Bil361 HW2 Rapor
Ahmet Hakan Aksu 191101068

1. Soru:

Test programı 16.384 yazma ve 16.384 okuma işlemi yapmaktadır.

Önbellek olmadan: Her erişim 25 çevrim

32.768 x 25 =819.200 çevrim

Önbellek ile:

Yazmalar 16.384 x 25 = 409.600 çevrim (cache miss)

Okumalar 16.384 x 1 = 16.384 çevrim (cache hit)

Toplam: 425.984 çevrim

Önbellek kullanımı, yaklaşık %48 daha hızlı sonuç vermiştir.

2. Soru:

Tasarımda iki sayaç eklendi:

sayac_erisim = 32.768

sayac_bulamama = 16.384

İlk yazmaların hepsi "miss", okumaların çoğu "hit" oldu.

(32768−16384) / 32768 = 0.5 = %50

Yüksek miktarda tekrar eden erişim olduğu için %50'lik oran oldukça verimli bir sonucu göstermektedir.

3. Soru:

Boş

4. Soru

Verilen test programı belleğe sürekli ve ardışık erişim yapan bir programdır. Bu gibi programlarda sistemdeki genel gecikmeyi azaltmak için önbellek tasarımında yapılabilecek bazı iyileştirmeler şunlardır:

1. Önbellek Boyutunu Artırmak
Mevcut önbellek boyutu 2KB'tır. Bu oldukça küçük bir kapasitedir ve bellekte tutulan veri sayısı sınırlıdır. Eğer önbellek boyutu örneğin 4KB, 8KB gibi artırılırsa, daha fazla veri satırı önbellekte tutulabilir. Bu da daha az sayıda önbellek bulamama (cache miss) oluşmasına yol açar. Miss oranı azaldığında, sistem daha az anabellek erişimi yapar ve bu da gecikmeyi azaltır.

2. Veri Öbeği Boyutunu Artırmak
Şu anda her veri öbeği 16 bayt uzunluğunda ve 4 adet 32-bit kelime barındırıyor. Eğer bu boyut 32 veya 64 bayta çıkarılırsa, ardışık adreslere erişen programlarda daha fazla veri aynı anda getirilebilir. Bu, mekanda yerellik (yani verilerin bellekte yan yana bulunması) ilkesinden daha fazla faydalanmayı sağlar. Böylece aynı veri öbeğinden daha fazla erişim hit olur.

3. Yazma Politikasını Değiştirmek
Şu anda sistem "write-through" (doğrudan yaz) politikasıyla çalışıyor. Bu da her yazmada ana belleğe erişilmesi gerektiği anlamına gelir. Eğer "write-back" (gecikmeli yaz) politikası kullanılsaydı, sadece güncellenmiş bloklar yazılacağından ana bellek erişimleri azalıp gecikme düşebilirdi.

4. Etiket Dizin ve Offset Daha İyi Ayarlamak
Adresin hangi kısmının etiket (tag), hangi kısmının dizin (index), hangisinin öbek içi yer (offset) olduğu tasarıma doğrudan etki eder. Test programı ardışık adreslere eriştiği için, daha az index biti, daha çok offset biti ile tasarım yapılırsa bir öbek içinde daha fazla ardışık veri saklanabilir.

5. Önbelleğe Önyükleme (Prefetching) Eklemek
Programdaki erişimlerin büyük kısmı ardışık adreslerdir. Bu durumlarda bir veri öbeği getirildikten sonra bir sonraki öbeğin önceden getirilmesiyle, olası cache miss durumları önlenebilir. Bu donanımsal ya da yazılımsal olarak yapılabilir. Test programına özel olarak, her yüklemeden sonra bir sonraki öbeği de yükleyen bir sistem düşünülmelidir.

5. Soru

1. Kümeli İlişkili (Set-Associative) Önbellek Kullanmak
Örneğin 4-yollu kümeli ilişkili bir önbellek yapısı kullanılırsa, birden fazla veri satırı aynı index'e bağlı olabilir. Bu sayede adres çakışmaları önlenir ve aynı dizine düşen ancak farklı etiketlere sahip bloklar önbellekte bir arada bulunabilir.

2. Daha Büyük Önbellek Boyutu
Matris çarpımı sırasında aynı veriler tekrar tekrar kullanılır. Bu nedenle 2KB yerine 8KB veya daha büyük bir önbellek kullanmak, verilerin önbellekte kalma süresini artırır. Böylece cache miss oranı düşer, performans artar.

3. Veri Öbeği Boyutunu Artırmak
Eğer öbek boyutu 16 bayt yerine 32 veya 64 bayt olursa, ardışık veri içeren satır veya sütunların daha büyük kısmı tek seferde getirilmiş olur. Bu da konum yerelliğini daha iyi değerlendirmemizi sağlar.

4. Write-Back ve Write-Allocate Politikalarını Uygulamak
Matris çarpımlarında aynı bellek hücresine çok sayıda yazma işlemi yapılır. Bu nedenle her yazmada anabelleğe erişmek (write-through) yerine sadece değişen blokları yazan "write-back" politikası kullanılabilir. Ayrıca, yazma sırasında veriyi önce önbelleğe getirip orada yazma işlemi yapan "write-allocate" politikası da verimi artırır.

5. Önceden Yükleme (Prefetching) Mekanizması
Modelin belleğe erişim düzeni tahmin edilebiliyorsa, bu erişimlerden önce verileri önceden önbelleğe getiren bir mekanizma eklenebilir. Örneğin bir satırın ilk elemanına erişildiğinde, aynı satırın tamamı yüklenebilir.

6. Çift Seviye (Multi-Level) Önbellek Yapısı
LLM gibi büyük modellerde sadece bir seviye önbellek yetersiz kalabilir. Bu yüzden, küçük ve hızlı bir L1 önbelleğin arkasında daha büyük ama daha yavaş bir L2 önbellek yer alabilir. Bu mimari, erişim gecikmelerini azaltır.

6. Soru

Zorluk 1: Handshake Protokolü (Ready-Valid)
İşlemci ile önbellek ve önbellek ile anabellek arasındaki iletişim hazır/geçerli sinyalleri (ready-valid handshake) ile yapılıyor. Bu protokolü ilk başta yanlış anladım ve sinyaller senkron çalışmadı.Sistemin her durumda sadece "geçerli ve hazır" sinyalleri birlikte aktifken veri ilettiğini fark ettim. Bu nedenle hem işlemci hem anabellek tarafı için kontrol sinyalleri yeniden düzenlendi.

Zorluk 2: Yazma İşleminin Eşzamanlı Yapılması
"Write-through" politikasında hem önbelleğe hem anabelleğe aynı anda yazmak gerekti. İlk başta sadece önbelleğe yazılıyordu ve test sonuçları hatalıydı. Çözüm olarak yazma işlemi sırasında hem data_array güncellendi hem de aynı veri anabelleğe gönderildi.

Zorluk 3: Durum Makinesi Geçişleri
Durumlar birbiriyle çakışıyor veya geçişler eksik olabiliyordu. Özellikle WAIT_MEM ve RESPOND (değişken isimlerini böyle kullandım) geçişlerinde sistem beklenmedik davranış gösterdi.Durumlar net bir şekilde ayrıldı, next_state değişkeni her durumda dikkatle belirlendi ve çevrim sayısı üzerinden testler yapıldı.