Solution 4

1.

```
for (p = 0; p < 2; p++) {
  for (q = 0; q < 2; q++) {
    for (i = p*64; i < (p+1)*64; i++) {
      for (j = q*64; j < (q+1)*64; j++) {
        Y[i] = Y[i] + A[i][j]*X[j];
      }
    }
  }
}
```

When **p = 0** we compute **Y[0:63]** in 2 steps: first, we use **A[0:63][0:63]** and **X[0:63]** when **q = 0**; then, we use **A[0:63][64:127]** and **X[64:127]** when **q = 1**. When **p = 1** we compute **Y[64:127]** in 2 steps: first, we use **A[64:127][0:63]** and **X[0:63]** when **q = 0**; then, we use **A[64:127][64:127]** and **X[64:127]** when **q = 1**.

Storing one **64x64** block of 32-bit numbers (for matrix **A**) requires 64*64*4=**16KB** of memory, and storing two **128x1** blocks of 32-bit numbers (for vectors **X** and **Y**) requires 2*128*4=**1KB** of memory. Hence, the cache size should be at least **17KB**.

2. The total size of **int X[256][256]** is 256*256*4=**256KB**, and each row requires 256*4=**1KB**. For every iteration of the outer loop (index **i**), we have 2 reads and 1 write per row element **X[i][j]**, i.e., each row **i** requires (2+1)*256=768 accesses to it.

If we have four **1KB**-pages, we get 4 page faults per 4 rows, or 4 page fault per 4*768 accesses (the size of each row is **1KB**, and there are 768 accesses to each row); therefore, the page fault rate is 4/(4*768) = 0.13%. On the other hand, if we have one **4KB**-page, we get 1 page fault per 4*768 accesses; therefore, the page fault rate is 1/(4*768) = 0.03%. In both cases the allocated memory amount is the same (**4KB** total), but the page fault rates are different.

3. If we allocate 512 **4KB**-pages for the **int a[1024][1024]** array, then the **Good** example will have 1 page fault per 1024 row element accesses (**p ≈ 0.1%**), and the **Bad** example will have 1 page fault per 1 row element access (**p = 100%**), i.e., increasing the number of allocated **4KB**-pages from 1 to 512 does not provide any benefits for this particular application code. If we allocate one **4KB**-page for the **int a[512][512]** array (i.e., one page holds 2 rows), then the **Good** example will have 1 page fault per 512+512 row element accesses (**p ≈ 0.1%**), and the **Bad** example will have 1 page fault per 1+1 row element accesses (**p = 50%**).

4. General formula: $\mathbf{T_{ave} = h_1C_1 + (1-h_1)(h_2C_2 + (1-h_2)((1-p)M + pD))}$.
(a) The page table must have a dedicated entry for every possible VPN. As we have at most **4GB/1MB = 4K** pages (i.e., **4K** VPNs), the page table has **4K** entries.
(b) For $\mathbf{h_1 = 0.95}$, $\mathbf{h_2 = 0.90}$, and $\mathbf{p = 0}$, we have:

$$\mathbf{T_{ave} = 0.95 \cdot 1\tau + 0.05(0.9 \cdot 4\tau + 0.1(1 \cdot 16\tau + 0 \cdot 10{,}000\tau)) = 1.21\tau}.$$

(c) For $\mathbf{h_1 = 0}$, $\mathbf{h_2 = 0.90}$, and $\mathbf{p = 0.001}$, we have:

$$\mathbf{T_{ave} = 0 \cdot 1\tau + 1(0.9 \cdot 4\tau + 0.1(0.9999 \cdot 16\tau + 0.0001 \cdot 10{,}000\tau)) = 5.3\tau}.$$

(d) For $\mathbf{h_1 = 0.95}$, $\mathbf{h_2 = 0.90}$, and $\mathbf{p = 0.001}$, we have:

$$\mathbf{T_{ave} = 0.95 \cdot 1\tau + 0.05(0.9 \cdot 4\tau + 0.1(0.9999 \cdot 16\tau + 0.0001 \cdot 10{,}000\tau)) = 1.215\tau}.$$