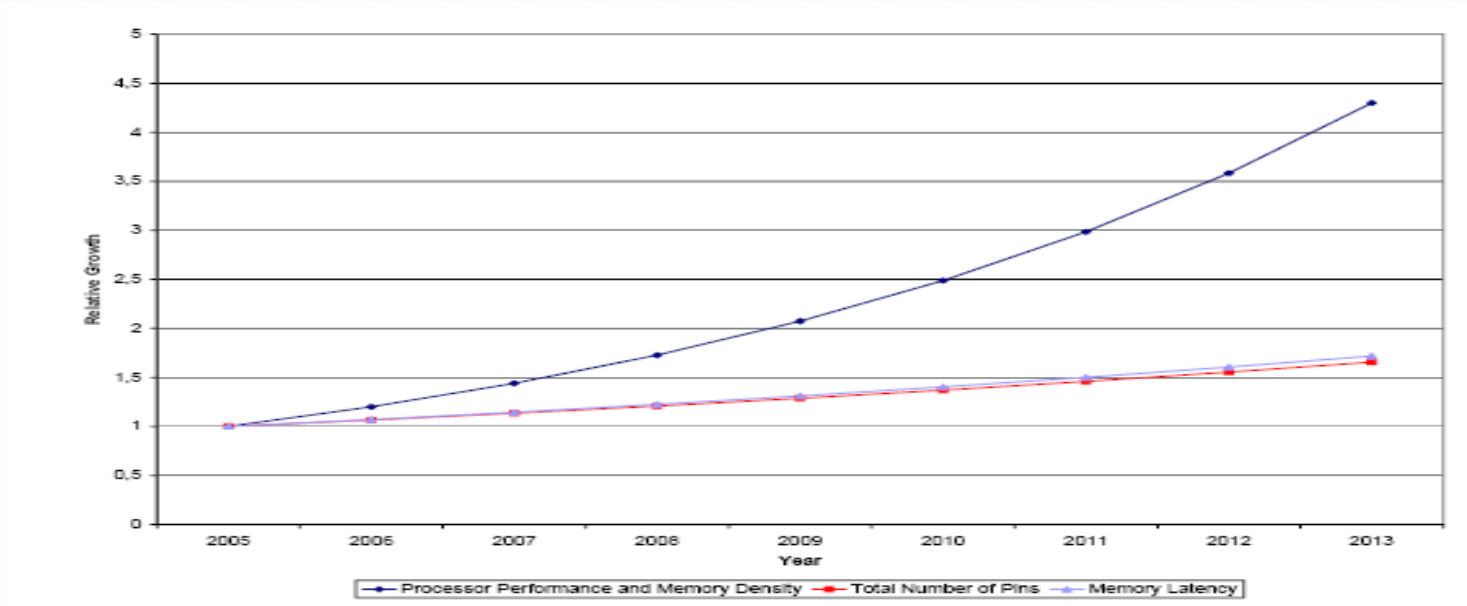
  Çok hızlı bir kırmık-içi iletişim söz konudur. Fakat yüksek seviyede tek-iplikçik performansı, yüksek frekansa bağlıdır

Mesaj Geçme Yolu ile İletişim

Burada prosesler arası iletişim açık mesajlarla olur. CMP'lerde fazla kullanılmaz.

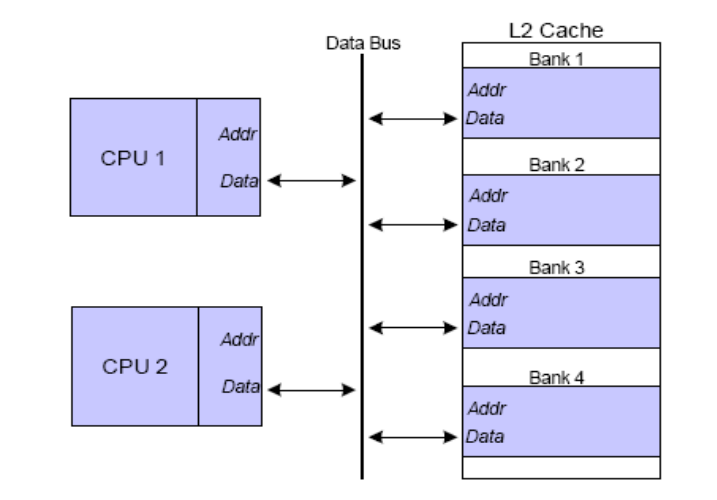
Çok İşlemcili Kırmıklarda Arabağlantı Mimarileri

- Bunlardan bazıları küçük alan ve düşük band genişliği gerektirirken bazıları büyük alan ve yüksek performans gerektirirler.



Paylaşımlı Ortam Ağları (Shared Medium Networks)

- Çekişmeli veriyolu(contention bus)
- Andaçlı veriyolu (token bus)
- Hakemli veriyolu(arbitrated bus)



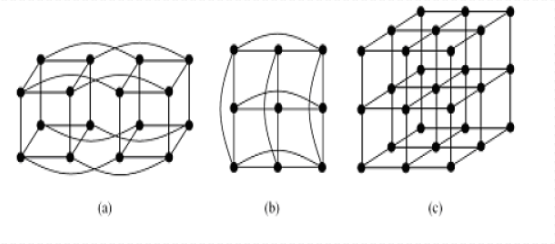
- Bu teknikler arsındaki fark, paylaşılan ortama erişimin kontroludur. Çekişmeli veriyolu çözümünde, tüm birimler herhangi bir anda iletilebilir. İletimin yanı sıra birimler veriyolundaki sonuçları da kontrol eder. Bu yolla, veriyolundaki değerler bekledikleri değerler mi diye bakarlar. Eğer değilse, farklı bir birimin de aynı anda iletildiği sonucuna varırlar, iletimi durdurup daha sonra yeniden denerler. Buna Ethernet'i örnek verebiliriz. Bu yöntemin dezavantajı, bir birimin ne sıklıkta veriyoluna erişim alacağının garanti olmamasıdır. Bu yüzden, çok birimli bir sistemde, her birim istediğinde veri iletirse, diğer birimlerin iletişimlerini yok etmek olasılığı bir hayli yüksektir.

- Bu probleme andaçlı veriyolunu çözüm olarak sunabiliriz. Burada, andaç round-robin biçiminde birimler arasında geçer. Bir birime, andacı varsa iletim izni verilir. Bu teknik ölçeklenebilirliği geliştirir ama performansı etkiler. Sorun iletmek isteyen birimin iletmek istemeyen birimleri beklemek zorunda olmasıdır.

- Bu sorunun çözümü ise hakemli veriyolundadır. Bu teknikte, birimler iletilmeye hazır verilerinin olduğunu belirtirler. Sonuç olarak bir birim sadece veriyolunu kullanması gereken birimleri bekler. Hakemli veriyolu diğer iki teknik arasında uzlaşma olarak görülebilir.

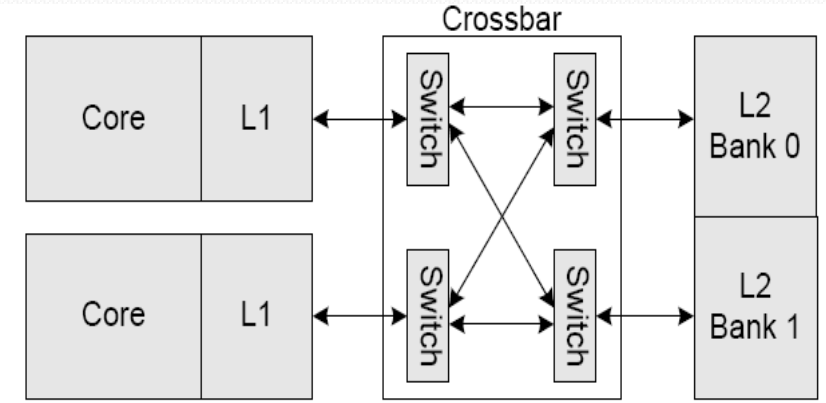
Doğrudan Ağlar (Direct Networks)

- Paylaşımlı ortamın tüm birimler için iyi ölçeklenememesi sonucu farklı alternatifler denenmelidir. Bunlardan biri doğrudan ağlardır. Burada CMP'mizi düğümlere böleriz. Bu düğümler genelde L1 önbellekli ve farklı L2 öbekli (bank) CPU çekirdekleri olur. Her düğümün iletişimi ele alan bir yol atayıcısı vardır ve her yol atayıcısı komşu düğümlerdeki yol atayıcılarına(router) bağlıdır. Diğer bir deyişle, iki komşu düğümün birbirlerine doğrudan bağlantısı vardır. Düğümlerin bağlanma şekillerine ağ topolojisi denir

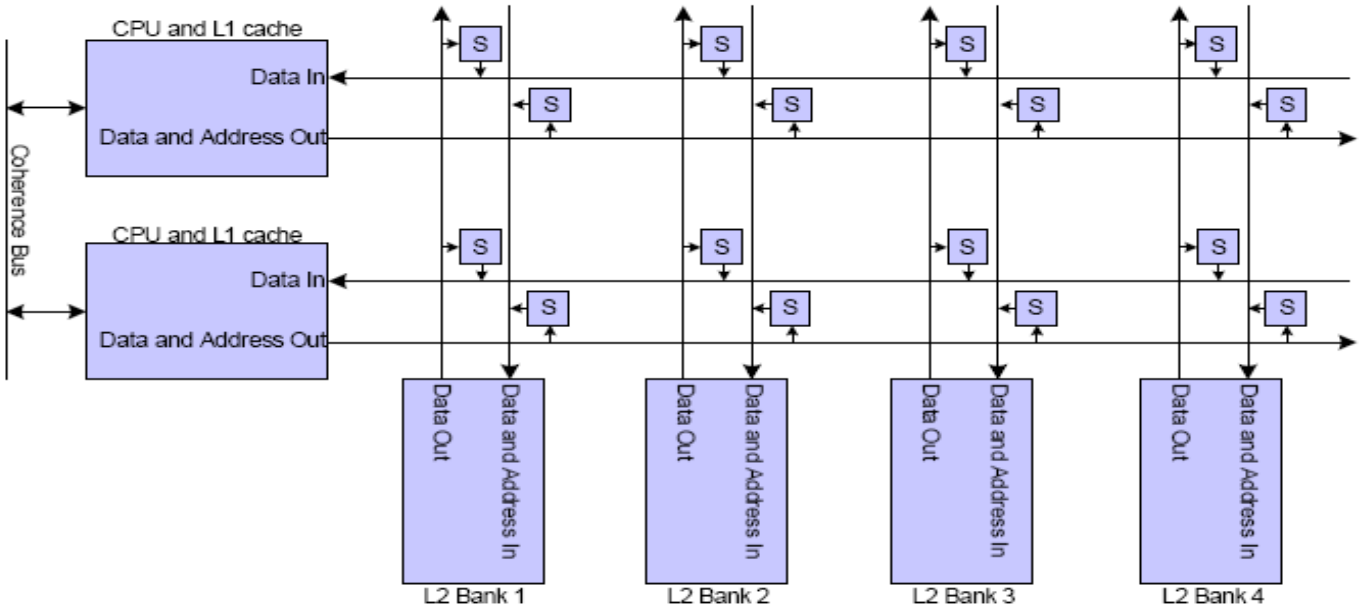


Dolaylı Ağlar (Indirect Networks)

- Bu ağlar doğrudan ağlara benzerler. Temel farkı bir dolaylı ağın düğümleri arasında anahtarlar olmasıdır. Bir başka deyişle, düğüme enjekte edilen bir mesaj varış düğümüne ulaşmadan belli sayıda anahtardan geçer. Bir anahtarın en azından bir giriş ve bir çıkış portu vardır ve düğüm anahtarlaması giriş portunu çıkış portuna bağlamakla yapılır.
- **Çapraz Bağlantı Ağları (Crossbar Networks)**



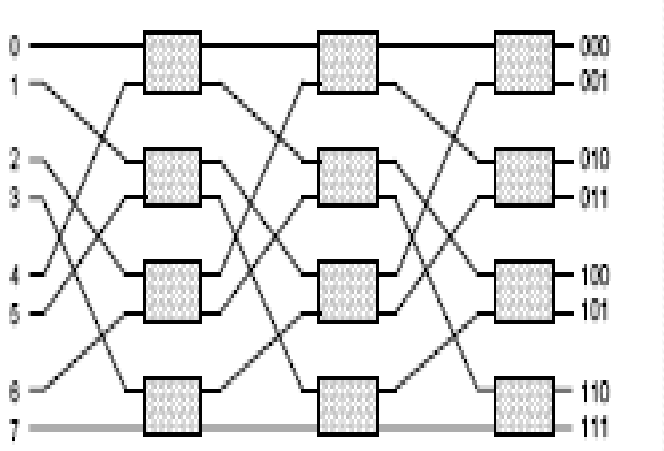
Çapraz Bağlantı Ağları (Crossbar Networks)



Çapraz bağlantı ağı alan maliyetinin hayli yüksek olduğu büyük bir anahtardır.
Öte yandan, L2 önbelleğindeki öbeklere(banks) gittiği sürece farklı işlemci çekirdeklerinin eşzamanlı önbellek erişimlerini ele almak kolaydır.
Bu CMP'lerde önemli bir avantajdır ve şu anki CMP'lerin bir kısmı çekirdeklerini L2 önbellek öbeklerine bağlamak için çapraz bağlantıyı kullanırlar.

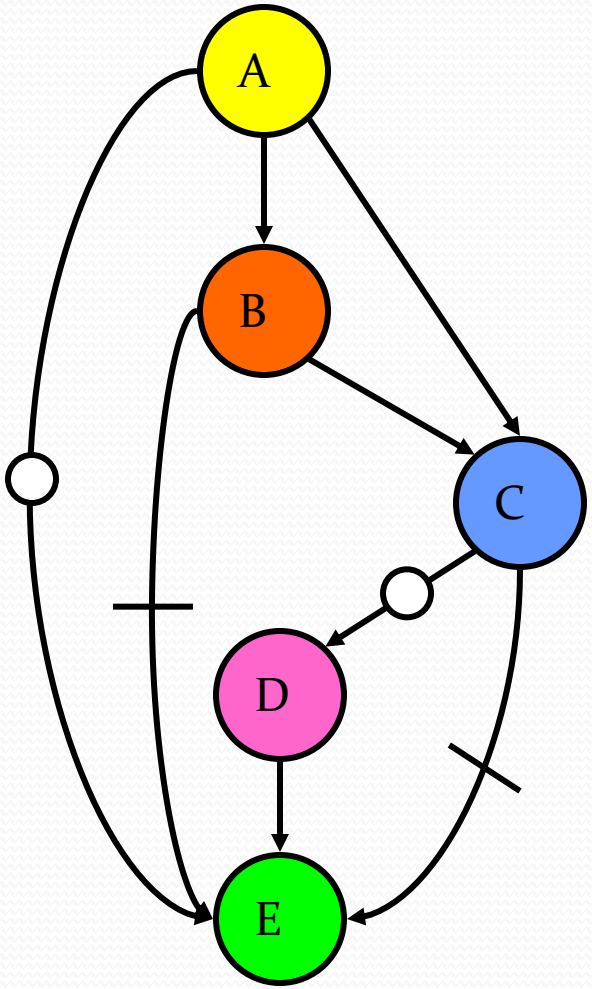
Çok Seviyeli Arabağlantı Ağları

- Çok seviyeli arabağlantı ağları mesajın varacağa yere kadar, anahtarlamalı evrelerden geçtiği dolaylı ağlardır. Birçok olası topoloji anahtarların birbirlerine nasıl bağlandıklarına göre farklı türleri mevcuttur.



CMP Komut Scheduling Algoritması

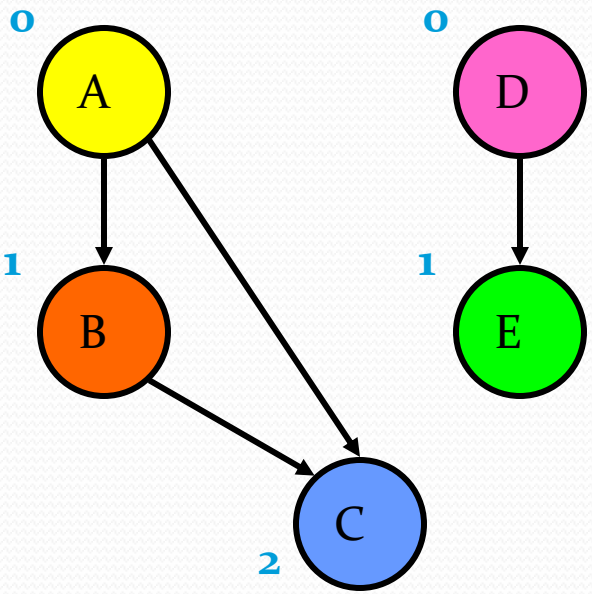
```
A: add   r1 = r3, r4
B: addi  r2 = r1, 6
C: mul   r6 = r2, r1
D: ld    r6 = 8(r5)
E: add   r1 = r6 + r6
```



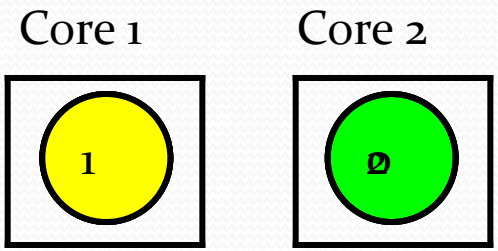
1.Program bağımlılık grafını çiz.
2.Yanlış bağımlılıkları sil.
3.Her komutu derinliğine göre numaralandır.

CMP Komut Scheduling Algoritması

```
A: add    r1 = r3, r4
B: addi   r2 = r1, 6
C: mul    r6 = r2, r1
D: ld     r6 = 8(r5)
E: add    r1 = r6 + r6
```

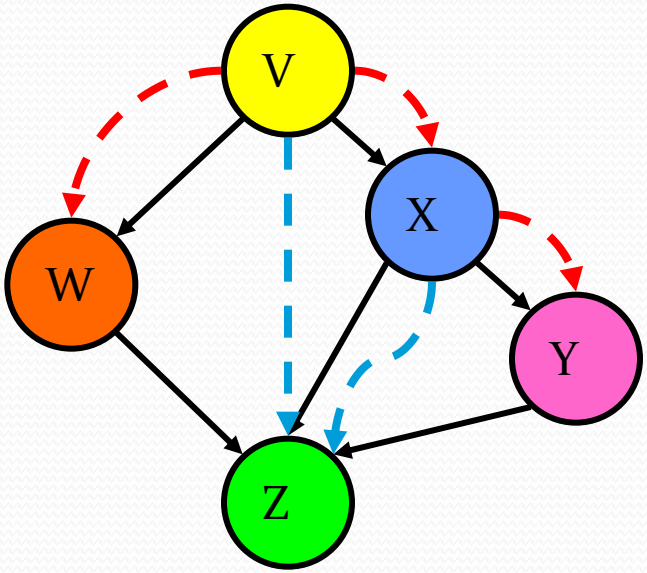


Bufferlar

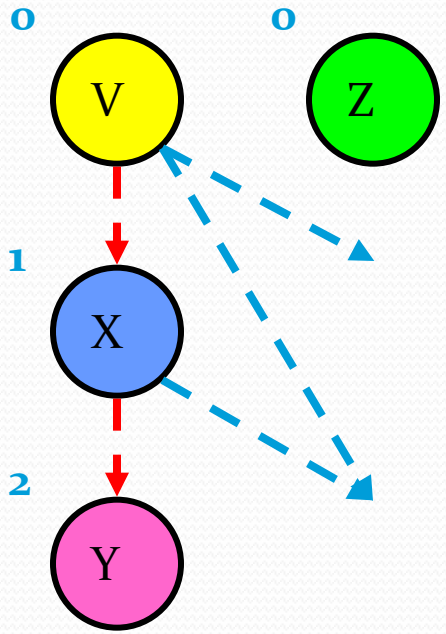


Dallanma Çözümlemesi

Kontrol Akış



İz Bağımlı Graf



Kontrol Bağımlılığı

Tutucu Kontrol Bağımlılığı

Kaynaklar

- **COMPUTER SYSTEMS ARCHITECTURE,**Mano,3RD/1993
- **Computer Organisation and Design**: The Hardware/software Interface, **Patterson**,Hennessy Patterson,Hennessy, 3d ed.,2005
- **CHIP multiprocessors and Usages,** Lappeenranta University of Technology Information Technology, Seminar Document,2007. www.it.lut.fi/kurssit/07 - 08/CT30A7000/seminars/Group17document.pdf
- **To Include or Not To Include:The CMP Cache Coherency Question,report,2007**
- **Investigating CMP Synchronization Mechanisms**,Chakraborty,Vaidyanathan,Wells,2003, pages.cs.wisc.edu/~david/courses/cs838/projects/pwells.pdf
- **Cooperative Caching for Chip Multiprocessors**,Jichuan Chang and Gurindar S. Sohi, www.cs.wisc.edu/mscalar/papers/2006/isca2006-coop-**caching**.pdf
- **CMP Workshop Papers** ,http://www.cse.ucsd.edu/~rakumar/dasCMP/dascmp.htm
- **Performance Implications of Single Thread Migration on a Chip Multi-Core** www2.cs.ucy.ac.cy/carch/xi/papers/**Migration**CAN.pdf
- **Multiprocessors for DSP** , www.site.uottawa.ca/~mbolic/elg6163/ELG6163_Burton.pdf
- **Improving the Performance of Parallel Applications in Chip Multiprocessors with Architectural Techniques NTNU**, 2007 idi.ntnu.no/~dam/internt/publikasjonsdatabase//pubfiles/JahreMasterThesis.pdf

Sorular

Teşekkür Ederim