ASSIGMENT 2 CS423

Pramodh Gopalan
190933
CSE, Indian Institute of Technology Kanpur
{pramodh}@iitk.ac.in

Ayush Kumar
190213
CSE, Indian Institute of Technology Kanpur
{kumayu}@iitk.ac.in

How to run the code

- We have provided the instrumentation code in a single file named *addrtrace.cpp*. The programs to be instrumented need to be compiled as

  ```
  gcc -O3 -static -pthread prog.cpp -o prog
  ```

- Inside the folder *pin-3.26-98690-g1fc9d60e6-gcc-linux/source/tools/CS423*, the .so file from our instrumentation code can be generated using the commands

  ```
  make obj-intel64/addrtrace.so
  ```

- Then the *prog.cpp* file's binary can be instrumented using the command -

  ```
  ../../../pin -t obj-intel64/addrtrace.so -- bin/prog1 8
  ```

  Here, 8 is the number of threads the program needs in the command line.
- We have also provided a python file named plot_cdf.py to plot three cdfs plots, one with only global cdf, one with using cache misses and another using both to compare them. The file can be run using -

  ```
  python3 plot_cdf.py
  ```

Question 1

Report the total number of machine accesses recorded in the trace for each of the four programs.
The trace of machine accesses prepared in this part needs to be used in the following parts of the assignment.
The total machine accesses cannot be the same from one run of the program to the next due to memory allocation mechanics. However, the order and the approximate number still remains the same. Here, we make a table for the machine accesses for each of these programs.

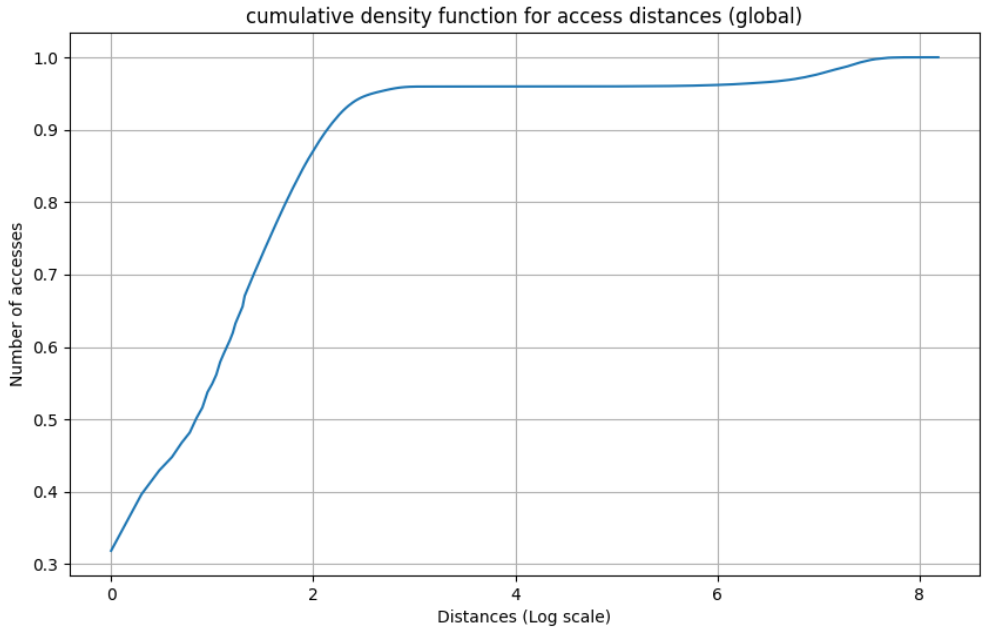
Program	Machine Acessess	Cache Machine Accesses
prog1	153112710	6664803
prog2	2683247	208645
prog3	11014224	626979
prog4	1068980	125954

Table 1: Table of Machine Accesses of Programs

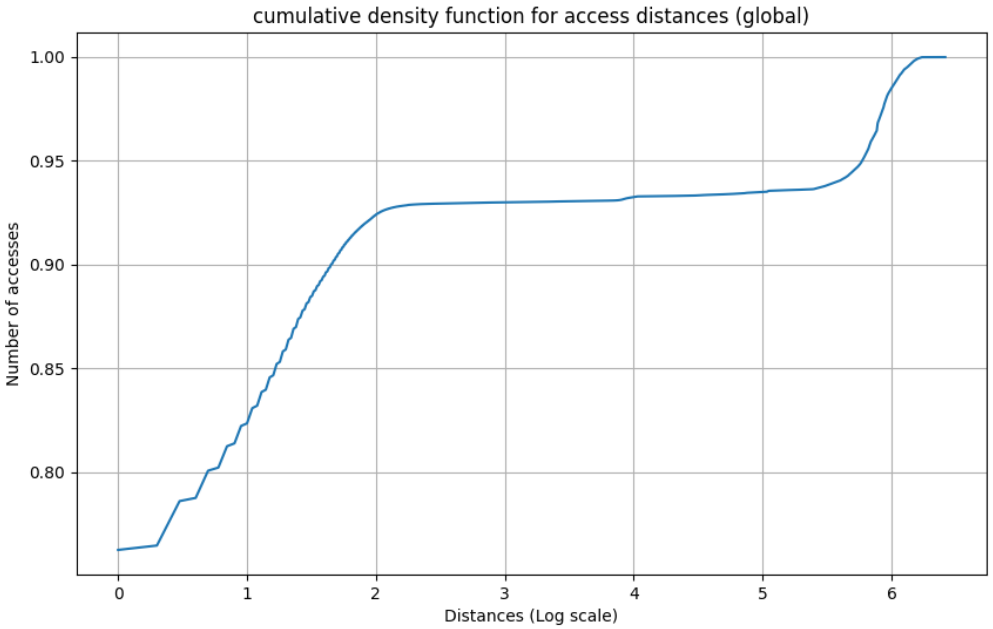
Question 2

Write a program to compute the cumulative density function of access distances from the traces. Submit the four cumulative density function plots, one for each trace. When preparing the cdf plots, please use a log-to-the-base-ten scale on the distance axis (this should be your x-axis).
The plots for cumulative density function for access distances for the four programs are as shown -

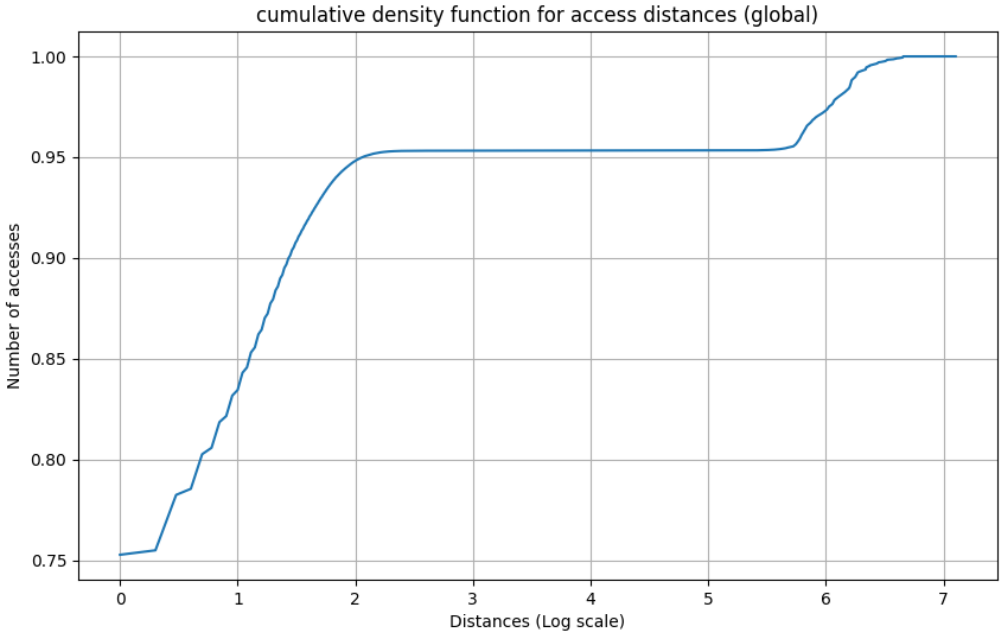
- prog1 -



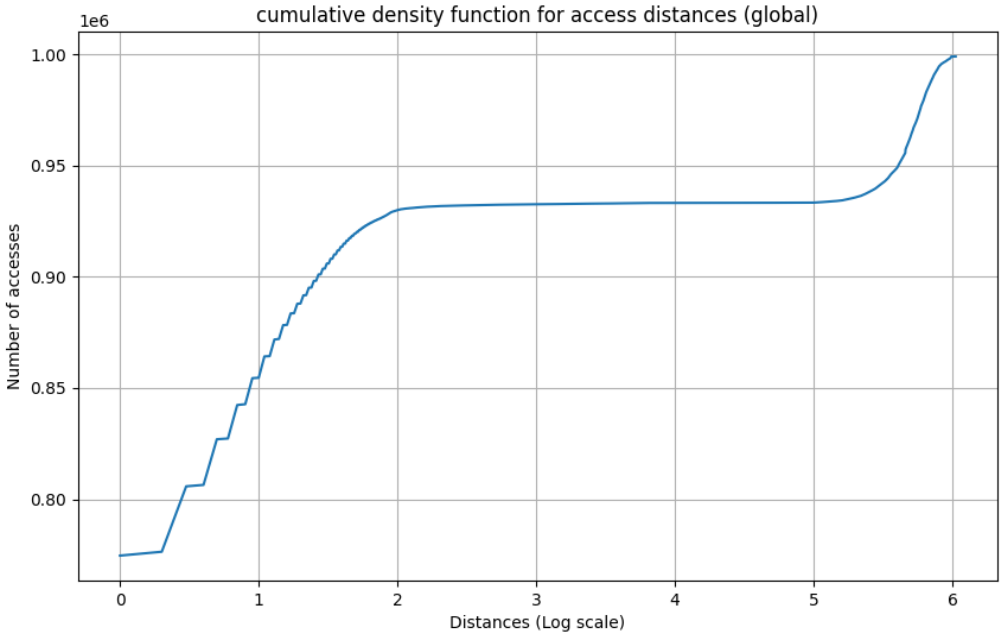
- prog2 -



- prog3 -



- prog4 -



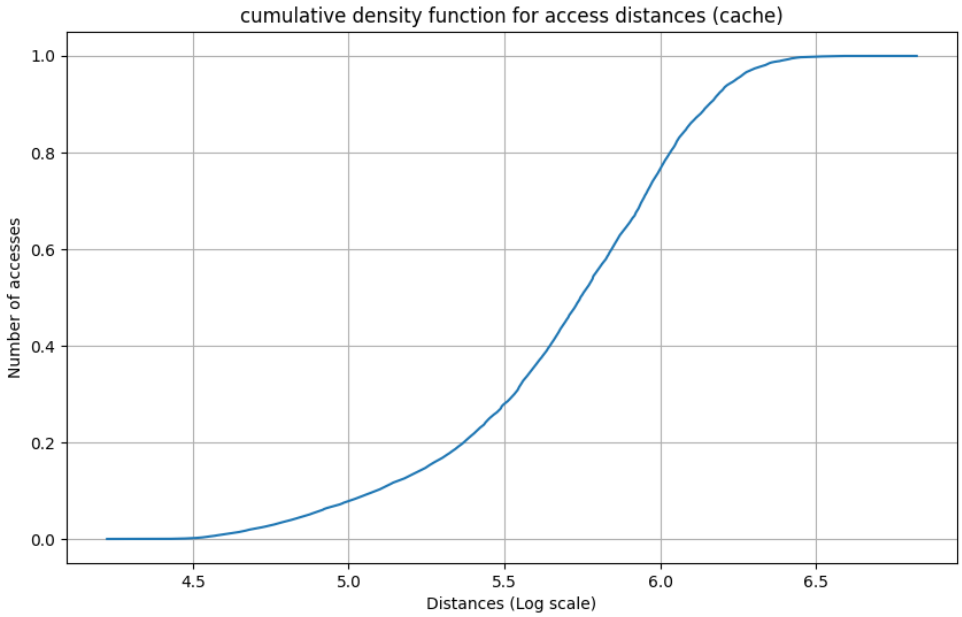
Question 3

Model a single-level 2 MB 16-way cache with 64-byte block size and LRU replacement policy. Pass each trace through the cache and collect the trace of accesses that miss in the cache. Report the number of hits and misses

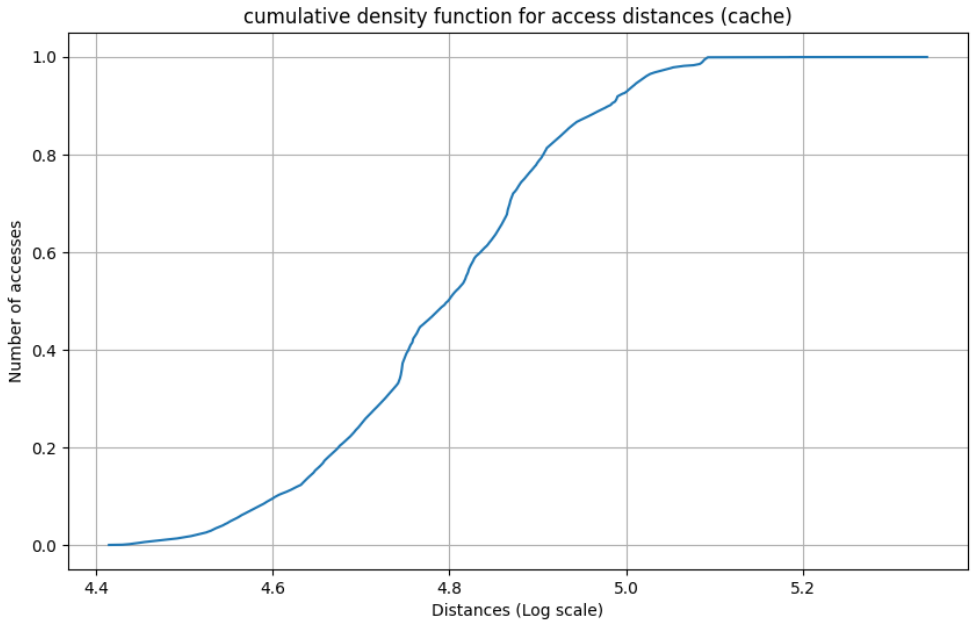
for each trace. Prepare the cumulative density function of the miss trace for each of the programs. Submit the four plots. Comment on whether the shape and nature of cumulative density function before and after the cache differ in any noticeable way. Explain your observation. Note that at the beginning of each trace, the cache is assumed to be empty. Also, note that the thread id should be completely ignored in parts II and III.

The plots are as follows -

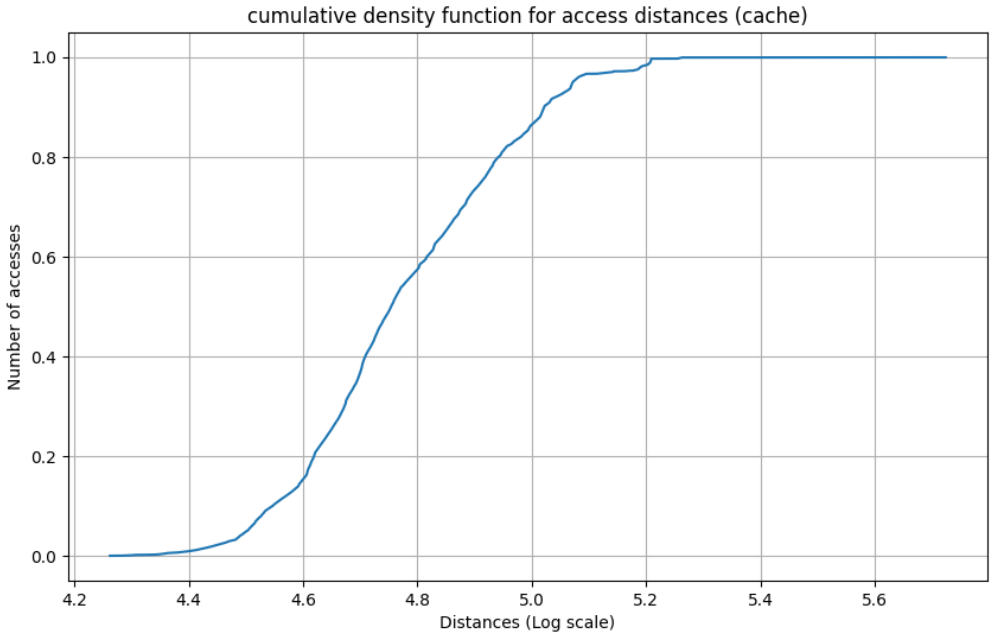
- prog1 -



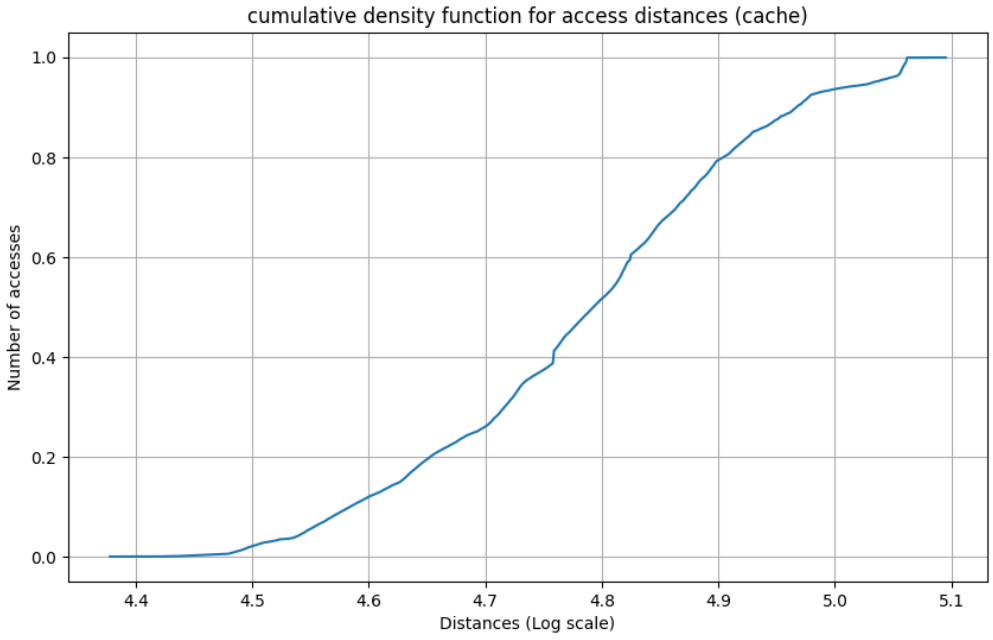
- prog2 -



- prog3 -



- prog4 -



- One straightforward observation from comparing the plots of cdf of access distances is that in the cache version, the plots is concentrated within a smaller distance range on x axis as compared to their global counterparts. For eg in the case of prog1.c, the distance on x axis are spread from 0 to 8 in the global version, but in the cache version the same is spread from 4 to 7. The reason for this is that the presence of a cache means that the

requests for accesses of the blocks that are present in the cache are satisfied in the cache itself. So, since we log only the cache misses, the hits are hidden and thus all the accesses that are at a small distance from the previous access of a block are satisified in the cache. Also, the scale for x axis is smaller for the cache version since a lot of blocks end up being in the cache itself, which results in fewer in total blocks in miss trace, hence, the distance span upto a shorter number.

- Another observation regarding the global plots is the presence of a sudden increase at larger distance. We suspected this was due to the presence of prefetch instructions being inserted by the compiler, which meant that the block was prefetched much earlier but was logged much later; but upon inspecting this with PIN, we did not find any instruction that was instrumented (we used *if (INS_IsPrefetch(ins)) {}*). This sudden increase is not in any cache plot, where all the plots show a smooth increase with decreasing slope. This observation could also be paired up with the previous observation of the cache hiding away the smaller access distances and thus the global plot where the access distance is large is the only thing being shown by the cache miss trace analysis.
- Before the actual program starts, the main thread accesses the whole array, and initialises it. Later, in every program, all threads access these memory locations. The locations which get accessed by the threads now for the first time, but later in the trace would have large access distance, Since this occurrs a lot, it leads to large access distance(Once by this thread, previously by the main thread). This causes a spike in the global cdf towards the larger distances.

Question 4

In this part, you will make use of the thread id to derive the sharing profile. Write a program to compute the number of memory blocks that are private, or shared by two threads, or shared by three threads, ..., or shared by eight threads. The sum of these eight should be equal to the total number of memory blocks for each trace. Report these eight numbers for each trace in a table. The LRU cache of PART III should be removed in this part. We try to explain the sharing profile of every program below.

Program	1 Thread	2 Threads	3 Threads	4 Threads	5 Threads	6 Threads	7 Threads	8 Threads
prog1	496	70	1872	32455	143250	244970	173832	124528
prog2	488	8262	16384	40957	4	0	1	11
prog3	497	63	0	0	0	0	1	65545
prog4	8678	57409	6	0	0	0	1	11

Table 2: Table summing up Sharing Profiles of each of the programs

- prog1 - In this program, each thread makes threadID sized jumps, which causes a lot of threads to share blocks as these accesses can overlap. Hence we see a well distributed sharing profile spread across all the threads. Most of the blocks in this case get shared by larger number of threads, hence the sharing profile shows more blocks being shared by more threads.
- prog2 - Each thread is given a portion of array to work on. The arrays both access their left and right neighbors, so each block has a maximum of three sharers. Adding in the main thread that accesses the whole array at the beginning of the program, each block (in the array) mostly has 4 threads accessing it simultaneously.
- prog3 - In this program, the array is divided into chunks of equal sizes and each thread accesses these chunks. These chunks are then reassigned to different threads and then they access them, this is what leads to huge portion of blocks being shared by all the threads.
- prog4 - Each thread is given a chunk of memory for processing it. Hence, each section of the array is accessed only once by its allocated thread, and once by the main thread for initialising it. Hence, most of the blocks have two sharers, and the blocks having synchronizing variables are accessed by all 8 threads, leading to 11 blocks being shared by most threads.

References