Bilgisayar Donanımı 2. Ödevi

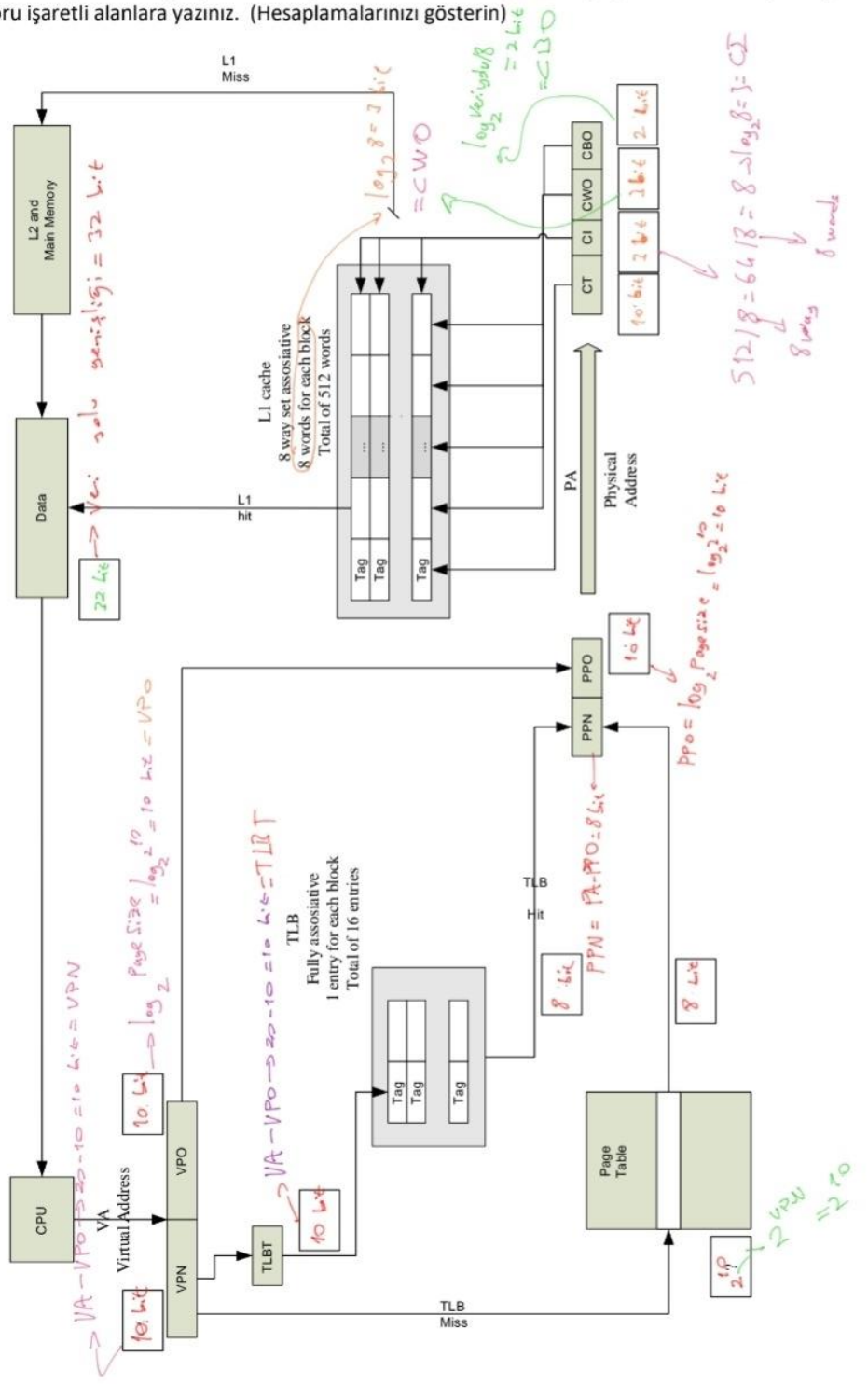
Muhammet Kayra Bulut

20011901

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

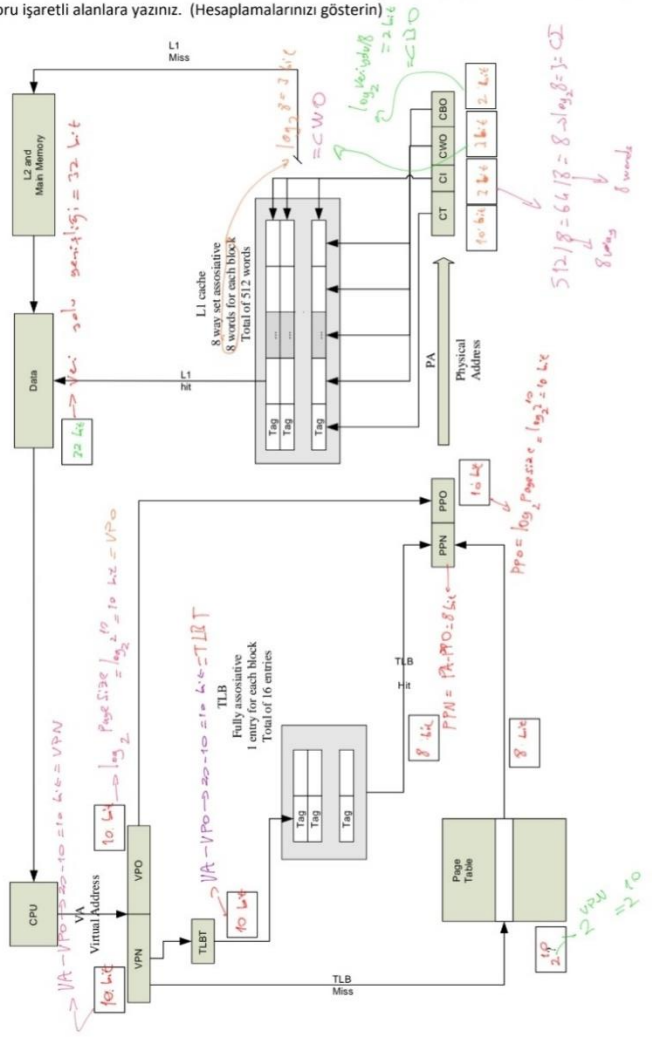
Soru 1)

Şekil ile verilen virtual memory, TLB, page table, cache yapısı için Page size 1KB, Veri yolu genişliği 32 bit, Virtual address genişliği 20 bit, Physical address genişliği 18 bit olarak veriliyor. Buna göre sistem adres dönüşümünde oluşan adres parçalarının kaç bit uzunlukta olduğunu ve page table satır sayısını şekil üzerinde soru işaretli alanlara yazınız. (Hesaplamalarınızı gösterin)



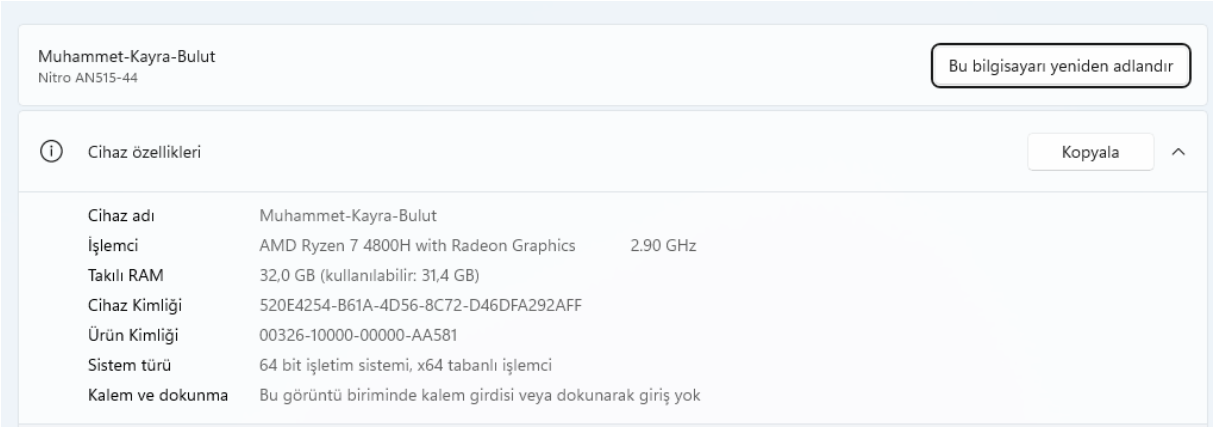
Soru 1)

Şekil ile verilen virtual memory, TLB, page table, cache yapısı için Page size 1KB, Veri yolu genişliği 32 bit, Virtual address genişliği 20 bit, Physical address genişliği 18 bit olarak veriliyor. Buna göre sistem adres dönüşümünde oluşan adres parçalarının kaç bit uzunlukta olduğunu ve page table satır sayısını şekil üzerinde soru işaretli alanlara yazınız. (Hesaplamalarınızı gösterin)



Soru 2 -)

Laptop işlemcimin modeli;



İşlemcimin cache içeriğinin ekran görüntüsü.

Level 1 cache size	8 x 32 KB 8-way set associative instruction caches 8 x 32 KB 8-way set associative data caches
Level 2 cache size	**8 x 512 KB 8-way set associative unified caches**
Level 3 cache size	**2 x 4 MB 16-way set associative shared caches**

Kodların çalıştırılması sonrasında aldığım ekran görüntüleri

1.c dimension = 64

```
muhammet@muhammet-Nitro-AN515-44:~/Desktop$ valgrind --tool=cachegrind ./1.out
==10678== Cachegrind, a cache and branch-prediction profiler
==10678== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10678== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10678== Command: ./1.out
==10678==
--10678-- warning: L3 cache found, using its data for the LL simulation.

secs:0.027891
==10678==
==10678== I   refs:      13,009,261
==10678== I1  misses:         1,261
==10678== LLi misses:         1,235
==10678== I1  miss rate:       0.01%
==10678== LLi miss rate:       0.01%
==10678==
==10678== D   refs:       5,079,790  (4,730,170 rd   + 349,620 wr)
==10678== D1  misses:        53,667  (   51,451 rd   +   2,216 wr)
==10678== LLd misses:         4,268  (    2,130 rd   +   2,138 wr)
==10678== D1  miss rate:        1.1% (      1.1%     +     0.6%  )
==10678== LLd miss rate:        0.1% (      0.0%     +     0.6%  )
==10678==
==10678== LL refs:           54,928  (   52,712 rd   +   2,216 wr)
==10678== LL misses:          5,503  (    3,365 rd   +   2,138 wr)
==10678== LL miss rate:         0.0% (      0.0%     +     0.6%  )
```

2.c dimension = 64

```
muhammet@muhammet-Nitro-AN515-44:~/Desktop$ valgrind --tool=cachegrind ./2.out
==10699== Cachegrind, a cache and branch-prediction profiler
==10699== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10699== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10699== Command: ./2.out
==10699==
--10699-- warning: L3 cache found, using its data for the LL simulation.

secs:0.025356
==10699==
==10699== I   refs:      13,009,254
==10699== I1  misses:         1,258
==10699== LLi misses:         1,232
==10699== I1  miss rate:       0.01%
==10699== LLi miss rate:       0.01%
==10699==
==10699== D   refs:       5,079,788  (4,730,169 rd   + 349,619 wr)
==10699== D1  misses:        15,236  (    13,020 rd   +   2,216 wr)
==10699== LLd misses:         4,268  (     2,130 rd   +   2,138 wr)
==10699== D1  miss rate:        0.3% (       0.3%     +     0.6%   )
==10699== LLd miss rate:        0.1% (       0.0%     +     0.6%   )
==10699==
==10699== LL refs:           16,494  (    14,278 rd   +   2,216 wr)
==10699== LL misses:          5,500  (     3,362 rd   +   2,138 wr)
==10699== LL miss rate:         0.0% (       0.0%     +     0.6%   )
```

3.c dimension=64

```
muhammet@muhammet-Nitro-AN515-44:~/Desktop$ valgrind --tool=cachegrind ./3.out
==10703== Cachegrind, a cache and branch-prediction profiler
==10703== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10703== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10703== Command: ./3.out
==10703==
--10703-- warning: L3 cache found, using its data for the LL simulation.

secs:0.028743
==10703==
==10703== I   refs:      13,009,254
==10703== I1  misses:         1,258
==10703== LLi misses:         1,232
==10703== I1  miss rate:       0.01%
==10703== LLi miss rate:       0.01%
==10703==
==10703== D   refs:       5,079,788  (4,730,169 rd   + 349,619 wr)
==10703== D1  misses:        77,854  (    75,638 rd   +   2,216 wr)
==10703== LLd misses:         4,268  (     2,130 rd   +   2,138 wr)
==10703== D1  miss rate:        1.5% (       1.6%     +     0.6%   )
==10703== LLd miss rate:        0.1% (       0.0%     +     0.6%   )
==10703==
==10703== LL refs:           79,112  (    76,896 rd   +   2,216 wr)
==10703== LL misses:          5,500  (     3,362 rd   +   2,138 wr)
==10703== LL miss rate:         0.0% (       0.0%     +     0.6%   )
```

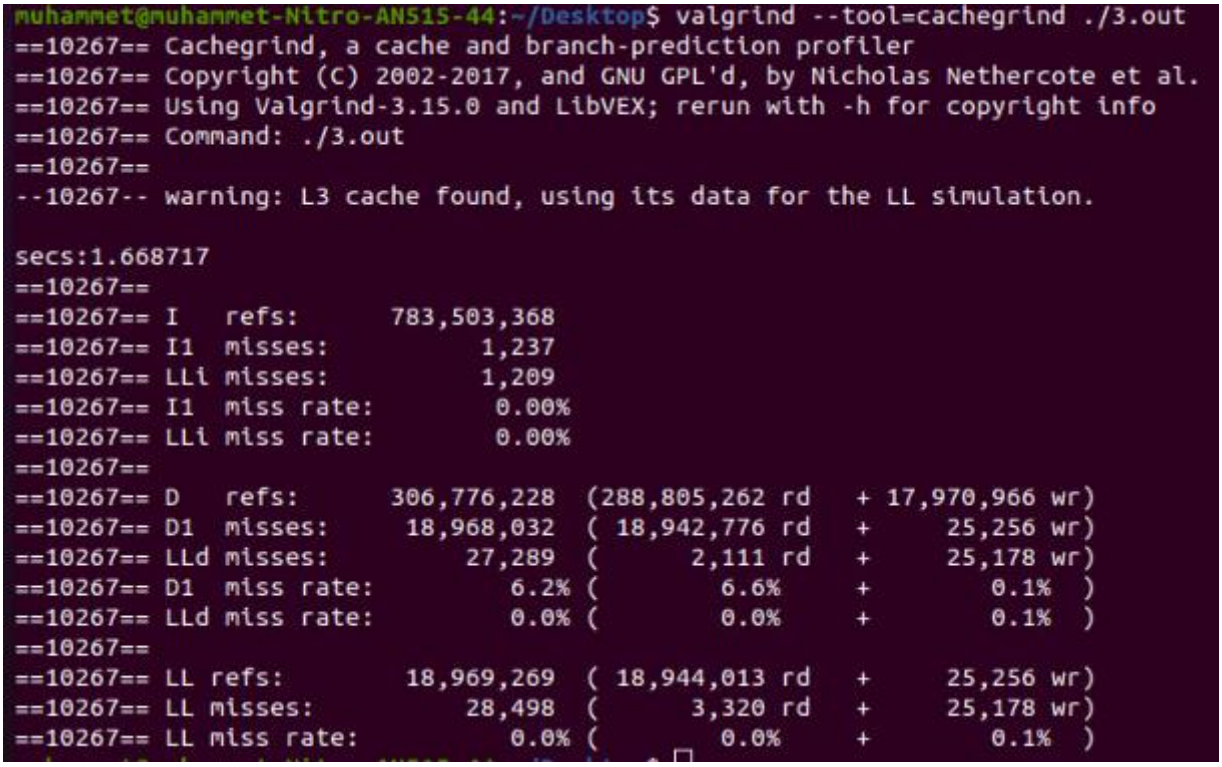
1.c dimension = 256

```
muhammet@muhammet-Nitro-AN515-44:~/Desktop$ valgrind --tool=cachegrind ./1.out
==10043== Cachegrind, a cache and branch-prediction profiler
==10043== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10043== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10043== Command: ./1.out
==10043==
--10043-- warning: L3 cache found, using its data for the LL simulation.

secs:1.634209
==10043==
==10043== I   refs:      783,503,368
==10043== I1  misses:          1,237
==10043== LLi misses:          1,209
==10043== I1  miss rate:        0.00%
==10043== LLi miss rate:        0.00%
==10043==
==10043== D   refs:      306,776,228  (288,805,262 rd   + 17,970,966 wr)
==10043== D1  misses:     16,895,714  ( 16,870,458 rd   +     25,256 wr)
==10043== LLd misses:         27,289  (      2,111 rd   +     25,178 wr)
==10043== D1  miss rate:         5.5% (        5.8%     +        0.1%  )
==10043== LLd miss rate:         0.0% (        0.0%     +        0.1%  )
==10043==
==10043== LL refs:        16,896,951  ( 16,871,695 rd   +     25,256 wr)
==10043== LL misses:          28,498  (      3,320 rd   +     25,178 wr)
==10043== LL miss rate:          0.0% (        0.0%     +        0.1%  )
```

2.c dimension = 256

```
muhammet@muhammet-Nitro-AN515-44:~/Desktop$ valgrind --tool=cachegrind ./2.out
==10085== Cachegrind, a cache and branch-prediction profiler
==10085== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==10085== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==10085== Command: ./2.out
==10085==
--10085-- warning: L3 cache found, using its data for the LL simulation.

secs:1.567190
==10085==
==10085== I   refs:      783,503,374
==10085== I1  misses:          1,237
==10085== LLi misses:          1,209
==10085== I1  miss rate:        0.00%
==10085== LLi miss rate:        0.00%
==10085==
==10085== D   refs:      306,776,230  (288,805,263 rd   + 17,970,967 wr)
==10085== D1  misses:      2,141,666  (  2,116,410 rd   +     25,256 wr)
==10085== LLd misses:         27,289  (      2,111 rd   +     25,178 wr)
==10085== D1  miss rate:         0.7% (        0.7%     +        0.1%  )
==10085== LLd miss rate:         0.0% (        0.0%     +        0.1%  )
==10085==
==10085== LL refs:         2,142,903  (  2,117,647 rd   +     25,256 wr)
==10085== LL misses:          28,498  (      3,320 rd   +     25,178 wr)
==10085== LL miss rate:          0.0% (        0.0%     +        0.1%  )
```

3.c dimension = 256



3 dizinin toplam uzunluğu $64 \times 64 \times 3 \times 8$ byte = 96 kB — dimension = 64 için

3 dizinin toplam uzunluğu $256 \times 256 \times 3 \times 8$ byte = 1536 kB — dimension = 256 için

Cache boyutu $8 \times 32\,kB = 256\,kB$

1.c dimension 64

$$\frac{8 \times \left(\frac{1}{4}\right) 64 \times 64 \times 3 + 8 \times 64 \times 64 \times 3 \left(\frac{1}{8}\right)}{8 \times 64 \times 64 \times 3 + 64 \times 64 \times 64 \times 3 \times 8} \rightarrow 0.76\% \text{ — çıkıda } D_1 \text{ miss rate} \rightarrow 1.5\%$$

Cache'i sadece 1.c kullanmadığı için aradaki fark normal.

1.c dimension 256

$$\frac{8 \times \left(\frac{1}{4}\right) 256 \times 256 \times 3 + \frac{1}{24} \cdot (256 \times 256 \times 256) \cdot 3 \times 8}{8 \times 256 \times 256 \times 3 + 256^3 \times 3 \times 8} \rightarrow 4{,}29\% \text{ — çıkışta } 5{,}5\ \%$$

Teorik ölçüm, empirik ölçümle farklı çıktı. Sebebi o sırada cache başka işlemler için de kullanılıyor.

2.C dimension 64

$$\frac{8 \times \frac{1}{8}(64 \times 64 \times 3 + 64 \times 64 \times 3 \times \frac{1}{16} \times 8}{8 \times 64 \times 64 \times 3 + 64 \times 64 \times 64 \times 3 \times 8} \rightarrow 0.29\% \text{ çıkışta } 0.3\ \%$$

Miss rate oranı düştükçe makas kapanıyor.

2.c dimension 256

$$\frac{3 \times \frac{1}{8} \cdot 256 \cdot 256 \cdot 8 + 256 \cdot 256 \cdot 256 \cdot 8 \cdot \frac{1}{8} \times 3 \times \frac{1}{24}}{256 \times 256 \times 8 \times 3 \quad + 256 \times 256 \times 256 \times 8 \times 3} \rightarrow 0.57\% \text{ çıkışta } 0.7\%$$

Değerler yine yaklaşık çıktı. Cache işlemleri, işlem adedi arttıkça çok hızlandırıyor.

3.c dimension 64

$$\frac{64 \times 64 \times 3 \times 8 \cdot \frac{1}{8} + 64 \times 64 \times 64 \times 8 \cdot \frac{1}{32}}{8 \times 64 \times 64 \times 3 + 8 \times 64 \times 64 \times 64 \times 3} \rightarrow 1{,}22\% \text{ çıkışta } 1.5\ \%$$

Burada oranlar arası fark az ama oran nispeten yüksek.

3.c dimension 256

$$\frac{8 \times 256 \times 256 \cdot \frac{1}{8} \times 3 + 256 \times 256 \times 8 \times 3 \times 256 \times \frac{1}{16}}{256 \times 256 \times 8 \times 3 + 256 \times 256 \times 256 \times 8 \times 3} \rightarrow 6.2\% \text{ çıkışta } 6.2\ \%$$

Değerler bu sefer çok yakın çıktı.

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.