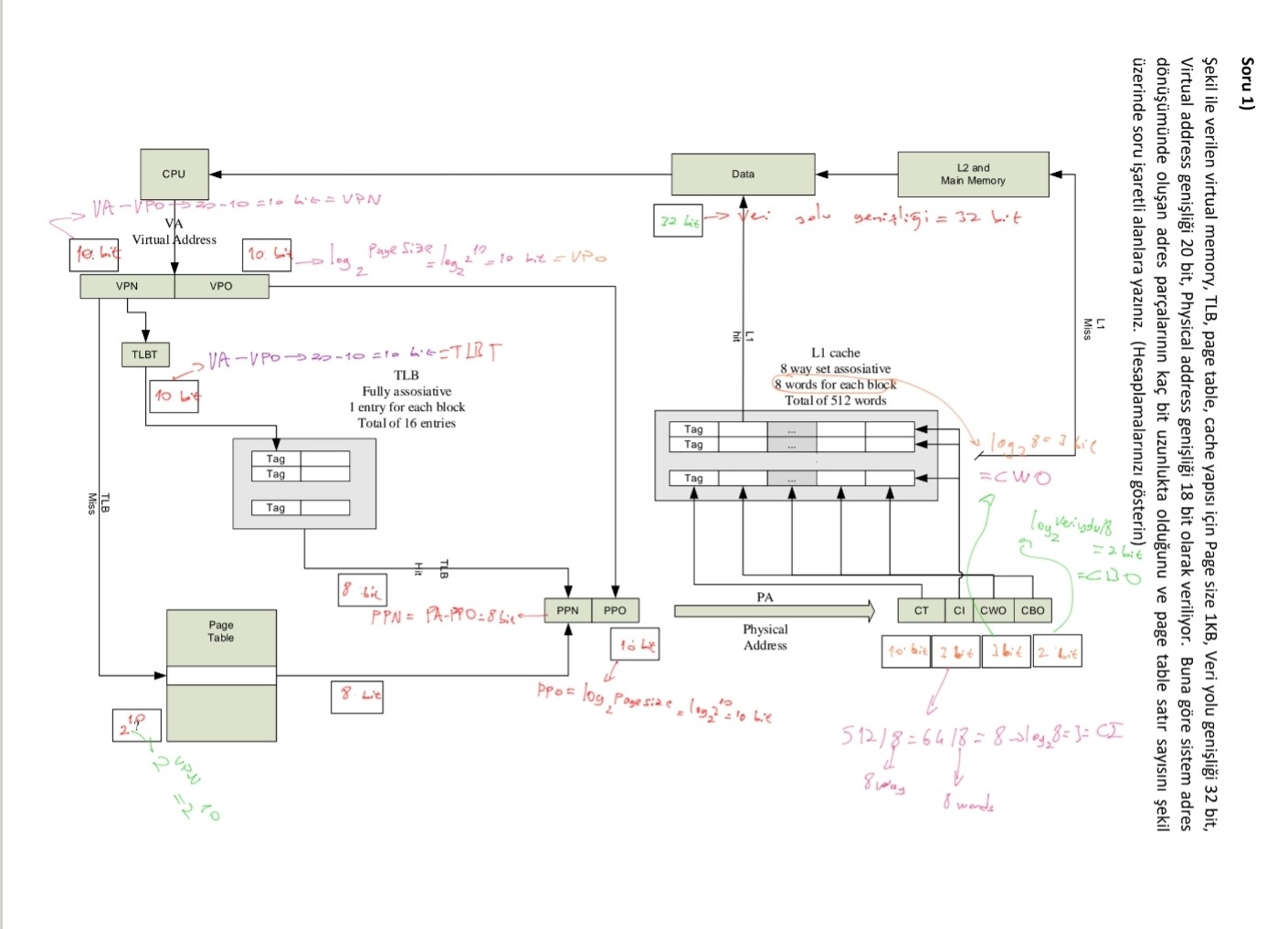


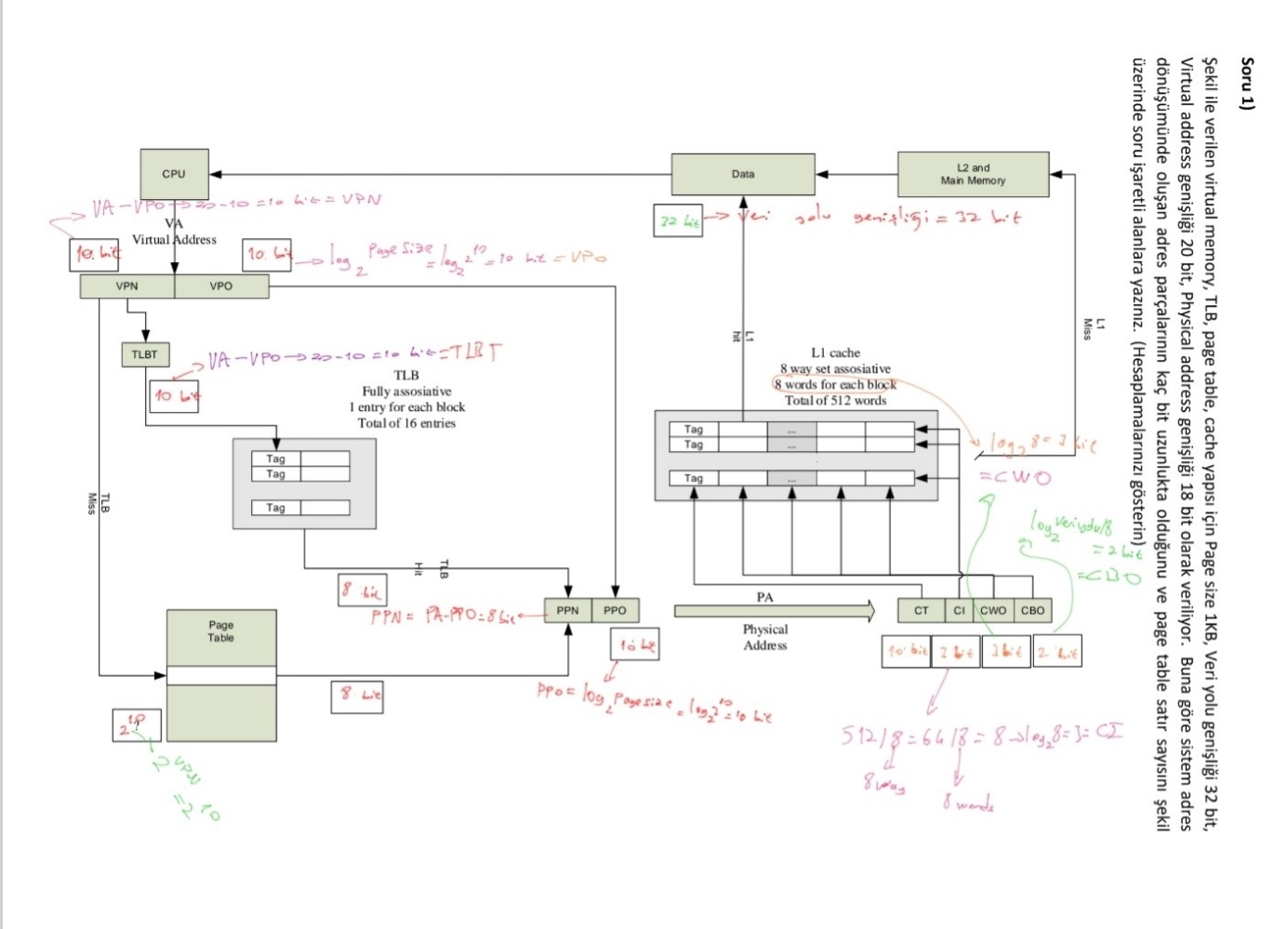
Bilgisayar Donanımı 2. Ödevi

Muhammet Kayra Bulut

20011901

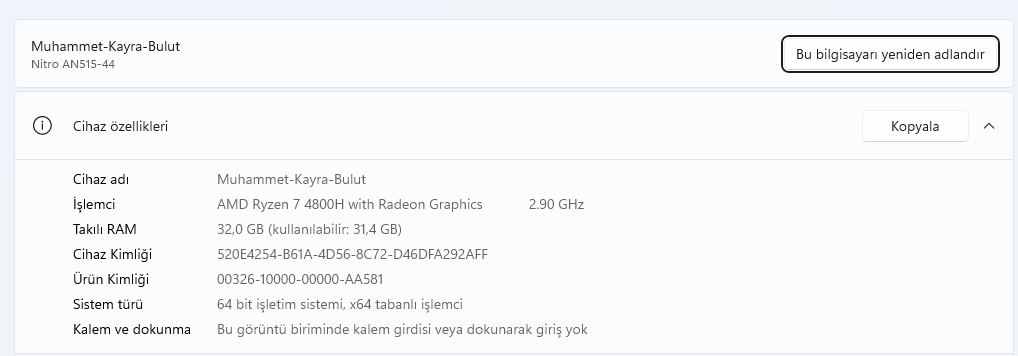
Soru 1 için boyutundan dolayı okunabilmesi için çözümümü dikey ve yatay şekilde koydum.



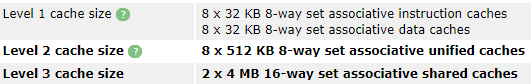


Soru 2 - )

Laptop işlemcimin modeli;

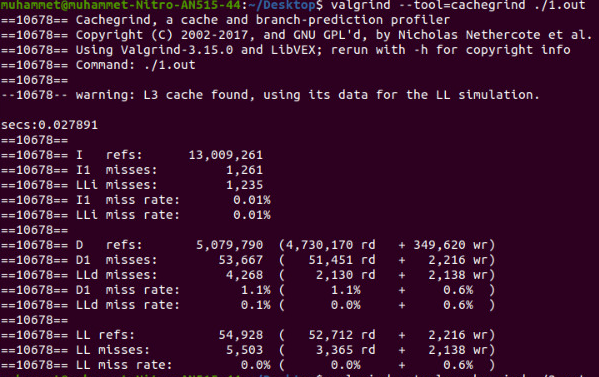


İşlemcimin cache içeriğinin ekran görüntüsü.

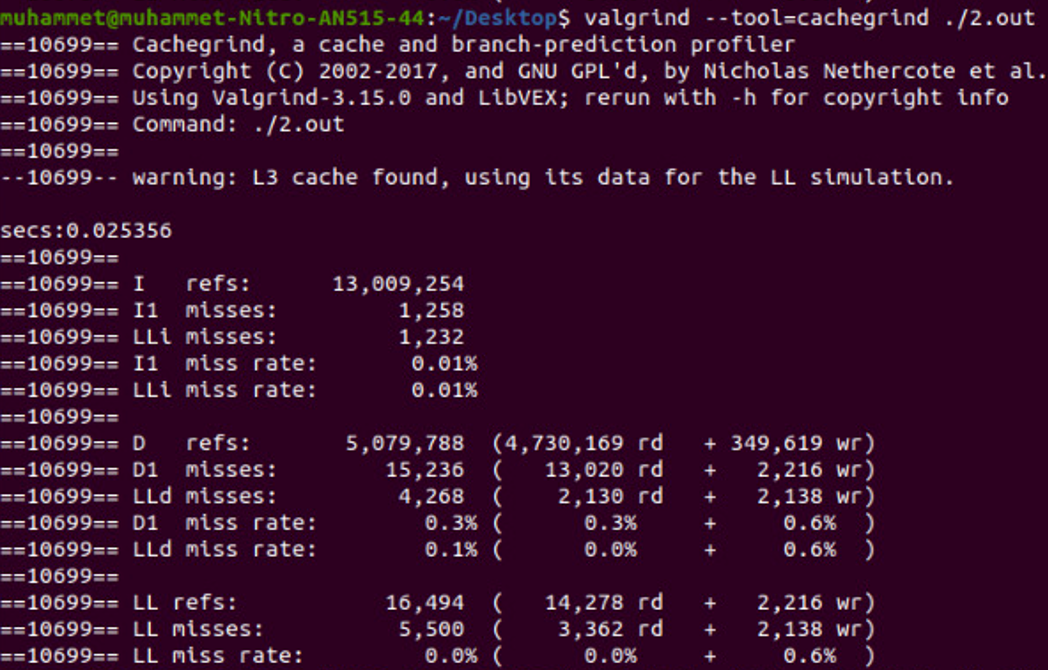


Kodların çalıştırılması sonrasında aldığım ekran görüntüleri

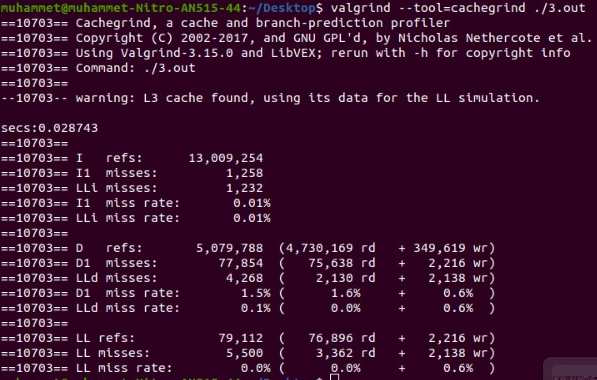
1.c dimension = 64



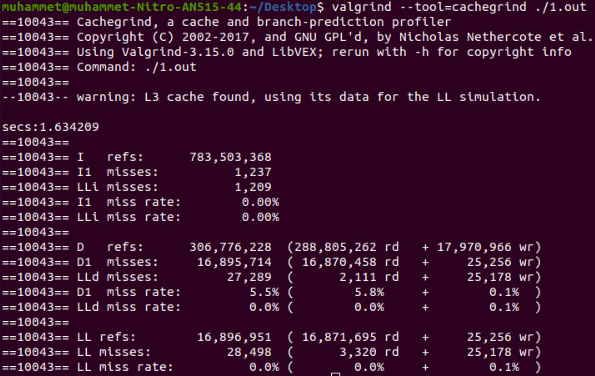
2.c dimension = 64



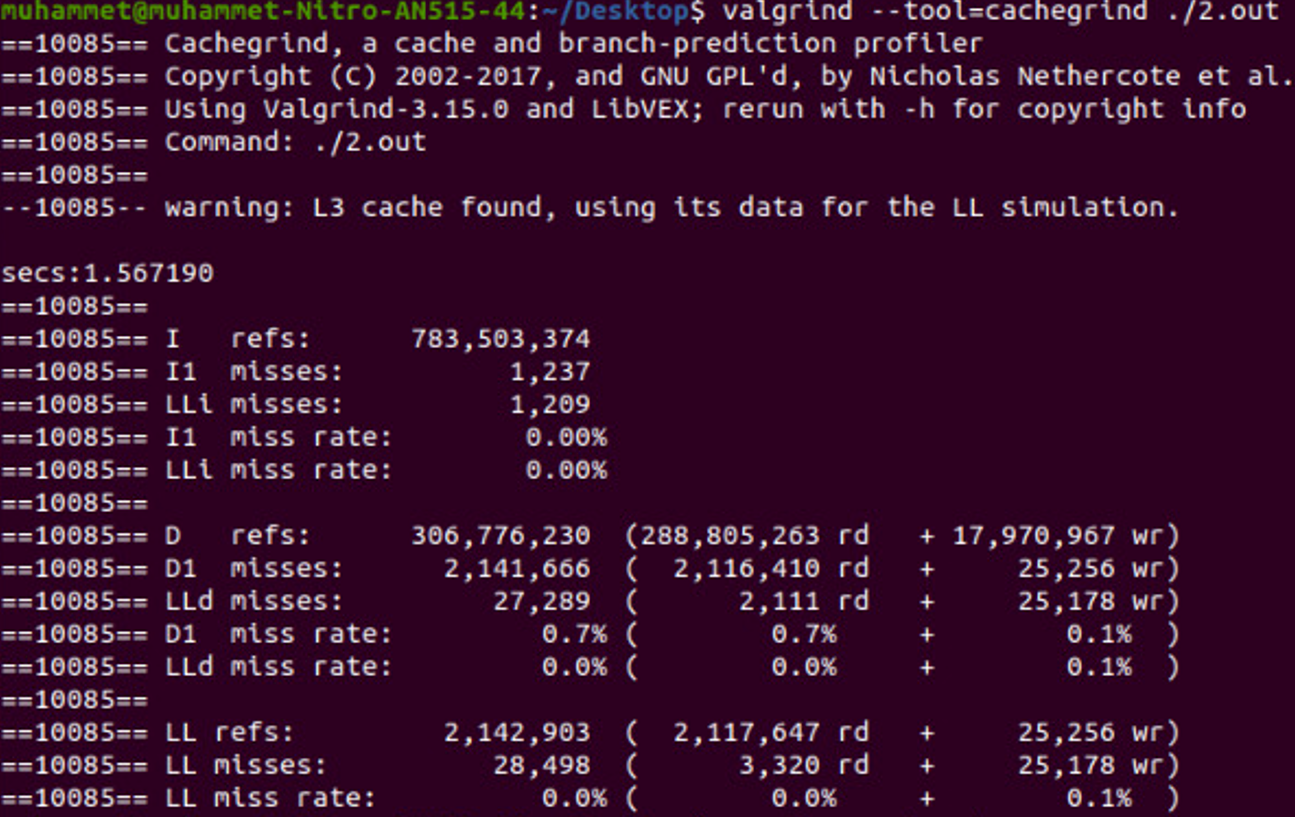
3.c dimension=64



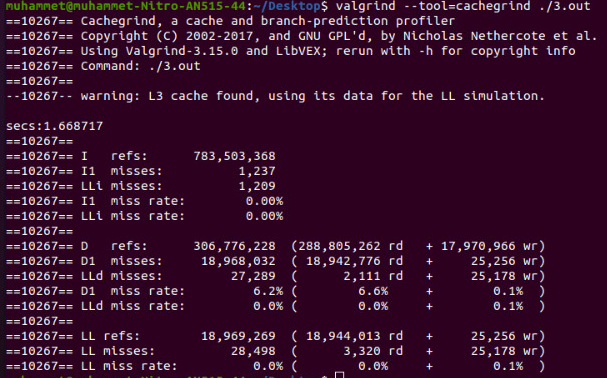
1.c dimension = 256

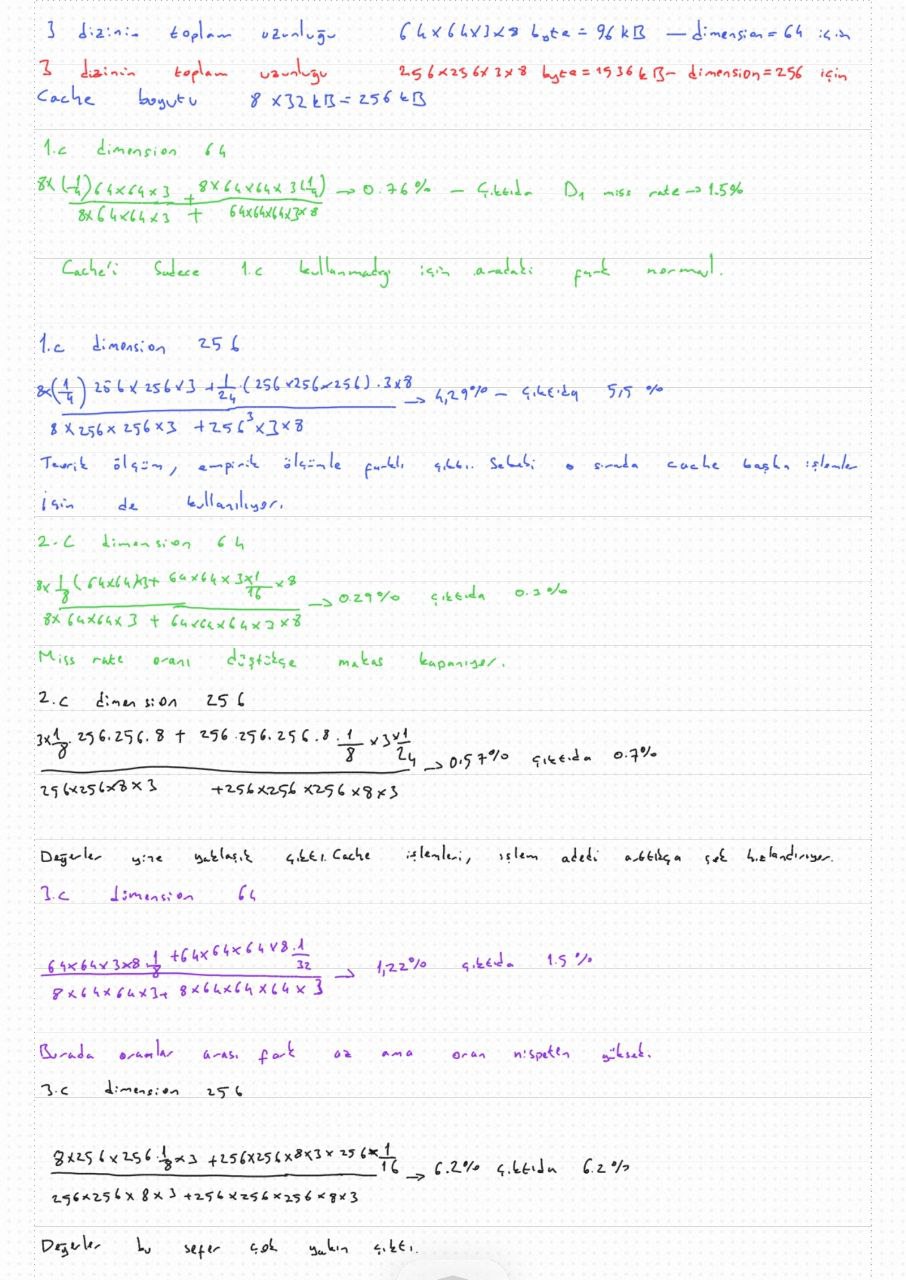


2.c dimension = 256



3.c dimension = 256





**Komut açısındansa**, MOV CX,DIMENSION – PUSH CX – POP CX – INC BX – INC SI – INC DI – XOR DI,DI – XOR BX-BX – XOR SI,SI – LOOP [memory] komutları kullanılıyor. Yaklaşık 1250 farklı komut.

İş böyle olunca dimension arttıkça verilen komut sayısı artarken, komut çeşidi değişmiyor. Bundan miss rate de veri arttıkça azalıyor. Komutlar aynı olduğu sürece, diğer etkenler bu durumu çok etkilemiyor. Bundan dolayı komut açısından boyutu aynı olduğu sürece kodlar aynı komutları verdiği için normal olarak hiçbir fark gözlemlemedik.

1.c dimension 256 için

1250/(2\*(256\*256\*3\*8+256\*256\*256\*3\*8)) -> 0.00015% ~ 0% ----< çıktıda 0%

2.c dimension 256 için

1250/(2\*(256\*256\*3\*8+256\*256\*256\*3\*8))-> 0.00015% ~ 0% ----< çıktıda 0%

2.c dimension 256 için

1250/(2\*(256\*256\*3\*8+256\*256\*256\*3\*8)) -> 0.00015% ~ 0% ----< çıktıda 0%

1.c dimension 64 için

1250/(2\*(64\*64\*3\*8+64\*64\*64\*3\*8)) -> 0.0097% ~ 1% ----< çıktıda 1%

2.c dimension 64 için

1250/(2\*(64\*64\*3\*8+64\*64\*64\*3\*8)) -> 0.0097% ~ 1% ----< çıktıda 1%

3.c dimension 64 için

1250/(2\*(64\*64\*3\*8+64\*64\*64\*3\*8)) -> 0.0097% ~ 1% ----< çıktıda 1%

**Süreler ile cache hit-miss oranı kıyaslama**

**Dimension = 256**

|  |  |  |
| --- | --- | --- |
| Kod | Miss rate | secs |
| 1.c | 5.5% | 1.63 |
| 2.c | 0.7% | 1.57 |
| 3.c | 6.2% | 1.66 |

Eleman sayısının fazla olduğu durumda miss-rate’ler mili saniye bazında büyük oranda etkili. Burada 1.c ile 2.c’ nin miss rate lerini kıyaslarsak 4.8% fark oluştuğunu görürüz ve bu da yaklaşık 60 ms bir fark oluşturmuş. 1.c ile 3.c arasındaysa bu fark yalnızca yaklaşık 30 ms. Aşağıda da görüleceği üzere miss rate oranları yüksek verilerde çalışırken çok daha önemli bir hal alıyor. Ya da rekabetçi oyunlar, borsa vb. işlerde yine cache yapısının önemi kat kat artıyor.

**Dimension = 64**

|  |  |  |
| --- | --- | --- |
| Kod | Miss rate | Secs |
| 1.c | 1.1% | 0.028 |
| 2.c | 0.3% | 0.025 |
| 3.c | 1.5% | 0.029 |

Eleman sayısının daha az olduğu durumdaysa hem miss rateler daha düşük seviyelerde hem de etkileri de bu nispette düşük. Burada 2.c ile 3.c’yi kıyasladığımızda miss rateleri arasındaki fark 1.2% seviyesindeyken mili saniye bazında yalnızca yaklaşık 40 mili saniye. 1.c ile 3.c’ye baktığımızdaysa miss rateleri atasındaki fark yalnızca 0.4% ve milisaniye bazında aradaki fark neredeyse 10 mili saniye.

**Dimension Kıyas**

Dimensionlarına göre kıyasladığımızda 1.c için 64 ve 256 arasında yaklaşık 4 katlık bir miss rate farkı var ve aynı zamanda saniye bazında bakarsak 57 katlık bir fark var, 3.c içinse miss rate oranın farkı yaklaşık 3 kat ve saniye farkıysa yaklaşık 56 kat. Veri boyutu büyüdükçe doğal olarak veri işlemleri uzuyor ama aynı zamanda miss rate oranları da cache boyutunun sınırlı olmasından dolayı artıyor.