ASL HW4
Student Name: Berke Egeli
Legi. No.: 20 948 220

Exercises:

1) Associativity (20 pts)

Consider the following function, executed on a machine with a write-back/write-allocate cache with blocks of size 16 bytes, a total capacity of 128 bytes and with a LRU replacement policy. Arrays x, y and z are cache-aligned (first element goes into first cache block). Assume that memory accesses occur in exactly the order that they appear in the code. Thus, no optimizations are performed that reduce the memory accesses or reorder computations. The variables i and t remain in registers and do not cause cache misses.

(a) Considering cache misses from both reads and writes, compute the following two things: i) the miss/hit pattern for x, y and z (something like x: MMHHM..., y:MMMH...); ii) the operational intensity (in flops/byte) of the above computation for the following cases. For the operational intensity assume empty caches and consider only reads but note: arrays that are only written are also read and this read should be included. Show your work.

i. Miss/hit pattern and operational intensity when the cache is 2-way set associative

Solution: In this exercise and the next one, I will use the following notation: X.Y indicates the location an array element is put in cache where X denotes the set and Y denotes the additional group of sets that come from associativity. For example, 0.1 means 1st set and 2nd group of sets etc. For every iteration, I will first write the elements accessed in the 1st column, then their location in cache and whether access is a hit or miss.

i = 1
- x[4] 2.0 M
- y[1] 0.0 M
- z[3] 1.0 M

i = 2
- x[8] 0.1 M
- y[4] 2.1 M
- z[7] 3.0 M

i = 3
- x[0] 0.0 M
- y[7] 3.1 M
- z[3] 1.0 H

i = 4
- x[4] 2.0 H
- y[2] 1.1 M
- z[7] 3.0 H

i = 5
- x[8] 0.1 H
- y[5] 2.1 H
- z[3] 1.0 H

i = 6
    x[0] 0.0 H
    y[0] 0.1 M
    z[7] 3.0 H
i = 7
    x[4] 2.0 H
    y[3] 1.1 H
    z[3] 1.0 H
i = 8
    x[8] 0.0 M
    y[6] 3.1 H
    z[7] 3.0 H
i = 9
    x[0] 0.1 M
    y[1] 0.0 M
    z[3] 1.0 H
i = 10
    x[4] 2.0 H
    y[4] 2.1 H
    z[7] 3.0 H

Cache access patterns for the arrays:
    x: MMMHHHHMMH (5H 5M)
    y: MMMMHMHHMH (4H 6M)
    z: MMHHHHHHHH (8H 2M)

There are 20 flops in total in the entire computation (2 summations for 10 iterations). For every cache miss we will bring 16 byte blocks from the memory. So,

$$\text{I(n) for n = 20 is } \frac{20}{13\times16}\, flops/byte \approx 0.096\, flops/byte$$

ii. Miss/hit pattern and operational intensity when the cache is 4-way set associative.

Solution:

i = 1
    x[4] 0.0 M
    y[1] 0.1 M
    z[3] 1.0 M
i = 2
    x[8] 0.2 M
    y[4] 0.3 M
    z[7] 1.1 M

i = 3
- x[0] 0.0 M
- y[7] 1.2 M
- z[3] 1.0 H

i = 4
- x[4] 0.1 M
- y[2] 1.3 M
- z[7] 1.1 H

i = 5
- x[8] 0.2 H
- y[5] 0.3 H
- z[3] 1.0 H

i = 6
- x[0] 0.0 H
- y[0] 0.1 M
- z[7] 1.1 H

i = 7
- x[4] 0.2 M
- y[3] 1.3 H
- z[3] 1.0 H

i = 8
- x[8] 0.3 M
- y[6] 1.2 H
- z[7] 1.1 H

i = 9
- x[0] 0.0 H
- y[1] 0.1 H
- z[3] 1.0 H

i = 10
- x[4] 0.2 H
- y[4] 0.3 M
- z[7] 1.1 H

Cache access patterns for the arrays:
- x: MMMMHHMMHH (4H 6M)
- y: MMMMHMHHHM (4H 6M)
- z: MMHHHHHHHH (8H 2M)

There are 20 flops in total in the entire computation (2 summations for 10 iterations). For every cache miss we will bring 16 byte blocks from the memory. So,

I(n) for n = 20 is $\frac{20}{14\times16} flops/byte \approx 0.089\ flops/byte$

(b) Assuming a 2-way set associative cache (motivate your answers):

i. What kind(s) of locality do the accesses to array x have?

Solution:

Array x only has temporal locality, because we repeatedly access x[4], x[8], x[0] after they are brought into the cache. Also, they are the only elements we access from x and none of them is brought into the same cache line at the same time.

ii. What kind(s) of locality do the accesses to array y have?

Solution:

Array y has both spatial and temporal locality. We access y[5] in iteration 5 after it is brought into cache with y[4] access in iteration 2. Similarly, we access y[3] in iteration 7 and y[6] in iteration 8 after they are brought with accesses to y[2] in iteration 4 and y[7] in iteration 3 respectively. Array y also has temporal locality because we access y[4] in iteration 10 after it was brought into cache in iteration 2 with access to y[4].

2. Cache Mechanics (35 pts)

Consider the following code, executed on a machine with a write-back/write- allocate direct-mapped cache with blocks of size 32 bytes, a total capacity of 12KiB and with a LRU replacement policy. Assume that memory accesses occur in exactly the order that they appear. The variables i,j,k,t and n remain in registers and do not cause cache misses. Arrays A and B are cache-aligned (first element goes into first cache block). For this and the following exercises, assume a cold cache scenario. sizeof(double) = sizeof(uint64 t) = 8.

Considering cache misses from both reads and writes, compute i) the cache miss rate and ii) the operational intensity (in flops/byte) of the above computation for the following cases. For the operational intensity assume only reads, i.e., data movement from main memory to cache. Show enough details so we can see your reasoning.

(a) Miss rate and operational intensity for n = 8.

Solution:

For n = 8, the cache can store 192 consecutive struct objects, so, technically all of the arrays A & B can fit in cache. Moreover, we always access the same 8 elements in the B array, so, for most of the execution, elements of B will be in cache, however, there are two distinct cases. Namely, iteration 0 is different from iterations 2 to n – 1 because of the compulsory misses caused by accesses to B.

i = 1

A[0, 7].v **All of these accesses will cause a miss**

A[0, 7].d[0, 2] **All of these accesses will be a hit because they are brought in with v**

A[0, 7].u[0, 2] **The first access will be a miss, the next two will be hits because they are consecutive**

B[0, 8, 16, 24, 32, 40, 48, 56].v **All of them will miss because of compulsory misses.**

B[0, 8, 16, 24, 32, 40, 48, 56].v **All of them will hit because of previous accesses.**

i = 2

A[8, 15].v **All of these accesses will cause a miss**

A[8, 15].d[0, 2] **All of these accesses will be a hit because they are brought in with v**

A[8, 15].u[0, 2] **The first access will be a miss, the next two will be hits because they are consecutive**

(*) B[0, 8, 16, 24, 32, 40, 48, 56].v **Only one of them (8) will miss because it shares the same cache position with the A access in the first line**

B[0, 8, 16, 24, 32, 40, 48, 56].v **All of them will hit because of previous accesses.**

(*) From i = 2 onwards we have the same general case and access pattern. One of the B accesses will be a miss because of the conflict miss caused by A access in the first line.

So, in total we will have $\mathbf{n^2 + n^2 + n + n - 1}$ misses & $\mathbf{9n^2}$ accesses.

$$Miss\ Rate\ = \frac{143}{576} \approx 0.25$$

For operational intensity we have $4n^2$ flops and we have 143 misses, so, we will read 143 * 32 bytes from memory to cache since we are ignoring writing values back.

$$I(n) = \frac{4 \times 8^2}{143 \times\ 32} \approx 0.056\ flops/byte$$

(b) Miss rate and operational intensity for n = 16.

Solution:

In this case, we have more array elements than what could fit in the cache. That is why we will get more conflict misses in this scenario. The elements of array A are accessed only once, so, their access pattern is similar to the previous case. We repeatedly access the same 16 elements of B, so we will get conflict misses. There are three cases:

For i = 0,

We will have compulsory misses so all of the read accesses of the B will miss but since first four and last four elements accessed are mapped to the same cache positions, they will cause conflict misses in later iterations. Read misses have 2 more alternatives because they have conflict misses with the elements of A as well.

We will have 3n misses when i = 0

For i=1:3 and i = 12:15

In this case the conflict misses between the accesses of B and A overlap, so, for the reads of B we will have 8 misses.

We will have 2n + 8 misses for each i=1:3 and i = 12:15

For i = 4:11

The conflict misses of B and A will not overlap, so, reads of B will have 9 misses.

We will have 2n + 9 misses for each i = 4:11

So, in total, we will have 33n + 128 misses and we will have $9n^2$ accesses.

$$Miss\ Rate\ = \frac{656}{2304} \approx 0.28$$

For operational intensity we have $4n^2$ flops and we have 784 misses, so, we will read 784 * 32 bytes from memory to cache since we are ignoring writing values back.

$$I(n) = \frac{4 \times 16^2}{656 \times\ 32} \approx 0.049\ flops/byte$$

(c) Miss rate and operational intensity for n = 16 and PADDING SIZE = 5.

Solution:

In this question, last four elements of the pad array will be mapped to every third cache line in the cache and since we don't use that array nor any other element in the struct arrays are mapped to those lines, we are effectively reducing the size of the array to 2/3 of its original size. As a result, the conflict misses increase even more. Again, every A element is accessed only once, so, the access pattern for it is still the same. This time however, the repeated accesses to B array will cause conflict misses, so, we will not have any hits for reads of B. This is because the first half of the elements accessed from B array are mapped to the same cache lines as the last half of the elements. When we access an element, it will not be in cache, replace the other, than it won't be used again until it is replaced by the previous value.

So, in total, we have $3n^2$ cache misses and we still have $9n^2$ accesses.

$$Miss\ Rate\ = \frac{3}{9} \approx 0.33$$

For operational intensity we have $4n^2$ flops and we have $3n^2$ misses, so, we will read $3n^2$ * 32 bytes from memory to cache since we are ignoring writing values back.

$$I(n) = \frac{1}{24} \approx 0.041\ flops/byte$$

3. Rooflines (40 pt) Consider a processor with the following hardware parameters (assume 1GB = 10^9B):

- SIMD vector length of 256 bits.
- Two instruction ports that execute floating point operations:
  – Port 0 (P0): FMA, ADD, MUL
  – Port 1 (P1): FMA, ADD, MUL
  Each port can issue 1 operation per cycle. Each operation has a latency of 1.
- One write-back/write-allocate cache.
- Read bandwidth from the main memory is 50 GB/s.
- Processor frequency is 2 GHz.

(a) Draw a roofline plot for the machine. Consider only double-precision floating point arithmetic. Consider only reads. Include a roofline for when vector instructions are not used and for when vector instructions are used.

Solution:

Peak performance is achieved when we perform 2 FMA operations. So, the peak performance is 4 flops/cycle and 16 flops/cycle for scalar and vectorized code respectively. The rooflines are $P \leq \pi_s$, $P \leq \pi_v$ and $P \leq I \cdot \beta$, where $\pi_s = 4$, $\pi_v = 16$ and $\beta = 25$ bytes/cycle. For scalar execution, compute bound will intersect memory bound at I = 0.16 flops/byte and for vectorized they will intersect at I = 0.64 flops/byte.

(b) Compute a hard upper bound on the operational intensity I of the functions below based on compulsory misses. Based on this I alone, i.e. ignoring instruction mix, add the maximum per- formance of each function to the roofline plot assuming first that vector instructions are not used (three dots). Then, assume that vector instructions are used to speedup the computations and add their new maximum performance (three additional dots). At the end, there should be six dots in the roofline. Consider only reads, cold-cache scenario, only compulsory misses. Ignore the effects of aliasing and assume that no optimizations that change operational intensity are performed (the computation stays as is).

Solution:

Unvectorized:
Comp1: W(n) = 3n, Q(n) ≥ 8(2n) = 16n, $\frac{3n}{16n} = \frac{3}{16} = 0.1875 \geq I(n)$
0.1875 is greater than 0.16, so, the function is compute bound and thus the maximum performance it can achieve is 4 flops/cycle.

Comp2: Comp2 has the same operational intensity upper bound and performance as comp1 due to their similar structure and the fact that we are ignoring the instruction mix.

Comp3: Since r = 3, W(n) = 6n, Q(n) ≥ 8(5n*) = 40n, $\frac{6n}{40n} = \frac{3}{20} \geq$ I(n)
0.15 is smaller than the 0.16 (duh), so, this function is memory bound. The maximum performance it can achieve is 3.75 flops/cycle.
*Note: 5 comes from rn + 2n where r = 3, so, 3n + 2n = 5n.

Vectorized:
Since operational intensity doesn't change with vectorization and all computations have operational intensity less than 0.64, they will all hit the memory bound. Comp1 & Comp2 will reach a performance of $\frac{25\,x\,3}{16} =$ 4.6875 and Comp3 will stay the same at $\frac{25\,x\,3}{20} =$ 3.75 $flops/cycle$.

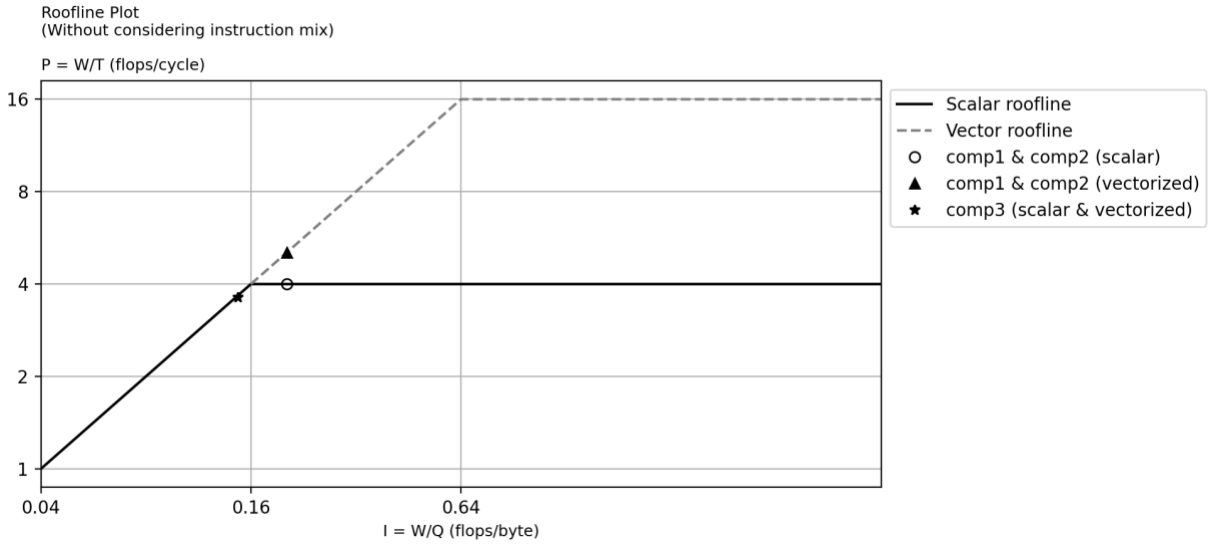

Fig. 1. Roofline plot for bounds and the programs without considering the instruction mix (fix scalar computations)

(c) Now, derive a hard upper bound on the performance of each function based on the instruction mix and compulsory misses. Again, assume that no optimizations that change operational intensity are performed, and FMAs are used to fuse an addition with a multiplication whenever applicable. For readability, place the new performance dots on a separate roofline plot (there should be six dots).

Solution:

Unvectorized:
Comp1: Performance of comp1 drops to 3 flops/cycle when we consider instruction mix. We achieve it by doing one addition and one FMA.

Comp2: Performance of comp2 drops down to 2 flops/cycle because there are 3n addition

operations but we can only do 2 addition per cycle because of port limitations, so, total execution lasts 1.5n cycles.

Comp3: There are 3n FMA operations and we can issue 2 FMAs per cycle, so, total execution lasts 1.5n cycles. Each FMA is 2 flops (6n flops in total), so, we could have achieved 4 flops/cycle if this operation was not memory bound. Since it is memory bound, however, we will hit the roof at 3.75 flops/cycle.

Vectorized:
Comp1: At I(n) = $\frac{3}{16}$ the maximum performance that can be attained is 4.6875. The maximum performance comp1 can attain based on vectorization and instruction mix is 12 flops/cycle. So, comp1 will be memory bound and it will have a performance of 4.6875 flops/cycle.

Comp2: At I(n) = $\frac{3}{16}$ the maximum performance that can be attained is 4.6875. The maximum performance comp2 can attain based on vectorization and instruction mix is 8 flops/cycle. The computation will hit the memory bound and have a performance of 4.6875.

Comp3: At I(n) = $\frac{3}{20}$ the maximum performance that can be attained is 3.75. The maximum performance comp3 can attain based on vectorization and instruction mix is 16 flops/cycle. So, comp3 will be memory bound and it will have a performance of 3.75 flops/cycle.

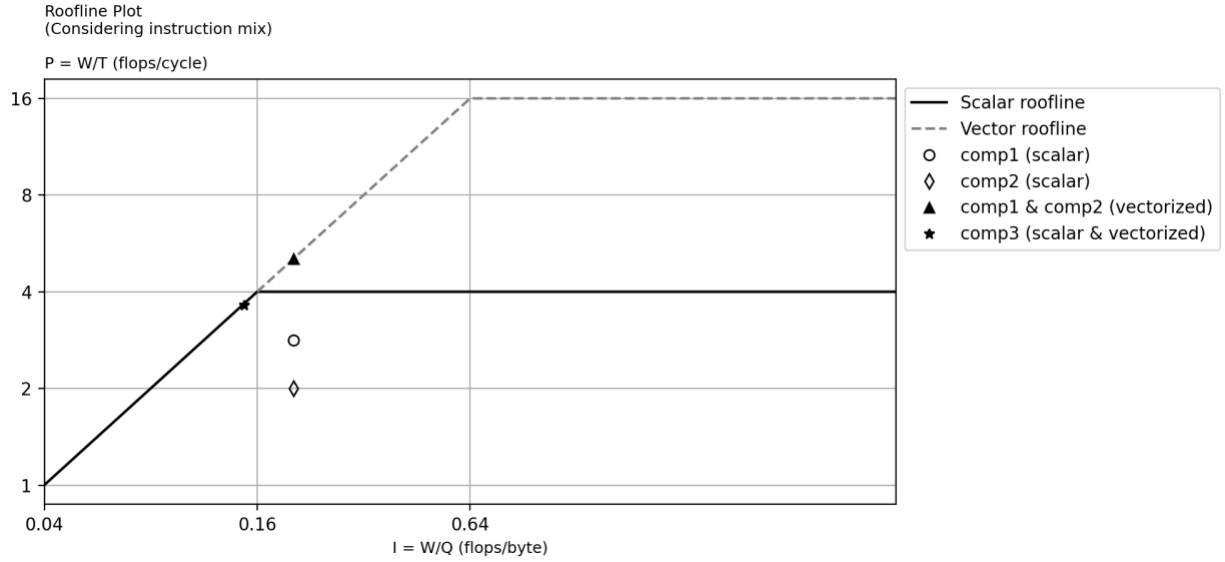

Fig. 2. Roofline plot for bounds and the programs considering the instruction mix

(d) How will the operational intensity and performance of comp3 change if we increase r (i.e. if we manually assign higher values to r in line 2)? What is the best possible performance of comp3 when increasing r and when implemented with and without vector instructions?

Solution:

For r much bigger than n, the operational intensity will converge towards I(n) = $\frac{1}{4}$. Without vectorization, the performance of comp3 will become 4 flops/cycle (Still using the assumption that additions and multiplications will be fused into FMAs whenever possible, otherwise, the performance becomes 2 flops/cycle when we consider the instruction mix since we have to issue additions and multiplications in separate ports). With vectorization comp3 will still hit memory bound since, at 0.25 flops/byte operational intensity the maximum performance is 6.25 flops/cycle which is less than the maximum performance of 16(8, without the assumption) flops/cycle comp3 can attain based on its instruction mix and vectorization.

4. Cache Miss Analysis (20 pts)

Assume that the code is executed on a machine with a write-back/write-allocate fully-associative cache with blocks of size 64 bytes, a total capacity of γ = 8KiB and with a LRU replacement policy. Assume that γ << n.

(a) Calculate the total number of cache misses that this computation has. You can ignore lower order terms. Show your work.

Solution:

For each row of A, we will access the elements of the row n times for each column of B. The no. of non-zero columns decrease by 1 compared to the previous row. We access the elements of A consecutively. So, we will have n$\sum_{i=0}^{n-1} \left\lceil \frac{n-i}{8} \right\rceil$ cache misses for A in total. We need to put a ceiling function inside summation because not all no. of non-zero elements will be divisible by the no. of doubles a cache block can contain, so, we will miscount if we don't have it.

For C, we will access all of its elements repeatedly and consecutively, so, when we bring a block of C, the next seven positions will be brought as well and we will have $\frac{n^2}{8}$ cache misses for C in total. We can ignore this since it is clearly a lower order term than cache misses for A. ($O(n^2)$ compared to A's $O(n^3)$)

For B, for each iteration of the outermost loop (i loop) we will read one less row of B then we did in the previous iteration, so, in total we will have $n \sum_{i=0}^{n-1} n - i$ cache misses for B.

However, these cache miss calculations are an approximation and there are some complicated lower order subtraction and addition terms that must be accounted for. The reason is, since, we do not read the entire row of A and we skip many rows of B, towards the end of the computation, there comes a point where the non-zero elements of A in the accessed row and rows of B are still in cache when the next column in B is accessed. This, however, is negligible since we can only store 128 blocks in the cache, it means that we will see this different access pattern for the last x rows where x is < 128 and considering n is much larger than cache size, this number will be a low order term.

So, the cache misses when we ignore the low order terms should be around

$$n\sum_{i=0}^{n-1}\left\lceil\frac{n-i}{8}\right\rceil+\frac{n(n(n+1))}{2}\approx\frac{9n^3}{2\times 8}=\frac{9n^3}{16}$$

(b) Based on the characteristics of the machine, determine a suitable value of b that improves locality, i.e. that reduces cache misses.

Solution:

Similar to the argument made for MMM, 3 blocks should fit into our cache to only get compulsory misses, thus we have the following:

$$3b^2\times 8\ bytes\leq 8\times 2^{10}\ bytes,$$

$$b=\left\lfloor\frac{32}{\sqrt{3}}\right\rfloor=18$$

(c) Calculate the total number of cache misses that this computation has when using blocking. You can ignore lower order terms.

Solution:

Since, for this problem we do not have a provided code I will state some of my assumptions. The calculations of blocks of C is similar to the calculation of elements of C in part a. So, when we start reading blocks of A we do not read the elements we know to be 0 for sure (elements behind the diagonal) and when we read B, we do not read blocks of B that we know will correspond to a 0 block in A. So, in each "block row*" iteration of C, we read one less "block row of B" than we did in the previous iteration. We also assume that 3 blocks fit into cache.

*Note: By block row, I mean a b x n portion of the matrix.

When we are reading the half blocks of A (blocks on the diagonal), we will get the following amount of cache misses per half block:

$$\sum_{i=0}^{b-1}\left\lceil\frac{b-i}{8}\right\rceil$$

We will read $\left(\frac{n}{b}\right)^2$ half blocks at the end for each block of C.
For each half block of A, we will read one full block of B which will create the following amount of cache misses per block:

$$\frac{b^2}{8}$$

We will read $\left(\frac{n}{b}\right)^2$ such blocks from B to multiply with the half blocks of A.
Rest of the cache misses are caused by full block MMMs of both A and B. Because of the assumption made in the beginning of the question (we do not start iterations over matrix elements whose multiplication will result in 0 for sure.) we read one less block in each row block iteration of C compared to the previous iteration. So, for example, while computing the first row block of C we read n – 1 full blocks per matrix (ignoring the half block of A and its corresponding full block of B we already read). In the second row block of C we read n – 2 full blocks and so on. For these operations we will have the following amount of cache misses:

$$\frac{n}{b}\sum_{i=1}^{\frac{n}{b}}\left(\frac{(n-ib)b}{8}+\frac{(n-ib)b}{8}\right)=\frac{n}{b}\sum_{i=1}^{\frac{n}{b}}\left(\frac{(n-ib)b}{4}\right)=\frac{n^3}{8b}+\frac{n^2}{8}$$

So, combining all of the above we will have the following amount of cache misses:

$$\frac{n^3}{8b}+\frac{n^2}{8}+\frac{n^2}{b^2}\frac{b^2}{8}+\frac{n^2}{b^2}\sum_{i=0}^{b-1}\left\lceil\frac{b-i}{8}\right\rceil=\frac{n^3}{8b}+\frac{n^2}{4}+\frac{n^2}{b^2}\sum_{i=0}^{b-1}\left\lceil\frac{b-i}{8}\right\rceil\approx\frac{n^3}{8b}=\frac{n^3}{144}\ when\ b=18$$