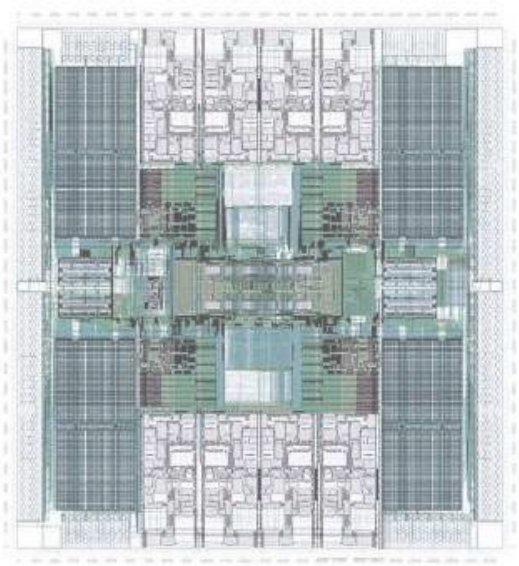
BİLGİSAYAR MİMARİSİNDE YENİ YAKLAŞIMLAR



CHIP MULTIPROCESSOR ARCHİTECTURES

ÇOK İŞLEMCİLİ KIRMIK MİMARİLERİ

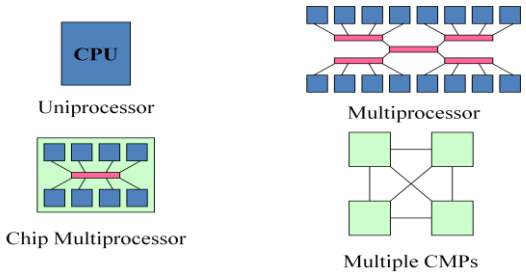
İNCİ CABAR 704061016

07

İÇİNDEKİLER

1.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.

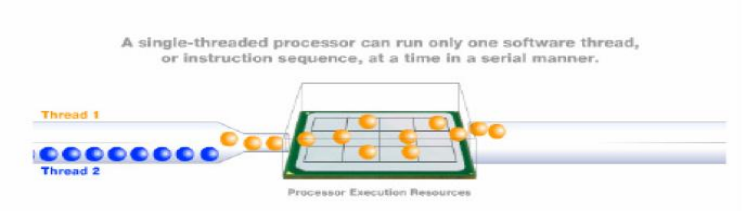


Şekil 1: CMP

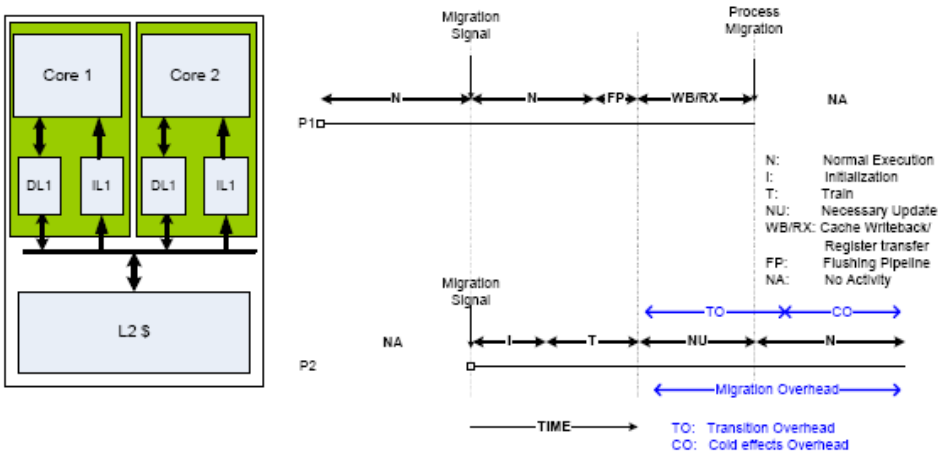
1.1 Çok İşlemcili Kırmıklarda Önemli Tasarım Kriterleri

- **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.



Şekil 2: Tek İplikçikli İşlemci



Şekil 3: CMP'de iplikçiğin P1 çekirdeğinden P2 çekirdeğine gitmesi

- **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.

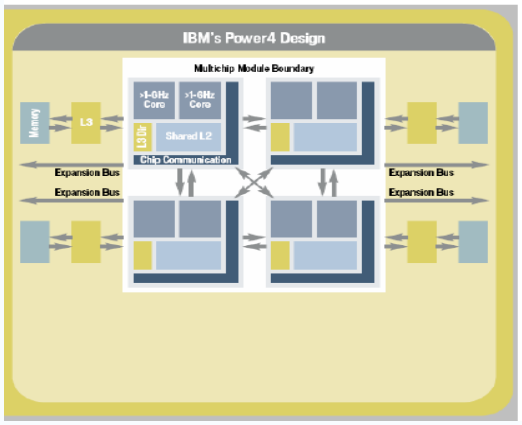
- **Bellek hiyerarşisindeki düzey sayısı:**

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.

1.2. Çok İşlemcili Kırmıklar

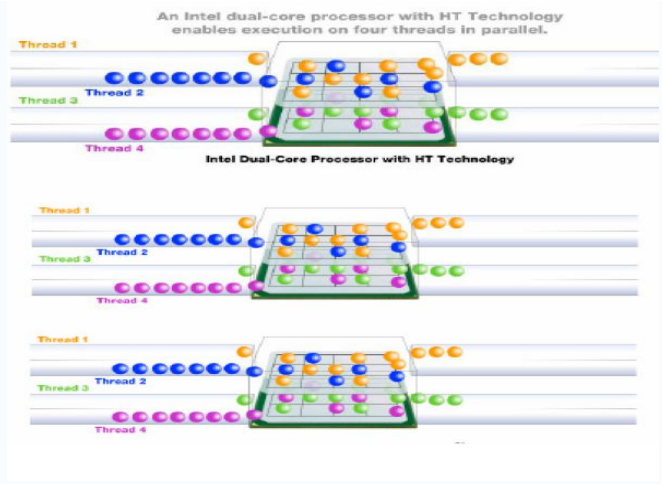
Günümüzde belli başlı kullanılan CMP'ler şunlardır:

- AMD Opteron server/workstation işlemcileri 2005,
  Çift çekirdek masaüstü işlemcileri, Athlon 64 X2 , 2005.
  Athlon 64 FX ,FX-60, FX-62 FX-64
  dizüstü işlemcileri Turion 64 X2
  quad-core ve triple-core işlemcileri 2008
- Sony PlayStation 3 te kullanılan ve IBM, Sony, Toshiba tarfından üretilen Cell
- IBM POWER4, 2000
  POWER5
  POWER6



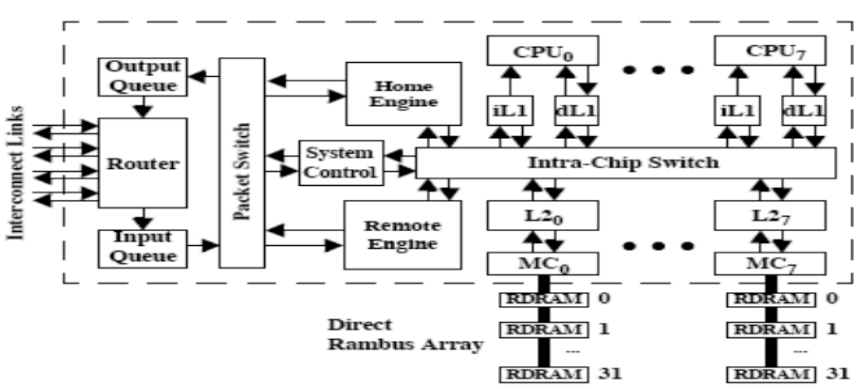
Şekil 3:IBM POWER 4 Mimarisi

- Intel quad core işlemcisi Core 2 Quad ve Xeon 2006
  Core Duo,
  Core 2 Duo



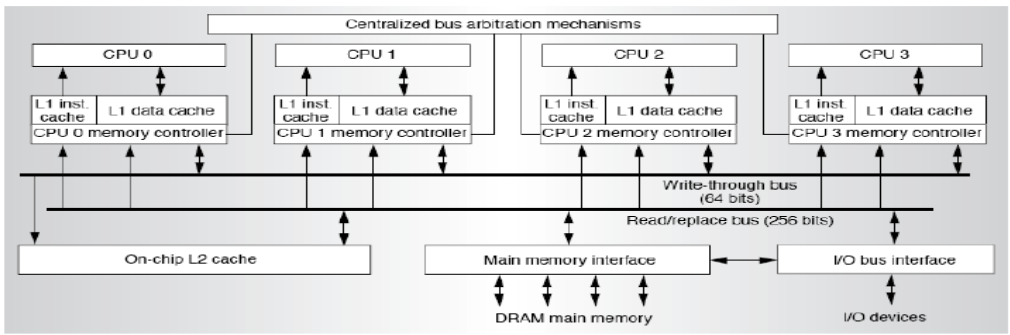
Şekil 4: İntel CMP

- Atmel Diopsis mAgic VLIW DSP
  ARM RISC MCU
- Sun Microsystems UltraSPARC IV,
  UltraSPARC IV+,
  UltraSPARC T2
- Compaq Piranha



Şekil 5:Piranha Mimarisi

- Stanford Hydra



Şekil 6:Hydra Mimari

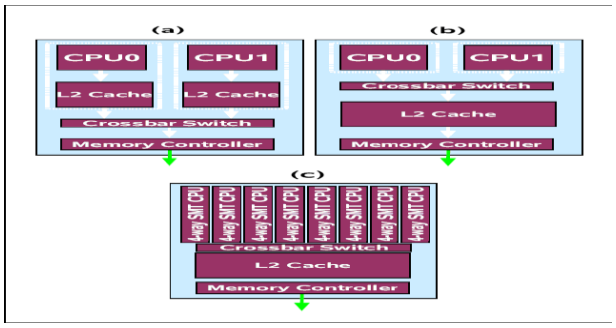
1.3 Çok İşlemcili Kırmık Çeşitleri

Çok işlemcili kırmıklar farklı mimarilere sahiptir.

1.3.1 Homojen Çok İşlemcili Kırmıklar

1.3.1.1 Geleneksel (Traditional) Çok İşlemcili Kırmıklar

Çoğu CMP mimarileri L2 önbelleği ve belleği paylaşırlar, bunları geleneksel CMP'ler olarak adlandırabiliriz.

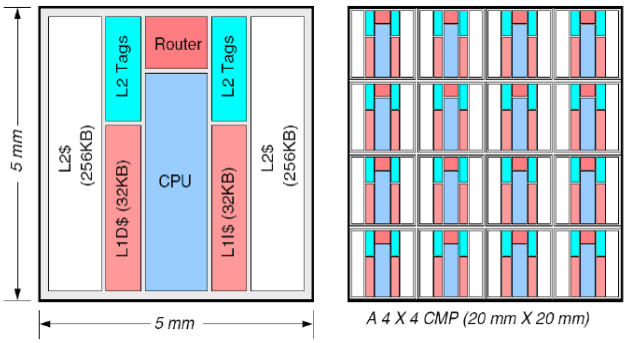


Şekil 7:Geleneksel CMP'ler

Şekilde son yıllarda CMP'lerin gelişim trendleri görülmektedir. İlk nesil CMP'ler a) da görüldüğü gibi sadece kırmık-içi bellek denetimcisini paylaşırken şekil 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. Şekil 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. Şekilde Sun'ın 32 yollu CMT işlemcisi ULTRASPARC T1 (diğer adıyla Niagara) görülmektedir. Bu anlayışı izleyen bir diğer mimari de Compaq'ın Piranha'sıdır.

1.3.1.2 Döşenmiş (Tiled) Çok İşlemcili Kırmıklar

Yakın zamanda Zhang ve Asanovic tarafından döşenmiş çok işlemcili kırmıklar ortaya atılmıştır. Burada işlemci çekirdeği , L1 önbelleği, L2 önbelleği ve iletişim yönlendiricisi bir döşe üzerine konulmuştur. Bu döşeler kırmık boyunca yinelenip daha sonra anahtarlanmış ağla bağlanırlar. Bu mimariyi geleneksel mimariye üstün kılan yanı teknoloji geliştikçe tellerin gecikmesinin (wire delay) azalmasıdır. Bu mimari yapı geleneksel yapıya bir alternatif sunar.



Şekil 8: Döşenmiş CMP

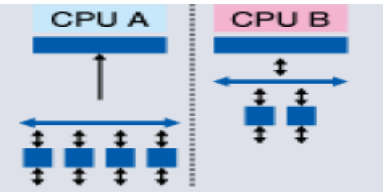
1.3.1.3 Birleştirilmiş Çekirdek (Conjoined Core) Çok İşlemcili Kırmıklar

Kumar'ın önerdiği bu tip mimarilerde birbirine yakın olan çekirdeklerdeki kaynaklar paylaşılır. Amaç performansın düşmesini minimumda tutarken, daha az alan kullanmaktır. Kaynakları kullanan çekirdekleri yerleşim planıyla beraber tasarlamak önemlidir, böylece kaynakları hızlıca uyumlu hale getirmek için gereken kablo bağlantıları önemli hale gelir. Kaynak paylaşımı ; kayan-noktalı birimler, çapraz bağlantı portları, komut önbellekleri, veri önbellekleri için dikkate alınır.

Kumar bu yöntemle çekirdeklerde %9-12 performans artışı elde etmiştir ve çekirdek alanı %50 küçülmüştür.

1.3.2 Heterojen Çok İşlemcili Kırmıklar

Şu ana kadar gördüğümüz CMP'lerde bütün çekirdekler aynı performans özelliklerine sahiptiler. Fakat, uygulamalar farklı sonuçlar olabilir ve bir çekirdek tasarımı bir uygulama için yüksek ,bir diğer uygulama için ise düşük performansla sonuçlanabilir. Buna dayanarak araştırmacılar heterojen CMP'leri önermişlerdir. Burada CMP'ler farklı performans özellikli ,farklı çekirdeklere sahiptir. Bu da ya daha az güç tüketimi, ya daha yüksek iş üretimi (throughput) veya her ikisi de demektir.



Şekil 9:Heterojen CMP

Kumar iki farklı tipte çekirdeğe sahip heterojen bir CMP nin tatmin edici bir performans sağladığını söylemektedir. Ayrıca, basit bir zamanlama algoritmasının ,en iyi statik iplikçik zamanlamasına göre daha iyi performans sağladığını göstermiştir.

Burada artan verimlilik kendiliğinden gelmemektedir. İşletim sistemi kararlı ve ölçeklenebilir performansa erişebilmek için asimetrinin farkında olmalıdır. Genelde yazılımlar geleneksel yöntemlere göre yapılmaktadır. Bu yöndeki yazılımlar geliştikçe heterojen CMP'lerin seyri farklı bir hal alacaktır.

2. Uygulama Düzeyinde Çok İşlemci Programlanması

Tek-iplikçikli program genelde CMP'lerde tek çekirdekli işlemcilere göre daha uzun bir yürütme zamanına sahiptir. Ayrıca, en iyi durum, iki işlemcininde yürütme zamanının aynı olduğu durumdur. Öte yandan bazı programlar aynı zamanda çalışabilir, böylece daha fazla iş aynı zaman diliminde yapılabilir. Diğer bir deyişle üretilen iş artar. Eğer programlar paraleleştirilirse daha hızlı çalışabilirler. Bu anlamda CMP'lerde peformans iyileştirmesi sağlanabilir.

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.

Hareketli Uyumluluk ve Tutarlılıkta (Transactional Coherence and Consistency),bir transaction; paralel işin, iletişimin ve uyumluluğun esas birimidir. Program farklı transactionlara bölünmüştür ve her transaction yeni durumunu belleğe bildirir.

TCC' nin özellikleri şunlardır:

- Transaction kritik bölgede başlamadığı veya bitmediği sürece, program doğru çalışır. Transactionlara bölümlendirme bir performans sorunudur. Diğer bir deyişle doğru paralel programı yazmak kolaydır. Fakat transactionlara ayırma programcı tarafından yapıldığından bunu verimsiz bir yöntemle yapmak mümkündür. Sonuçta da doğru sonuçlar üreten fakat düşük performansı olan bir program olabilir.
- Paralel bir program çalışan seri bir programı genişleterek geliştirilebilir.
- Yüksek performansa erişilebilir.

3. 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.

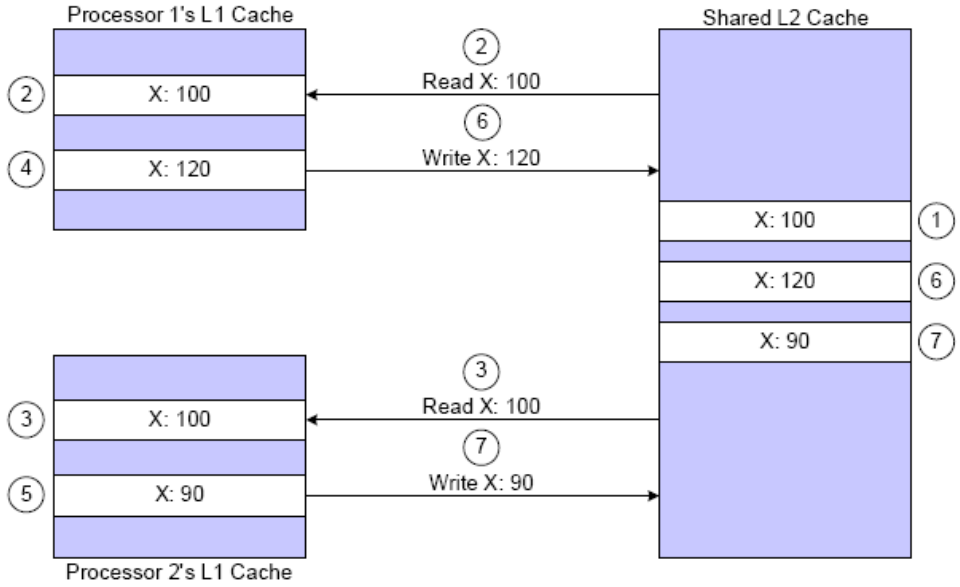
Şimdiki konuda bunlar daha ayrıntılı incelenecektir.

4. Çok İşlemcili Kırmıklarda Önbellek Uyumluluğu ve Tutarlılığı

4.1. Önbellek Uyumluluğu

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, güncelleştiklerinde 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.



Şekil 10: Önbellek Uyumluluğu Sorunu

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 önbelle ğin de 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 önbelleğindekideğer 90 olmuştur ve 1 no'lu işlemci tarafından yapılan değişiklik kaybolur.

Donanıma dayalı önbellek uyumluluğu protokollerini ele alalım. Hennesy ve Petarson bellek sistemini aşağıdaki şartları sağlıyorsa uyumlu olarak tanımlıyor:

- Program sırası: X konumuna P işlemcisi yazıyor, daha sonra P işlemcisi X' ten okuyor. P'nin yazma ve okuması arasında X konumuna başka bir işlemci yazmıyor. Bunun sonucuda P tarafından yazılan değer döner.
- Belleğin uyumlu görünümü: Bir işlemci X' e yazıyor, daha sonra başka bir işlemci X' ten okuyor. Eğer yazma ve okuma, zaman açısından yeteri kadar ayrık ise ve bu iki erişim sırasında X'e başka bir yazma olmamışsa, yazılı değer döndürülür.
- Yazma sıralaması: Aynı konuma yazmalar serileştirilir , herhangi iki işlemci tarafından aynı yere yapılan iki yazma, tüm işlemciler tarafından aynı sırada görülür.

4.2. Önbellek Tutarlılığı

Önbellek tutarlılığı, bir işlemcinin yazdığının hemen diğer işlemciler tarafından görülemeyeceğini belirtir. Programcının bellek sistemini kullanabilmesi için bellek isteklerinin nasıl sıralandığının kesin bilinmelidir. Bu tanım da bellek tutarlılığıdır. Ardışıl tutarlılık en açık tutarlılık modelidir . Buna göre aynı işlemciden tüm bellek erişimleri bir sırada tutulsa ve farklı işlemcilerden erişimler keyfi olarak aralıklara ayrılsa, herhangi bir yürütmenin sonucu aynıdır. İşlemci bellek erişiminin bitmesini beklediği için, performans kötü etkilenebilir. Bu problemi gevşemiş tutarlılık modelleri ile çözeriz. Burada yazma operasyonları out of order tamamlanabilir ve senkronizasyon işlemleri program sırasını güçlendirmek için kullanılır. Sonuçta,program, ardışıl tutarlılık uygulanmış gibi davranır.

4.3. 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.

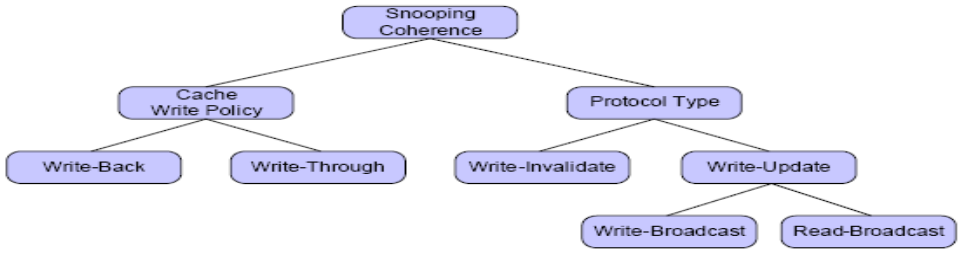
Verimlilik düşüncesiyle her bir işlemciye bir tane özel önbellek bağlanması istenir. Bu yöntem sadece paylaşılmayan ve yalnız okunabilen verilerin önbelleğe yazılmasıdır. Düzenleyici verileri önbelleklenebilir ve önbelleklenemez olarak ayırır. Önbelleklenebilir veriler önbelleklere yazılırken önbelleklenemez 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' ların tüm önbelleklerde kopyaları vardır, RW bloğunun ise tek kopyası bulunur .Böylece veriler önbellekte RW bloğuyla güncellenirse diğer önbellekler etkilenmez .

Şimdi donanım temellileri görelim.

4.3.1 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.



Şekil 11: Trafik Gözetleme Önbellek Protokol Olasılıkları

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 yokur.

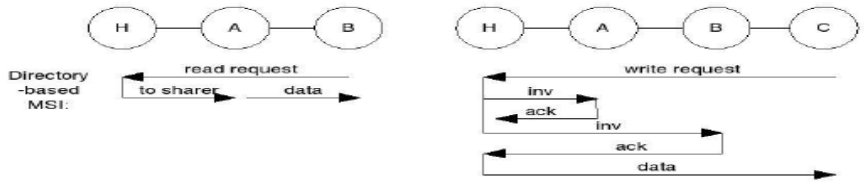
•Protokol daha güçlü arabağlantılar kullanılacak şekilde güncellenir. Örneğin protokoller halka(ring) arabağlantı ile çalışabilirler.

4.3.2 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 verileln 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 12: Dizin Protokolü

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.

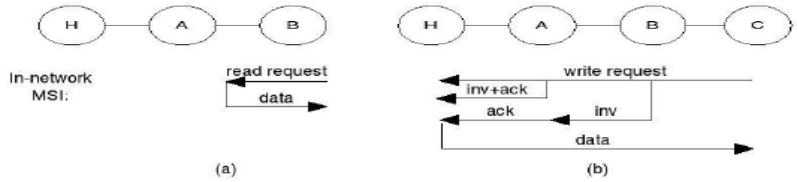
Şimdi iki dizin protokolünü görelim:

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

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

4.3.2.1 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 13: In-Network Önbellek Uyumluluğu Optimizasyonu

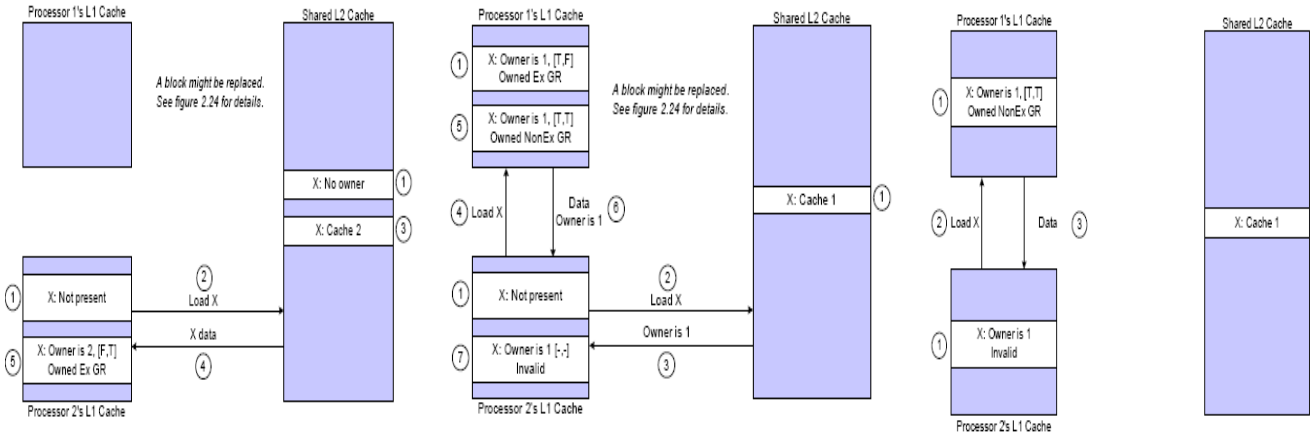
Şekil a'da okuma isteği için arabağlant ağı geçilip veri direkt olarak temin edilir, gecikme azalır. Şekil b'de İn-network protokolü protokol eylemlerini, yazma isteğini tüm paylaşanlardan geçirerek, optimize eder. A ve B geçersiz kılma mesajlarını C'nin isteğini görünce işleme koyarlar. Sonra H; tüm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

4.3.2.2 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.

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 ertelemesiz 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.

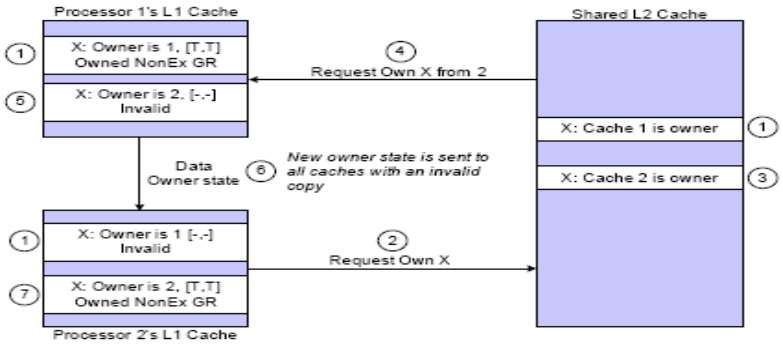


a) b) c)

Şekil 14: Okuma Iska

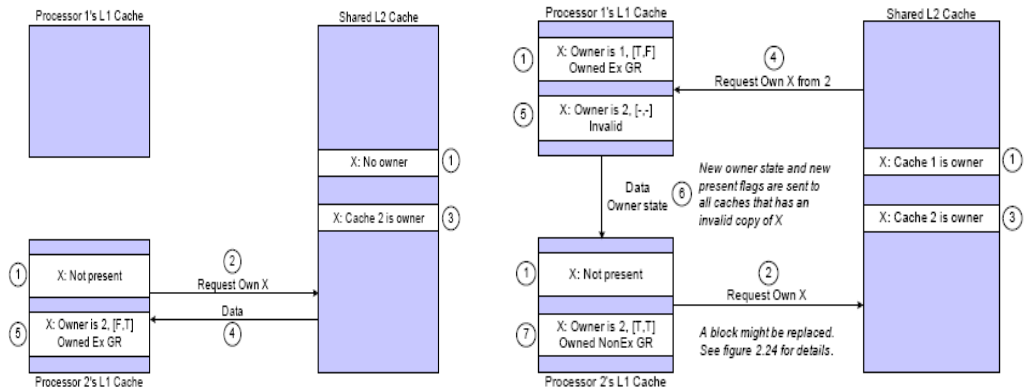
Şekil a'da sadece paylaşımlı L2 'de X'in koyası vardır. Diğer önbelleklerde X' in kopyası yoktur. Şekil b'de en azından bir tane başka önbellekte X'in kopyası vardır. Şekil c'de istenen önbellekte X in geçersiz bir kopyası vardır.

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. İkincisinde blok, Owned NonExclusively Global Read durumunda olabilir. Burada, bloğun diğer tüm kopyaları geçersiz olduğundan ertelemesiz yürütülür. Son olarak önbellek hattının geçersiz olma durumundaysa önbellek bloğa yazmadan önce sahip olmak ister.



Şekil 15:Yazma Vuru

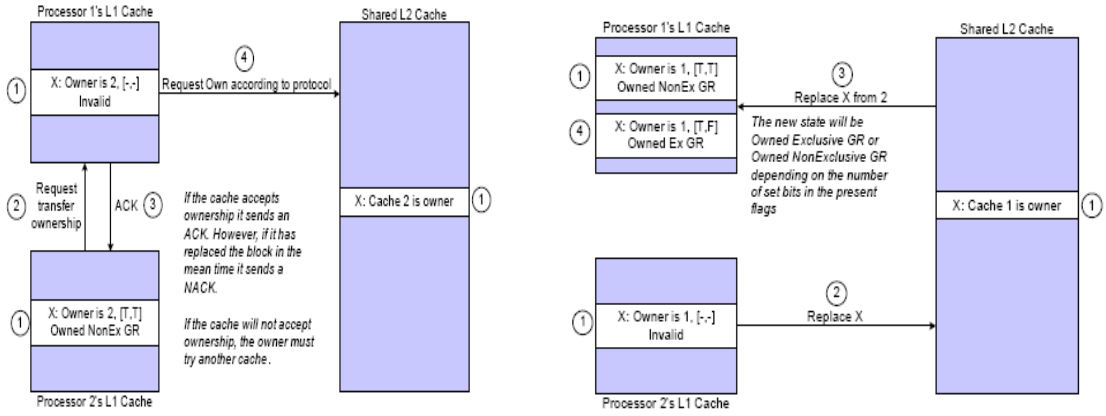
Yazma Iska Durumu: Sadece sahibi bloğa yazabilir. Bu yüzden yazmadan önce önbellek bloğun sahipliğini elde etmelidir. İki durum sözkonusudur. İlki blok L1'de mevcut, ikicisinde ise L2 den getirilmeli.



Şekil 16: Yazma Iska

Şekil a'da L1 önbelleklerinde bloğun mevcut olmama durumu incelenmiştir. Yani diğer önbelleklerde X 'in kopyası yoktur. Şekil b'de ,en azından bir tane başka önbellekte X in kopyası vardır.

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.



a) b)

Şekil 17:Blok değiştirme

Şekil a bloğun yalnız sahiplenilmediği durumdur. Burada önbellek bloğunun sahipliği farklı bir L1 e transfer edilir. Yeni sahip mevcut bayraklardan rastgele seçilir. Şekil b ise bloğun geçersiz olduğu durumdur.

5.Çok İşlemcili Kırmıklarda Senkronizasyon (Synchronization) Mekanizmaları

Çok işlemcili kırmık senkronizasyonu için 3 temel kabul yapıyoruz:

- CMP' nin özellikleri ve iş yükü: hızlı kırmık-içi veri transferleri,
- Göreceli büyük miktarda kırmık-içi band genişliği,
- Göreceli seyrek senkronizasyon

Buna dayanarak spin-lockların CMP'ler için çekici bir mutex mekanizması olduğu söylenebilir. Çok daha yavaş olan kırmık-dışı iletişim ve kırmık-dışı band genişliği nedeniyle CMP senkronizasyonunda ödünler verilebilir. Bu tür durumlarda kuyruk bazlı kilitler (queue-based locks) iyi bir alternatif olabilir.

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.

Atomik komutlar gibi donanım senkronizasyonu mekanizmaları mimari tarafından desteklenmelidir. Lock-based Test-and-Set, Fetch-and-Increment, bunlardan bazılarıdır. Lock-free komutları ise Compare-and-Swap ve Load-Linked/Store-Conditional gibi komutlardır.

6. Çok İşlemcili Kırmıklarda İletişim

6.1 Paylaşımlı Bellek Kullanarak İletişim

Paylaşımlı bellek iletişimi CMP'lerde çok verimli olmaktadır. Bunun sebebi hem belleğin hem de işlemci çekirdeklerinin aynı kırmıkta olmasındandır. Belli bir düzeyde önbellek paylaşımı çatışma ıskalarına (conflict misses) sebep verir. Bu ıskalar sadece set-associative veya direct mapped önbelleklerde ortaya çıkar ve bir blok, tüm setler meşgul olduğu için bekletilir veya atılır. Çatışma ıskalarının artmasının sebebi, farklı iplikçik bloklarının, paylaşımlı önbellekte aynı şekilde haritalanmasıdır. Yani CMP'deki tüm iplikçikler bu önbellek paylaşımından adaletsiz bir şekilde etkilenir ve bu adaletsiz paylaşım iplikçik starvationına veya öncelik evrikliğine neden olabilir.

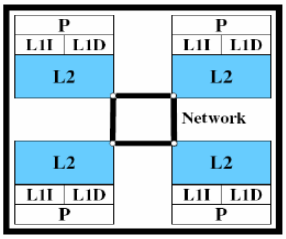
6.1.1 Özel (Private) L1 ve Özel L2 Önbellekler

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

Burada iki tane sorunla karşılaşılır. İlkinde tüm iletişimlerin bellek tarafından gözden geçirilmesi gerekir, bu da paylaşımlı önbellekle olan iletişime göre çok daha yavaştır. Diğer sorunsa, zayıf kırmık genişliği kaynak kullanımıdır.(chip-wide resource utilisation).Bu da kırmık-dışı önbellek ıskalarını arttırır.

İmeceli Önbellekleme(Cooperative Caching)

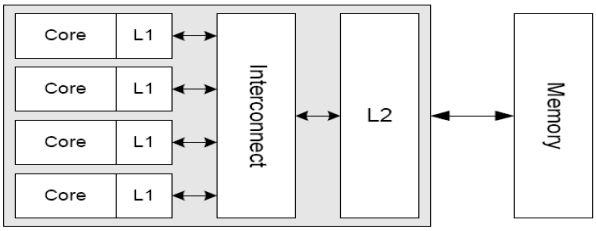
Yakın bir zamanda Chang ve Sui tarafından sunulan bu teknikle kırmık-dışı önbellek ıskaları azalırken özel L2 nin sağladığı yararlardan faydalanılmıştır. Burada işlemci çekirdeği farklı işlemcilerin L2 çekirdeklerine veri koyabilir.



Şekil 11:İmeceli Önbellekleme

6.1.2 Özel L1 Önbellekler ve Paylaşımlı L2 Önbellek

Bu tür CMP'ler ticari olarak çok kullanılır. Intel, AMD, Sun ve IBM' in paylaşımlı L2 önbellekli CMP modelleri vardır.



Şekil 12: Özel L1 Önbellekler ve Paylaşımlı L2 Önbellek

Bu tür bir yapı çekirdekler arsında hızlı bir iletişim ve hızlı bir vuru zamanı sağlar. Bu hızlı vuru zamanı neredeyse tüm bellek erişimlerinin L1 önbelleği üzerinden yapılmasıyla olur. Buna rağmen L2'nin vuru zamanındaki küçük artış, büyük bir performans artışına neden olmaz. Bu yüzden L2 yeteri kadar büyük tasarlanabilir. Ayrıca fazladan pin eklemek pahalıdır. Kırmık-dışı band genişliğini verimli kullanmak önemlidir. Kırmık-dışı bellek erişimlerinde azalma iyi önbellek kullanımının sonucu olabilir. 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.

6.1.3 Paylaşımlı L1 Önbellek

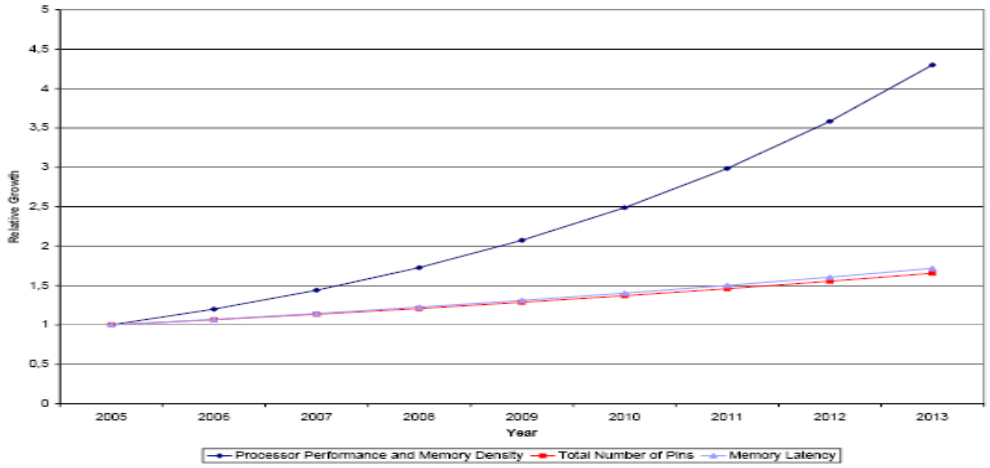
Çok hızlı bir kırmık-içi iletişim söz konudur. Fakat yüksek seviyede tek-iplikçik performansı, yüksek frekansa bağlıdır. Sun MAJC mimarisi örnek verilebilir.

6.2 Mesaj Geçme Yolu ile İletişim (Communication by Message Passing)

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

7. Çok İşlemcili Kırmıklarda Arabağlantı (Interconnection) Mimarileri

Çok işlemcili kırmıklarda, farklı çekirdekler birbirlerine veya ortak bir yapıya bağlanabilir. Bu bölümde CMP'lerin arabağlantı tasarımlarına değinilecektir. Bunlardan bazıları küçük alan ve düşük band genişliği gerektirirken bazıları büyük alan ve yüksek performans gerektirirler.



Şekil 13: CMP'lerde Performans, Pin Sayısı ve Bellek Gecikmesi

7.1 Paylaşımlı Ortam Ağları (Shared Medium Networks)

Paylaşımlı ortam ağları için 3 olasılık incelenecektir:

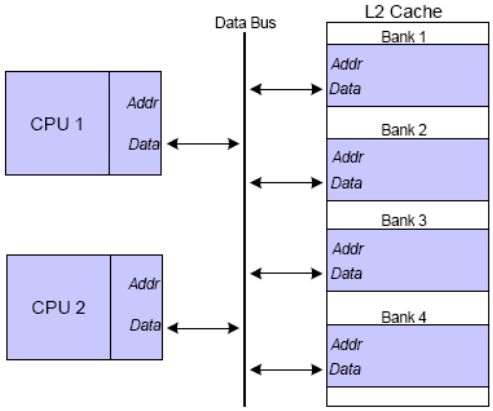
- Çekişmeli veriyolu(contention bus)
- Andaçlı veriyolu (token bus)
- Hakemli veriyolu(arbitrated bus)

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.



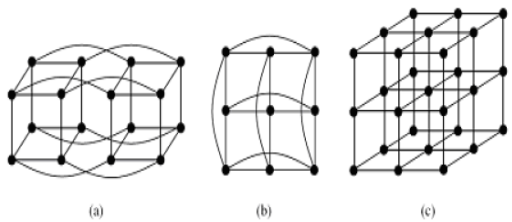
Şekil 14: Paylaşımlı veriyolu

Dört çekidekli Hydra çok işlemcileri, iki tane paylaşımlı veri yolunu her çekirdeğin write-through L1 önbellekleri ile kırmık-içi L2 ön belleklerini bağlamak ve kırmık-dışı L3 önbellek denetçisini, belleği ve G/Ç denetçilerini bağlamakta kullanırlar.

Paylaşımlı veriyolu ağının avantajı basitliğidir. Fakat, veriyolu birçok birim tarafından paylaşıldığından performans sorunu yaşanabilir. Birçok iletişim birimi olan sistemlerde yüksek performans için başka çözümler gerekir.

7.2 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



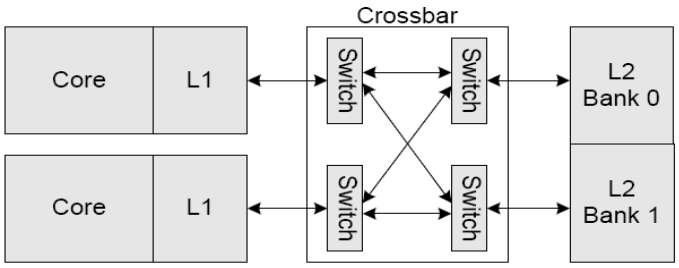
Şekil 15: Doğrudan Ağ Topolojileri

7.3.Dolaylı Ağlar (Indirect Networks)

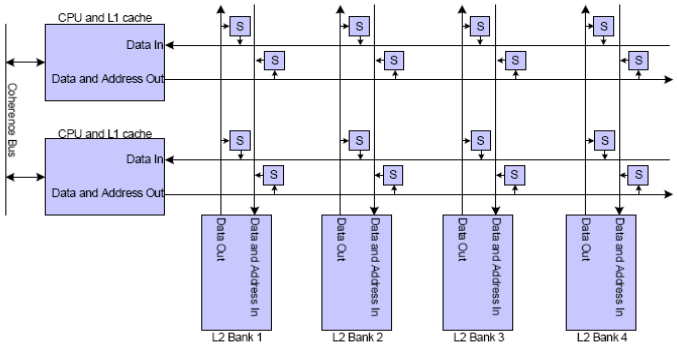
Bu ağlar doğrudan ağlara benzerler. Temel farkı bir dolaylı ağın düğümleri arasında anahtarlar olmasıdır. Bir başka deyişle, düğüme enjekte edilen bir mesaj varış düğümüne ulaşmadan belli sayıda anahtardan geçer. Bir anahtarın en azından bir giriş ve bir çıkış portu vardır ve düğüm anahtarlaması giriş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

7.3.1 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.
8 çekirdekli Piranha da ,L1 önbelleklerini ,paylaşımlı L2 önbelleğine bağlamak için çapraz bağlantı kullanır.



Şekil 16: 2 öbekli L2 önbelleğe sahip , 2 çekirdekli CMP 'de çapraz bağlantı

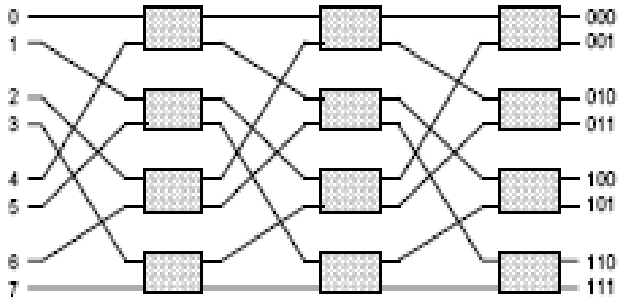


Şekil 17:Çapraz Bağlantı Topolojisi

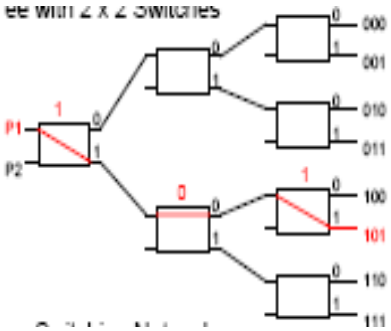
Çapraz bağlantı ağı, ek alanı tolere edebildiği sürece iyi bir alternatiftir.

7.3.2 Çok Seviyeli Arabağlantı Ağları (Multistage Interconnection Networks)

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



Şekil 18: 8x8 Çok Seviyeli Ağ



Şekil 19: 2 x 2 Anahtarlı Binary Ağacı

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