CS-622A
Assignment-2
Report

Sumit Lahiri 19111274

Radhit Dedania 150544

Group No 13

PART-I : Collection of Traces

The following table depicts the total count of memory accesses per trace(program). Here, ”$trace_i$” means the memory trace obtained due to the execution of $program_i$.

NOTE:
Everytime the programs are executed, the total no. of memory accesses corresponding to each come out to be different but the output of the respective programs come out to be the same(on consecutive executions). Thus, the correctness of the program is maintained but the memory trace gets affected by the randomness involved due to the usage of threads.

NOTE:
It has been observed that the maximum memory access size is around 32 bytes. Thus, a memory access can either remain in the same block or spill over maximum two blocks.

In our work, we make two cases for splitting memory accesses minimally in the sizes of 1, 2, 4 and 8 bytes:

- If the size of memory access is such that it spills over a block, it is broken down into two parts. One of the part is called ”buffer” which belongs to the same block as the starting address of access and the other part is called ”overflow” which trespasses into the next consecutive block. We recursively divide this two parts into chunks of 8, 4, 2, 1 bytes till a part can no longer be broken down into a chunk of particular size(start from 8, next to 4, next to 2, last to 1). We also separate the read and write accesses and consider them as two different access.

- If the size of memory access is such that it remains within the block of the starting address of access, it is recursively divided into chunks of 8, 4, 2, 1 bytes till it can no longer be broken down into a chunk of particular size(start from 8, next to 4, next to 2, last to 1). We also separate the read and write accesses and consider them as two different access.

Memory Trace(Program)	Total memory accesses
1	155069021
2	3241790
3	14388785
4	1099232

PART-II : Access Distance Analysis

Following are the plots of cumulative frequency distributions of access distance for each program trace. Here, i^{th} plot corresponds to $program_i$. CFD stands for "Cumulative Frequency Distribution".

In this part, we used an unordered map to keep track of the blocks. A block can be indexed using the access address($addr >> log\_block\_offset$). For each entry of the block in the unordered map, we maintain a vector that stores the serial no. of access that landed on that block. We, thus, get the access distance by using the vector corresponding to each block. Note that, finding the difference between two consecutive entries of any vector would give a particular access distance for that block. Thus, maintaining another unordered map for access distance would help in bookkeeping the frequency of a particular access distance(by maintaining a counter field corresponding to the access distance). In this way, we can calculate the frequency of access distances which can then be divided by N(total no. of accesses - no. of first accesses to any block) to get cumulative frequency.

Observations:

- Regarding CFD of access distance for access trace of Program 1, we can conclude that only access distances ranging from 1(10^0) to 100(10^2) have high frequency meaning that for a memory block used in Program 1, on an average at most after the 100^{th} access from an access to the block, the block is again accessed and reused. Thus, a cache of capacity 100*64(block size) bytes would have a good hit rate as most of the blocks in the cache would keep on getting hits. It could be a case that the same thread accesses the block on next access to the block in which case it would be better to keep the block in the private cache of a core(assuming different threads of a program run on different cores). If different threads access the same block on consecutive accesses to the block, it would be better to keep the block in the shared cache. Depending on the case, the capacity of the cache mentioned earlier might refer to the private or shared cache.
- Regarding CFD of access distance for access trace of Program 2, we can conclude that only access distances ranging from 1(10^0) to 4($10^{0.5}$) and from 316227($10^{5.5}$) to 1584893($10^{6.2}$) have high frequency meaning that for a memory block used in Program 2, on an average after (1-4)th or (316227-1584893)th access from an access to the block, the block is again accessed and reused. Thus, it would be better to use two level cache hierarchy - fast L1 of capacity 4*64 bytes and slow L2 of capacity 1584893*64 bytes. It could be a case that the same thread accesses the block on next access to the block in which case it would be better to keep the block in the private cache of a core(assuming different threads of a program run on different cores). If different threads access the same block on consecutive accesses to the block, it would be better to keep the block in the shared cache. Depending on the case, the capacity of the cache mentioned earlier might refer to the private or shared cache.
- Regarding CFD of access distance for access trace of Program 3, we can conclude that only access distances ranging from 1(10^0) to 10(10^1), from 1000000(10^6) to 3162277($10^{6.5}$) and from 7943282($10^{6.9}$) to 10000000(10^7) have high frequency meaning that for a memory block used in Program 3, on an average after (1-10)th or (1000000-3162277)th or (7943282-10000000)th access from an access to the block, the block is again accessed and reused. Thus, it would be better to use three level cache hierarchy - fast L1 of capacity 10*64 bytes and medium L2 of capacity 3162277*64 bytes and slower L3 of capacity 10000000*64 bytes. It could be a case that the same thread accesses the block on next access to the block in which case it would be better to keep the block in the private cache of a core(assuming different threads of a program run on different cores). If different threads access the same block on consecutive accesses to the block, it would be better to keep the block in the shared cache. Depending on the case, the capacity of the cache mentioned earlier might refer to the private or shared cache.
- Regarding CFD of access distance for access trace of Program 4, we can conclude that on an average the blocks are again accessed after at most 63095($10^{4.8}$)th access again to the same block. We could keep a cache of capacity 63095*64 bytes to retain all the blocks which will keep on getting hits as their average access distance is at most the no. of blocks in the cache. It could be a case that the same thread accesses the block on next access to the block in which case it would be better to keep the block in the private cache of a

core(assuming different threads of a program run on different cores). If different threads access the same block on consecutive accesses to the block, it would be better to keep the block in the shared cache. Depending on the case, the capacity of the cache mentioned earlier might refer to the private or shared cache.

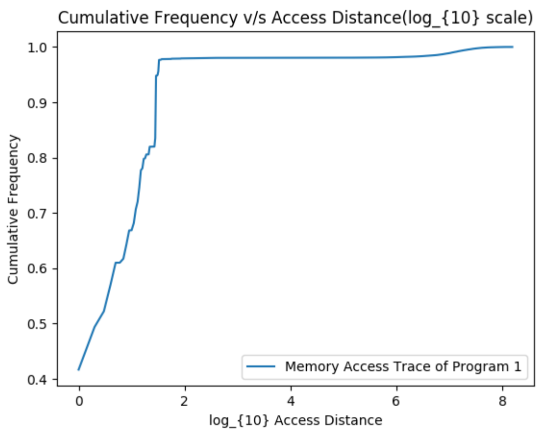

Figure 1: CFD of Access Distance for access trace of Program 1

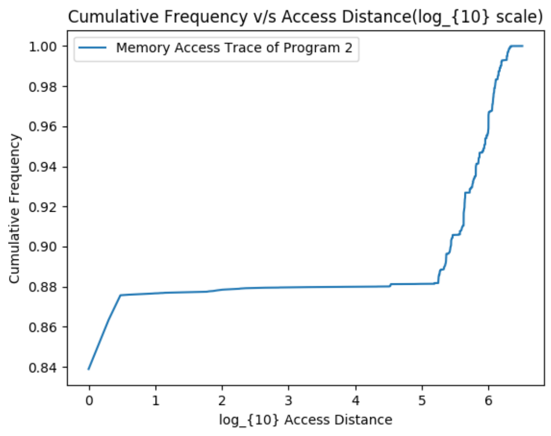

Figure 2: CFD of Access Distance for access trace of Program 2

PART-III : Access Distance filtered by LRU Cache

In this part, we used an unordered map to keep track of the blocks. A block can be indexed using the access address($addr >> log\_block\_offset$). For each entry of the block in the unordered map, we maintain a vector that stores the serial no. of access that landed on that block. We, thus, get the access distance by using the vector corresponding to each block. Note that, finding the difference between two consecutive entries of any vector would give a particular access distance for that block. Thus, maintaining another unordered map for access distance would help in bookkeeping the frequency of a particular access distance(by maintaining a counter field corresponding to the access distance). In this way, we can calculate the frequency of access distances which can then be divided by N(total no. of accesses - no. of first accesses to any block) to get cumulative frequency. The difference from part-2 is that in part-2, the original address stream was used for CFD plot but in this part, we use the missed address stream(addresses that missed in the single-level cache) to get the CFD plot.

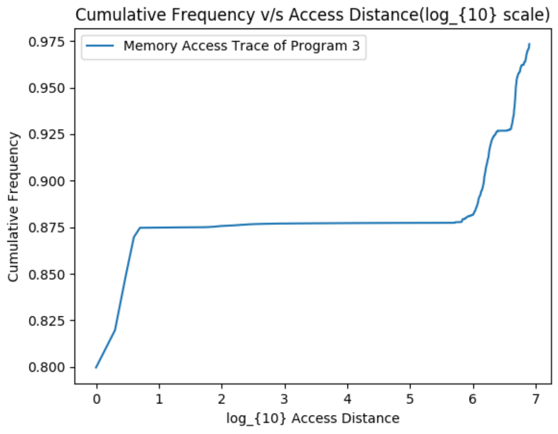

Figure 3: CFD of Access Distance for access trace of Program 3

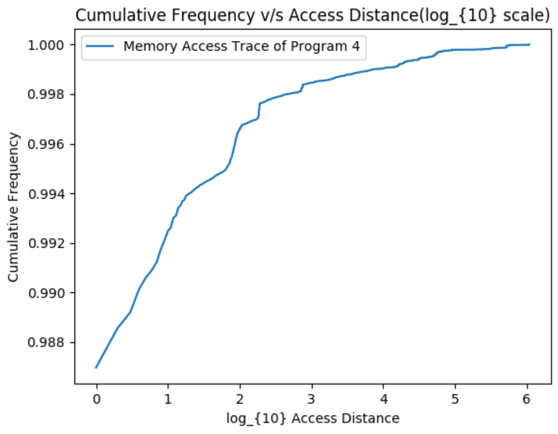

Figure 4: CFD of Access Distance for access trace of Program 4

0.1 Plots

Following are the plots of cumulative frequency distributions of access distance for each miss trace of program after passing through the single-level cache hierarchy. Here, i^{th} plot corresponds to the miss trace of $program_i$. CFD stands for ”Cumulative Frequency Distribution”.

Explanations for observed changes:

- In case of miss trace plot of Problem 1 as compared to it's original access plot, we find that the frequency corresponding to the access distances ranging from 1 to 100 has fallen drastically as the blocks having average access distance of about 100 got a large no. of hits as the capacity of the used cache was 2MB(2^{15} blocks(64-byte size) supported). $log_{10}(2^{15}) = 4.5$. Thus, all the requests to the blocks having average access distance below $10^{4.5}$ got satisfied and hence, the miss stream trace was nearly devoid of the accesses that corresponded to access distances below 100. In the earlier case, all the accesses having access distances beyond 100 had a very small frequency. Now, the frequencies of all the other accesses(¡100 access distance) has dropped so low that it has become comparable to them as can be observed from the plot.
- In case of miss trace plot of Problem 2 as compared to it's original access plot, we find that the frequency corresponding to the access distances ranging from 1 to 4($10^{0.5}$) has fallen to zero and from 316227($10^{5.5}$) to 1584893($10^{6.2}$) has decreased drastically as the blocks

having average access distance of about 4 got all hits and of access distance (316227-1584893) got large no. of hits as the capacity of the used cache was 2MB(2^{15} blocks(64-byte size) supported). $log_{10}(2^{15}) = 4.5$. Thus, all the requests to the blocks having average access distance below 4($10^{0.5}$) got satisfied and hence, the miss stream trace was devoid of the accesses that corresponded to access distances below 4. In the earlier case, all the accesses having access distances beyond 4 till 316227 had a very small frequency. Now, the frequencies of all the other accesses has dropped so low that it has become comparable to them as can be observed from the plot.

- In case of miss trace plot of Problem 3 as compared to it's original access plot, we find that the frequency corresponding to the access distances ranging from 1 to 10(10^1) has fallen to zero, from 1000000(10^6) to 3162277($10^{6.5}$) and from 7943282($10^{6.9}$) to 10000000(10^7) has decreased drastically as the blocks having average access distance of about 10 got all hits and of access distance (1000000-3162277) and (7943282-10000000) got large no. of hits as the capacity of the used cache was 2MB(2^{15} blocks(64-byte size) supported). $log_{10}(2^{15}) = 4.5$. Thus, all the requests to the blocks having average access distance below 4($10^{0.5}$) got satisfied and hence, the miss stream trace was devoid of the accesses that corresponded to access distances below 10. In the earlier case, all the accesses having access distances beyond 10 till 1000000 and from 3162277 to 7943282 had a very small frequency. Now, the frequencies of all the other accesses has dropped so low that it has become comparable to them as can be observed from the plot.
- In case of miss trace plot of Problem 4 as compared to it's original access plot, we find that the frequency corresponding to the access distances ranging from 1 to 316($10^{2.5}$) has fallen to zero and from 316($10^{2.5}$) to 63095($10^{4.8}$) has fallen considerably as the blocks having average access distance of about 316 got a large no. of hits as the capacity of the used cache was 2MB(2^{15} blocks(64-byte size) supported). $log_{10}(2^{15}) = 4.5$. Thus, all the requests to the blocks having average access distance below $10^{2.5}$ got satisfied and hence, the miss stream trace was devoid of the accesses that corresponded to access distances below 316. In the earlier case, all the accesses having access distances beyond 63095 had a very small frequency. Now, the frequencies of all the other accesses(316 to 63095 access distance) has dropped so low that it(beyond 63095) has become comparable to them as can be observed from the plot.

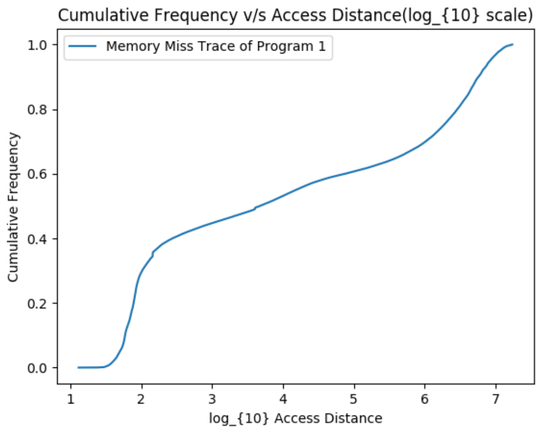

Figure 5: CFD of Access Distance for miss trace of Program 1

0.2 Hits and Misses

The following table reports the no. of hits, misses and total accesses of the memory trace of a program when passed through a single-level cache hierarchy. Here, "$trace_i$" means the memory trace obtained due to the execution of $program_i$.

Observations:

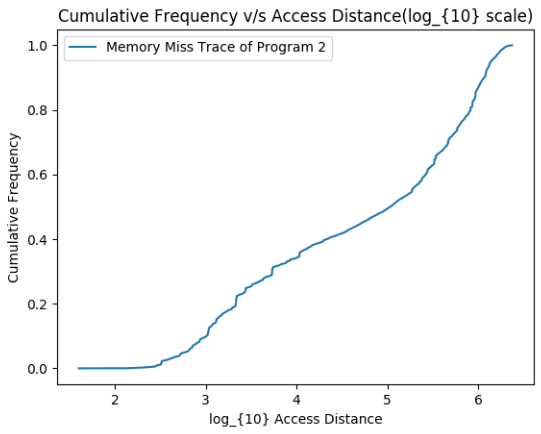

Figure 6: CFD of Access Distance for miss trace of Program 2

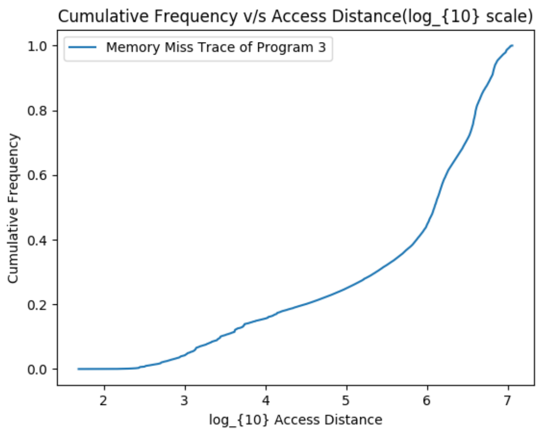

Figure 7: CFD of Access Distance for miss trace of Program 3

- As there are more hits than misses in case of trace 1, it can be concluded that majority of the hit blocks had an average access distance of 100 as the access distance ¡100 has very low frequency in the miss trace CFD plot. Also, as the cache capacity was very large(larger than the 100*64 bytes), this became possible.

- No. of hits are less than the no of misses for trace 2 and 3. Hence, it can be hypothesized that majority of the hit blocks had an average access distance of 4(trace 2) or 10(trace 3) because these access distances have completely disappeared from the miss stream CFD plot for trace 2 and 3(as the accesses corresponding to them got filtered out(got a hit) of cache) while the blocks having higher access distances(of high frequency in original plot) have not vanished completely but have become comparable to blocks that had low frequency access distances.

- For trace 4, double the misses as compared to hits are observed. Here again, majority of blocks having average access distance of 316($10^{2.5}$) or less have got continuous hits before they got evicted. Due to this, the frequency of such access distances dropped to zero. Other access distances having value such that the condition ($64 * access\_distance[bytes] < cache\_capacity[bytes]$) is satisfied, had their frequency stoop low as the accesses corresponding to them also enjoyed a large no. of hits in the cache.

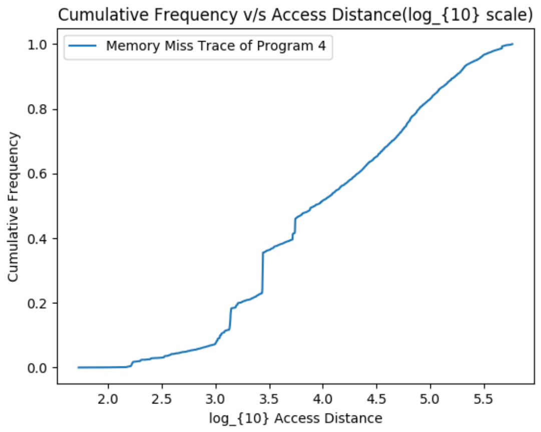

Figure 8: CFD of Access Distance for miss trace of Program 4

Memory Trace(Program)	Cache Hits	Cache Misses	Total memory accesses
1	84600703	17235806	101836509
2	658273	2373866	3032139
3	4106767	11415276	15522043
4	210348	585869	796217

PART-IV : Sharing Profile

The following table lists the no. of blocks shared by exactly n threads(where n ranges from 1 to 8) for each program trace. Here, "$trace_i$" means the memory trace obtained due to the execution of $program_i$. Below, "Ex n" stands for total no. of memory blocks shared by exactly n threads of a program.

In this part, we maintain an unordered map to keep track of the blocks that get accessed. If a block is accessed, the map is indexed using the block no.($addr >> log\_block\_offset$) and the information related to thread id is inculcated in the corresponding field. At last, we group together all the blocks that get accessed by exactly same no. of threads. Thus, we get the following table.

It can be observed that total no. of memory blocks accessed by trace 1 is higher than that of any other trace. Thus, it would require a cache of huge capacity to cache all the memory blocks accessed by trace 1 as compared to other traces. Also, for trace 1, 2 and 4, the no. of memory blocks accessed by exactly one thread(private blocks) is higher than the ones accessed by more threads. Thus, it would be beneficial to cache this blocks in private caches of cores if the threads of a program are run on different cores. For trace 3, the no. of memory blocks accessed by exactly two threads is fairly large, even larger than the no. of blocks accessed by exactly one thread. Thus, assuming different threads of a program run on different cores, blocks other than the private blocks should be placed in the shared cache of the cores so as to provide optimum performance for all the threads. Also, during block replacement in shared cache, priority should be given to blocks shared by multiple threads, i.e., they should be given priority over less shared ones to remain in the cache if a tie occurs. Apart from trace 1, all the other traces are under-utilising the 8 threads. They could have achieved the same performance using less no. of threads which would have allowed some other program's heavy thread an opportunity to run.

Trace	Ex 1	Ex 2	Ex 3	Ex 4	Ex 5	Ex 6	Ex 7	Ex 8	Total no. of blocks
1	1061447	224360	140985	119368	112507	90806	57029	22902	1829404
2	276003	243730	38918	1	1	0	9	21	558683
3	1651	1060828	233513	11	0	0	9	17	1296029
4	312857	21	0	1	0	1	8	22	312910