Advanced Computer Architecture

CS622 Assignment 2: Report

Debanjan Chatterjee (20111016) and P J Leo Evenss (20111038)

Group: 13

1. Introduction

Objective: Firstly, to collect the thread-wise memory access trace for each of the four binaries provided using a PIN instrumentation tool. Secondly, using the obtained traces, to analyze the access distances (an approximation of reuse distance) and to plot the cumulative density functions of the analyzed pattern, and the effect of introducing a LRU cache. Finally, to analyze the sharing profile of the memory access among 8 threads.

2. Result Analysis

Problem 1:

The binaries provided is executed using instrumentation routine with 8 threads and by maintaining a memory block of 64 bytes, the memory accesses are divided into smaller machine accesses of 8, 4, 2 ,1 bytes, in order to maintain the memory block boundary conditions.

The total number of machine accesses recorded in the trace for each of the 4 programs is reported in the Table 1.

Table 1: Total machine accesses per program

Prog1	Prog2	Prog3	Prog4
140525259	2537215	9606211	1064528

Observation:
Each execution of the instrumentation tool might lead to a variation in the values of total machine accesses due to the execution by parallel threads.
The read and writes to memory access are considered separately.

Problem 2: Access Distance Analysis

The Cumulative Density Function of the to the access distances (log base 10) obtained as the outputs is plotted in the figures given below for each program trace.

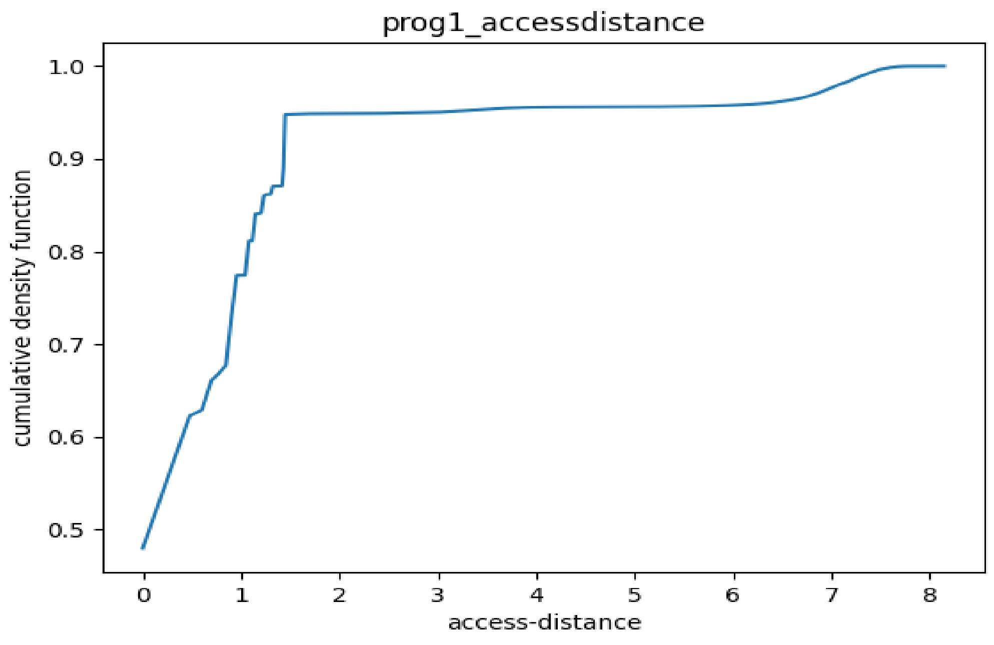

Figure 1: Cumulative Density Function Prog1

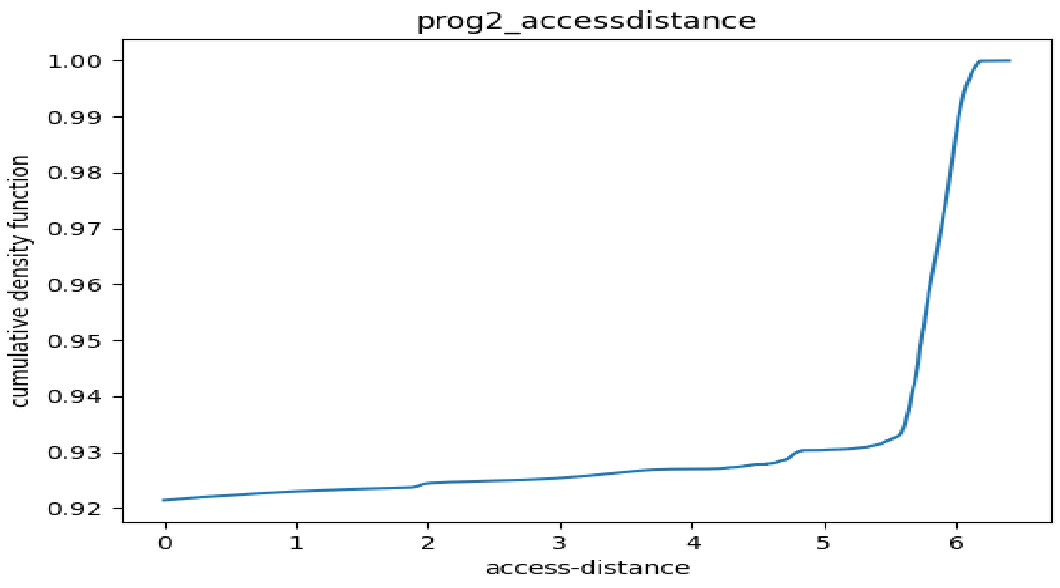

Figure 2: Cumulative Density Function Prog2

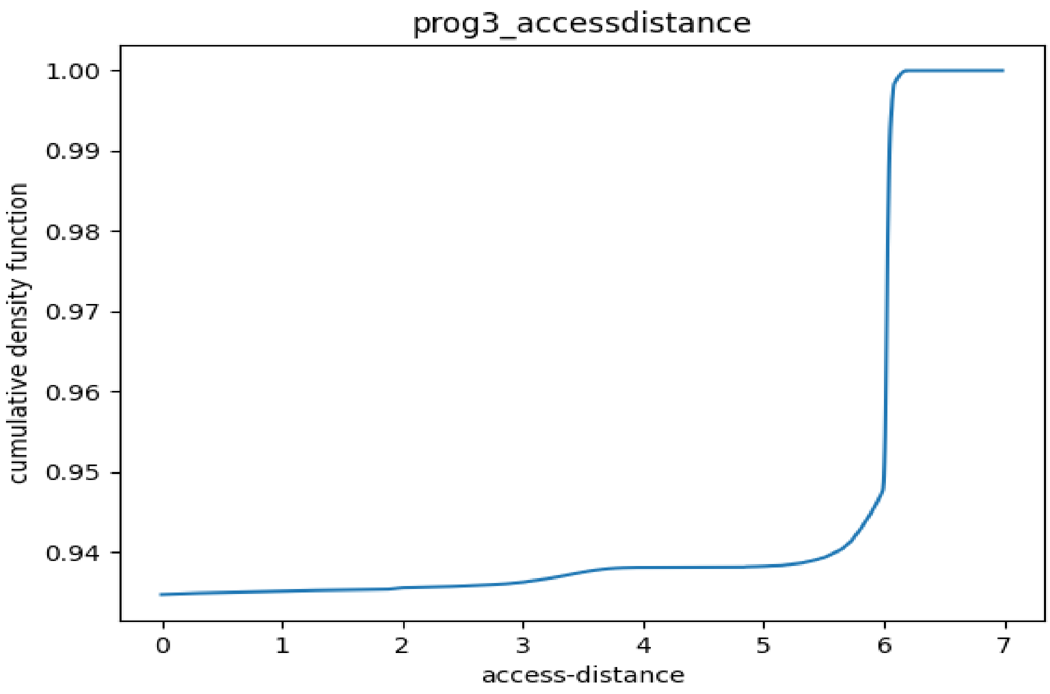

Figure 3: Cumulative Density Function Prog3

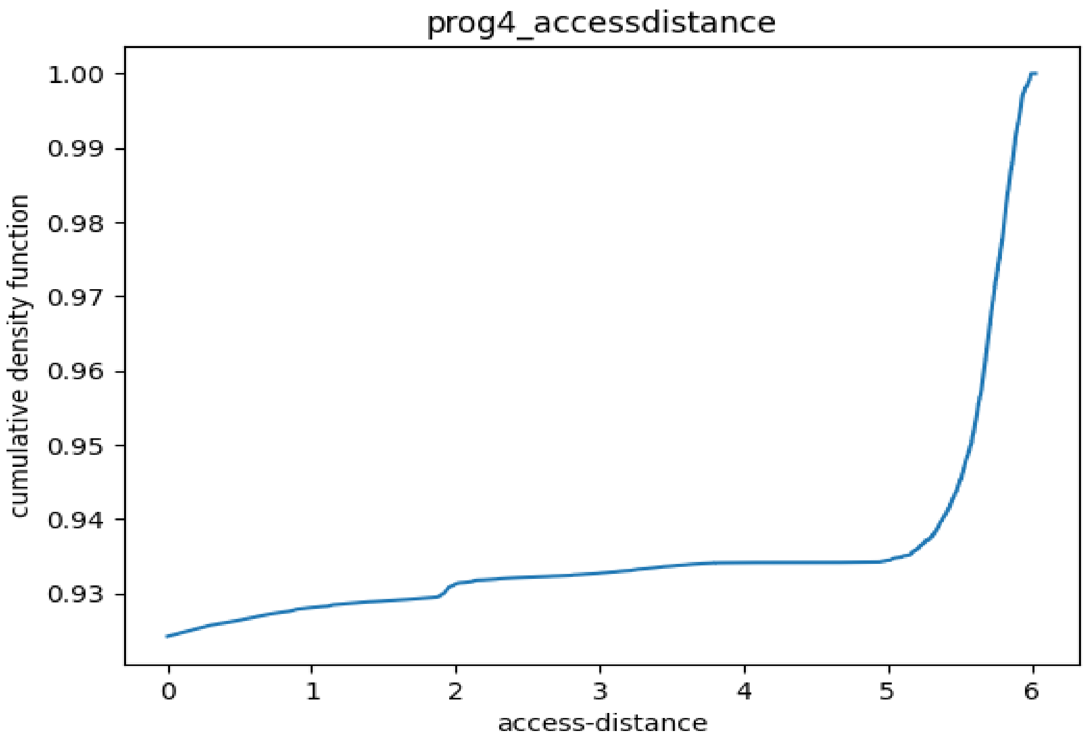

Figure 4: Cumulative Density Function Prog4

Observation:

The access distance which is considered as an approximate measure of the reuse distance can be used to evaluate the cache capacity needed for maximizing the cache hits. Figure 1 (CDF plot for prog1) shows that above 90% of the access distances are within a distance of 1 and 100. Figure 2, 3 and 4 for prog 2, 3 and 4 respectively shows that over 90% of the access distances are within a distance of 1 and 10. Thus, all the four parallel programs: prog1, prog2, prog3, prog4 exhibit a good temporal locality. Since, memory blocks are reused frequently, with an introduction of an LRU cache, a large percentage of the accesses (depends on cache capacity) can result in cache hits.

Problem 3:

The access trace is passed through a 2MB, 16-way set-associative, 64 block-sized single level LRU cache, and the hits and misses of each program trace is listed in the Table 2 below. The miss trace is collected and the cumulative density function is plotted for the access distances of the cache miss trace. The plots are shown in figures 5,6,7,8 for prog1, prog2, prog3, prog4 respectively. Now we can find out the number of cache lines in the cache by:

$$Numner\ of\ cache\ lines = \frac{Cache\ capacity}{Block\ size} = \frac{2^{21}}{2^{6}} = 32768 \approx 32\text{k}$$

Table 2: Hits and Misses

TRACES	HITS	MISSES
prog1	133838676	6686583
prog2	2305116	232099
prog3	8970351	635860
prog4	941135	123393

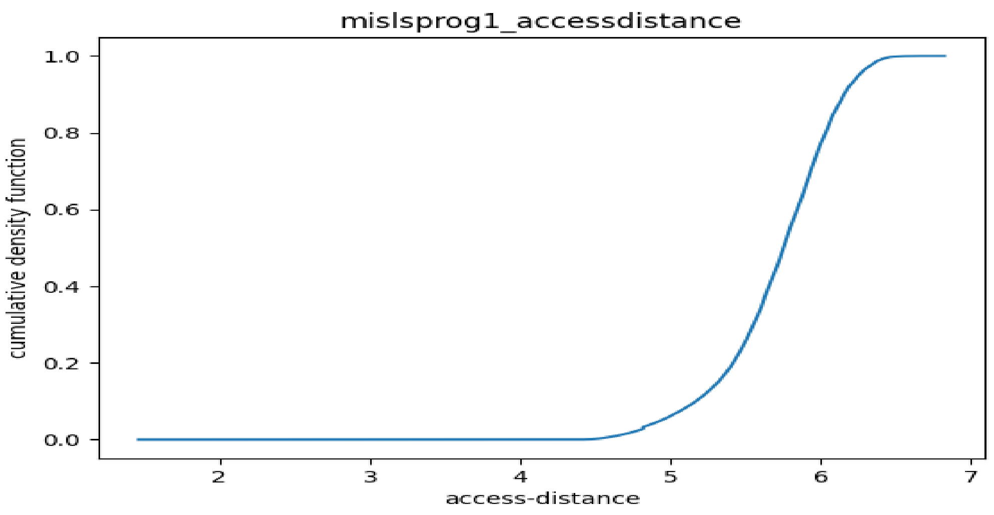

Figure 5: Miss Trace-Cumulative Density Function Prog1

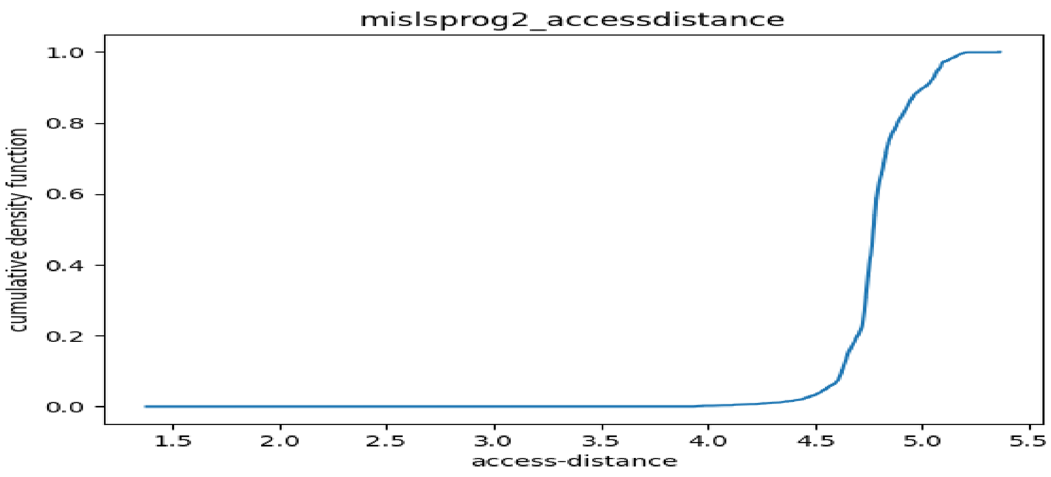

Figure 6: Miss Trace-Cumulative Density Function Prog2

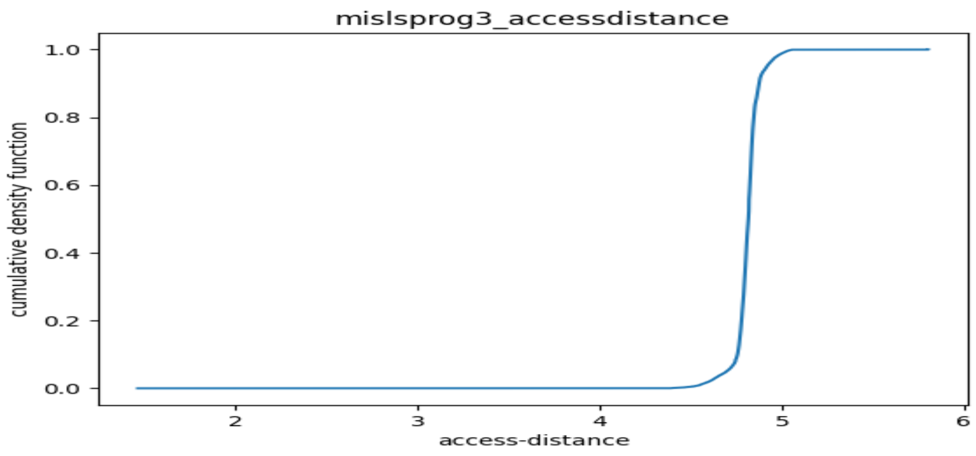

Figure 7: Miss Trace-Cumulative Density Function Prog3

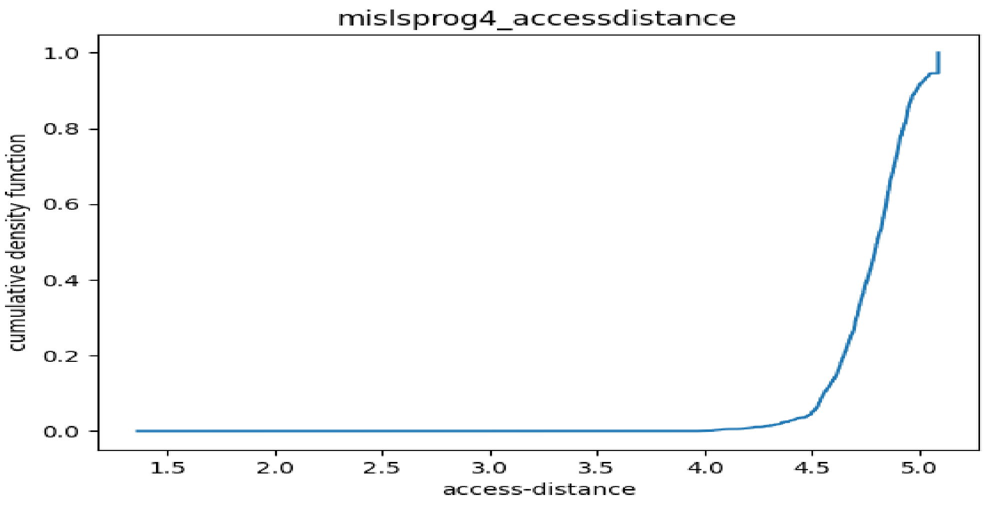

Figure 8: Miss Trace-Cumulative Density Function Prog4

Observation:

From Table 2, we observe that for each the parallel programs around 90% of the memory block accesses has resulted in cache hits with the introduction of a 2 MB LRU cache. This has given an effective improvement in performance by reducing the latency of memory access. The remaining 5%-8% accesses fall far apart in the trace, i.e. these accesses have access distances greater than $10^{4.5} \approx 32\mathrm{k}$, which is equal to the number of cache lines in the cache. This also explains the sudden spike in CDF we observe in Figure 5,6,7,8, when access distance is between 10^4 and 10^5 (almost at $10^{4.5}$), as majority of the accesses with access distance less than 32k will enjoy cache hits now, thus will not be included in the cache miss trace. Therefore, the CDF, increases rapidly, after access distances crosses 32k, and almost reaches 1, with a slight further increase of access distance. The memory accesses which miss in cache would require higher cache capacity, in order to enjoy cache hits. However, increasing the cache size to a large extent, might be detrimental to performance, as it leads to increasing access latency, therefore leading to poor performance.

Problem 4: Sharing Profile

The sharing profile of memory blocks among the eight threads are tabulated below for the 4 program traces

Table 3: Sharing Profile of Traces

Traces	Private	Two	Three	Four	Five	Six	Seven	Eight	Total Memory Blocks
prog1	434	70	1872	32455	143250	244970	173832	124529	721412
prog2	433	8262	16384	40957	4	0	1	12	66053
prog3	440	63	0	0	0	0	1	65547	66051
prog4	8622	57409	6	0	0	0	1	13	66051

Observation:

The sharing profile above gives the idea of number of memory blocks that are private or shared among the threads. The sharing profile helps in understanding of how intensive the sharing is among the threads. As we can observe from Table 3, in prog1, a large number of memory blocks are being shared by four, five, six, seven, eight threads. In prog 2, the majority of the sharing of memory blocks is limited to a maximum of 4 threads. In prog 3, most of the memory blocks are shared across all 8 threads. In prog4, we observe that, most of the memory blocks are either private to the threads, or are accessed by a maximum of two threads.

In thread level parallelism, the idea is to minimize the data dependence among the threads as much as possible, thus enabling them to run independently. The greater the number of memory blocks that are private, in turn improves the parallelism of the threads, as private cache access have lower latency and the accesses are far less complicated than accessing a shared block. Thus, prog4 has the best thread-level parallelism.

Note: All the data tabulated and plotted above may be subjected to little variation due to the parallel execution by 8 threads.