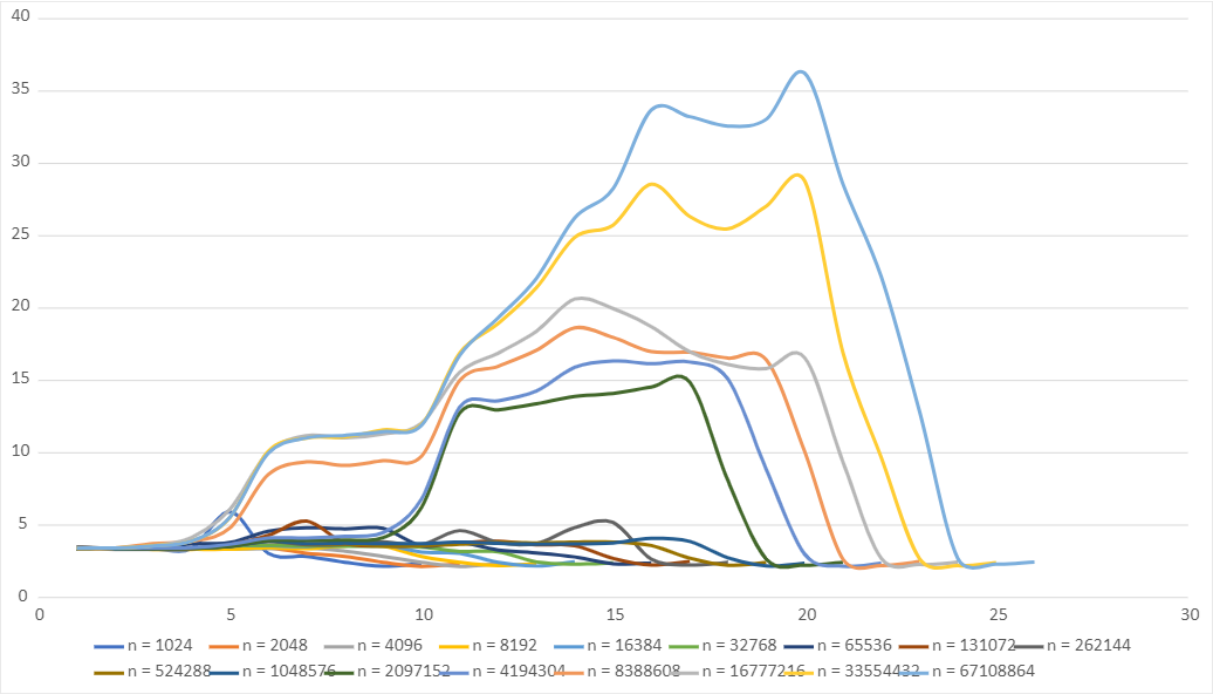
Analysis for Problem1 benchmark



The reults of (stride, time) pairs for array size from 2^10 to 2^26 is shown in the plot above.

1. L1 Cache size: ???...
2. L1 cache line length: 64 bytes(16 words)
3. L1 associativity: 8 lines per set, the degree of associativity is 3.
4. Analysis:

WarmUp :
1 float = 4 bytes
For a 64bits CPU, 1 word = 8 bytes(_ _ _)
imagine both Main Memory and Cache Memory use byte addressing
imagine cache line length = 8 words, which means “offset” contains 6 bits _ _ _| _ _ _ for ex. 101|000 (3 index bits)
Therefore, 1 line can store 8 * 8 bytes = [16] elements

Stride = 2^s (s = 0:26)
The address difference between A[i] and A[i + 2^s] is 2^s * 4 = [2^(s+2)] bytes (_ _ _ …_ _ s+2 bits in the byte addressing representation)

When s+2 <= 6 (_ _ _ _ _ _) , A[i] and A[i + 2^s] are in the same line, after loading A[i], A[i + 2^s] will be a hit.
Byte address 000 000 -> 111 111 are in the same line(A[i] = 000 000, A[i + 16] = 111 100,
A[i + 17] = 1 000 000)

When array size (2^n) is less than cache size, after the first loop of loading all elements into cache(compulsory miss), the repeated K - 1 times will all be a cache hit. MAT =
When stride = 1, the cache memory address of the A[i] we want to load/read changes like this :

First of all, array data are load to the first line of each set.
Set 0
A[0] 000 000,
A[1] 000 100,
A[2] 001 000,
A[3] 001 100,
…(16 elements)
A[15] 111100

Change to Set 1
A[16] 00...001 000 000
A[17] 00...001 000 100
..(16 elements)
A[31] 00...001 111 100

Change to Set 2
A[32] 00...010 000 000

Change to Set 3
A[48] 00...011 000 000

imagine there are S bits to represent sets(2^S sets in the cache)
Therefore,
A[16 * 2^S] 11...(S bits)..11 000 000

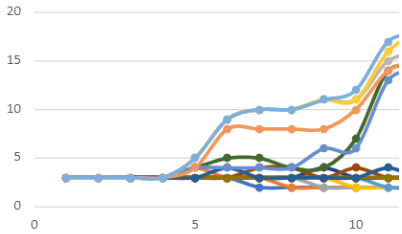
Tag Changed, array data will be load to line1
A[16 * (2^S + 1)] 00...1 00...(S bits)..00 000 000

...

====
L1 cache line length: 64 bytes(16 words)
Analysis:
As shown in the graph below, AMAT starts to rise when stride change from 2^4 to 2^5. The first rise should be caused by line length. So the cache memory addressing should have (4 + 2) bits offset _ _ _ _ _ _ , +2is because 1 float = 4 bytes, which needs to bits to represent. Each line can store 2^4 = 16 words = 64 bytes

For array that are within (the capacity of cache / the degree of associativity), whether we are visiting element in the same line or in different lines will make little AMAT difference after all the data have been loaded into cache.

For array that are within the capacity of cache but larger than (the capacity of cache / the degree of associativity) , …? Anyhow...

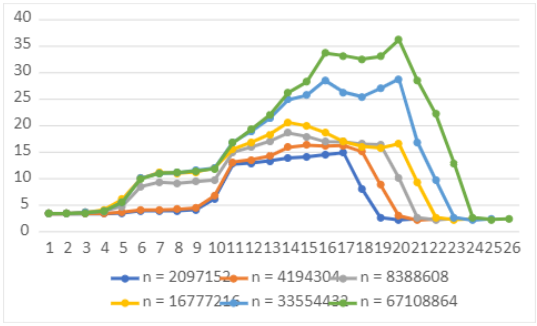


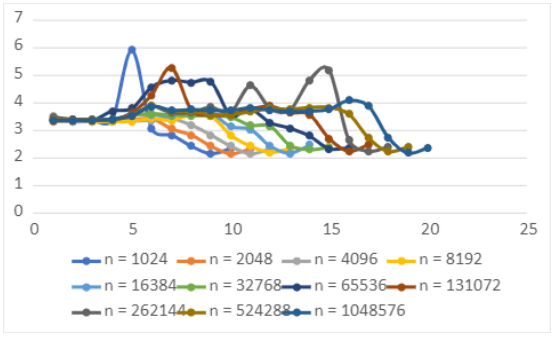
L1 Cache size:?

Analysis:

For each set, we have 8 * 64 = 512 bytes = 2^9 bytes

When the size of array are from 2^21 to 2^26, the graph shows different trend compare to shorter array.





We can see that when array size is larger or equal to 2^21, there shows a turning point when strip = 2^10.

L1 cache size is usually 256KB = 2^18 bytes which can store 2^16 elements.

We cannot find this answer through the graph.

L1 associativity: 8

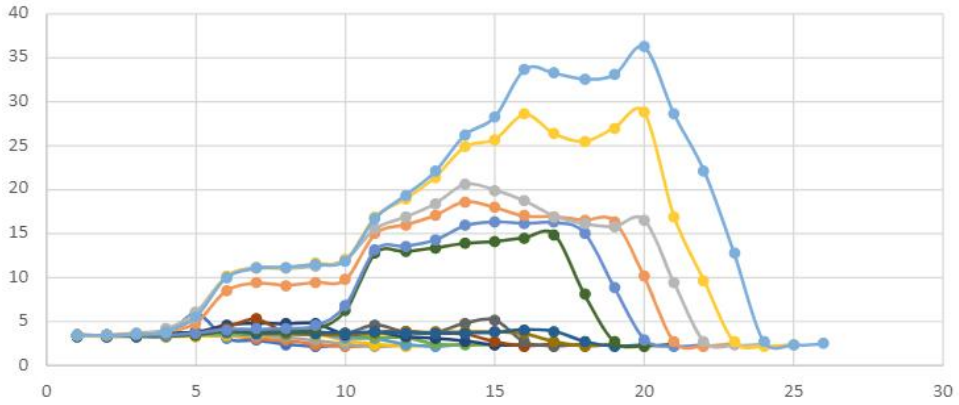
Analysis:

For cache memory byte addressing, we consider Set (S bits) offset(6 bits, for float type, the last 2 bits are always 00)

When stride is larger than 2^(S + 4), the cache address for set and offset will be fixed, which means the data we visit will be stored into the same Set (Set 0).

As we can see in the form and graph below， AMAT is pretty low when we repeatedly reading 2/ 4/ 8 elements in the array, but starts to rise quickly when we visit more than 8 elements of the large array.

n = 2^	21	22	23	24	25	26
strip = 2^	n = 2097152	n = 4194304	n = 8388608	n = 16777216	n = 335544	n = 67108864
14	14.102135	16.350526	17.974447	19.974416	25.73035	28.24828
15	14.541327	16.155011	16.991308	18.717433	28.56358	33.69456
16	14.888977	16.273866	16.947955	16.980063	26.33717	33.23351
17	8.069879	15.094753	16.542129	16.105428	25.48679	32.57967
18	2.655066	8.85356	16.42944	15.820655	27.04045	33.05054
19	2.234858	2.995675	10.108809	16.579899	28.80363	36.24359
20	2.4333	2.160184	2.652009	9.377523	16.93299	28.5811
21		2.365589	2.212624	2.711807	9.673381	22.1749
22			2.481996	2.278022	2.689599	12.86289
23				2.443066	2.195947	2.692001
24					2.411348	2.308865
25						2.449829



The reason of this phenomena is that, we are using different lines of the same set when the bit for set does not change.

The distance between the elements we want to visit(element_visit) is longer than one line's capacity(16 elements), therefore, each element_visit is stored in a line apart from other element_visit we are interested in, and of course these elements are stored with their 15neighbors together in the line.

When the amount of element_visit is larger than the amount of lines in each set, we need to replace elements already stored in the set with new element_visit, therefore, there will be cache miss all the time, AMAT will rise quickly.

Since each set has 2^3 = 8 lines, the degree of associativity is 3.

Analysis for Problem2 matrix multiplication

How does the performance of the sequential row-by-column compare to the sequential blocked matrix-matrix multiplication?

matrix size	rbyc	tile 128	tile 64	tile 32	tile 16	tile 8	tile 4
8192*8192	too long time						
1024 *1024	11739886936	13785793375	12134730354	10842595221	8872520218	9032457282	9314215566
256*256	127769763	130784506	124651736	122434493	123065982	146904675	129096460

The tile multiplication costs less time when the matrix is too big to be load into cache all at once and when the block size is not too big to be load into the cache.

In theory, block size should obey the following rule to acquire good result:

$$3b^2 \leq M \;\Rightarrow\; q \approx b \leq \left(\frac{M}{3}\right)^{\frac{1}{2}}$$

---from course ppt

How does perfomance vary with different dimensions of tiles? What is the speed-up (or slow down) when tiling is used?

When tile is too large, the time cost by tile multiplication starts to become larger than the time caused by rbyc multiplication. When the matrix size is small, rbyc and tile multiplication perform similar to each other. It's difficult for me to find the speed-up according to the data. In theory,

For rbyc:

Load $B(:,j)$ n times for total n^3 loads

Laod $A(i,:)$ only one time for total n^2 loads

Load and store $c(i,j)$ once for total $2n^2$ slow memory ops

Total slow memory ops are n^3+2n^2, $2n^3$ flops

$\mathbf{q \approx 2}$, what did go wrong? Weak reuse of B.

---from course ppt

For tile :

load $A(i,k)$ and $B(k,j)$ p^3 time for total of $2p^3b^2 = 2pn^2$ loads

load and store each $C(i,j)$ once for total of $2n^2$ memory ops

total memory ops is $(2p+2)n^2$

$q = \frac{2n^3}{(2p+2)n^2} \approx \frac{n}{p} = b$

---from course ppt