Platform-independent Data Locality Analysis to Predict Cache Performance on Abstract Hardware Platforms

Sonish Shrestha
sshrestha2@miners.utep.edu
University of Texas at El Paso

INTRODUCTION

The context of the research is a co-design project that has the goals of designing hardware systems to match application requirements and mapping applications to the hardware efficiently. To determine application requirements, we characterize the application using platform-independent locality metrics. Next we use locality data to predict cache performance of sequential versions of the application codes for various cache configurations. After using an analytical model to select a candidate set of cache configurations, we use architectural simulation to refine the selection for the target multicore systems.

GOAL

The overall context of the CoDAASH hardware-software co-design project has the goals of designing hardware to match the requirements of computational chemistry and physics algorithms important to the materials science problems of interest. To design the optimal cache configuration for a given algorithm is our main goal.

APPROACH

Data Access Pattern:

Our first step is to evaluate tools for obtaining platform-independent locality metrics and to validate the results. We are evaluating the PMaC locality measurement tool from SDSC and the MACPO data access analysis tool from TACC. We also use the PAPI hardware counter library to sanity check results returned by these tools. The PMaC locality measurement tool instruments the application using PEBIL in order to measure reuse distances and strides for data accesses. It computes reuse distances per basic block rather than by data structure, but we are working with SDSC on refining the tool to obtain per data structure metrics. The MACPO data access analysis tool reports reuse distance per non-scalar variable and also reports the strides with which these data structures are accessed. PAPI gives the information of different events related to cache memory like cache hits, cache misses etc.

To validate the results reported by each tool, we wrote simple matrix and blocked matrix multiplication benchmark codes for which we know the expected reuse distances and strides. After manipulating the results to compensate for the different ways in which the tools work, we found discrepancies between expected and actual results that are currently being fixed by the tool developers. We have the tool from SDSC now fixed and we got some initial results for matrix multiplication(simple and blocked).

Our second step is to use locality data to predict cache performance of sequential versions of the applications for various cache configurations. The application code that we are using initially is the LULESH benchmark, which serves as a proxy for full shock physics applications CTH and ALEGRA. We can use a straightforward analytical model to predict cache misses for a fully associative cache, and we can use a probabilistic model to predict cache misses for a set-associative cache. We got some initial results for LULESH benchmark too.

Simulators:

Our applications will need to need to run in parallel mode to scale to realistic problem sizes. Predicting cache behavior for a thread-parallel program running on a multicore system is much more complicated, and our predictions will be only approximate. To accurately evaluate and select optimal cache configurations without building the actual hardware, we make use of architectural simulators. Simulators we are evaluating include MacSim, SST, gem5, as well as GPGPU simulators.

RESULTS

Results for cache configuration(Intel Nehalem-EP) of L1 cache size= 32KB ,L1 cache line size=64, L1 Associativity= lru , L2 cache size=256KB ,L2 cache line size= 64 , L2 Associativity=lru , L3 cache size=2048KB, L3 cache line size= 64,L3 Associativity=lru

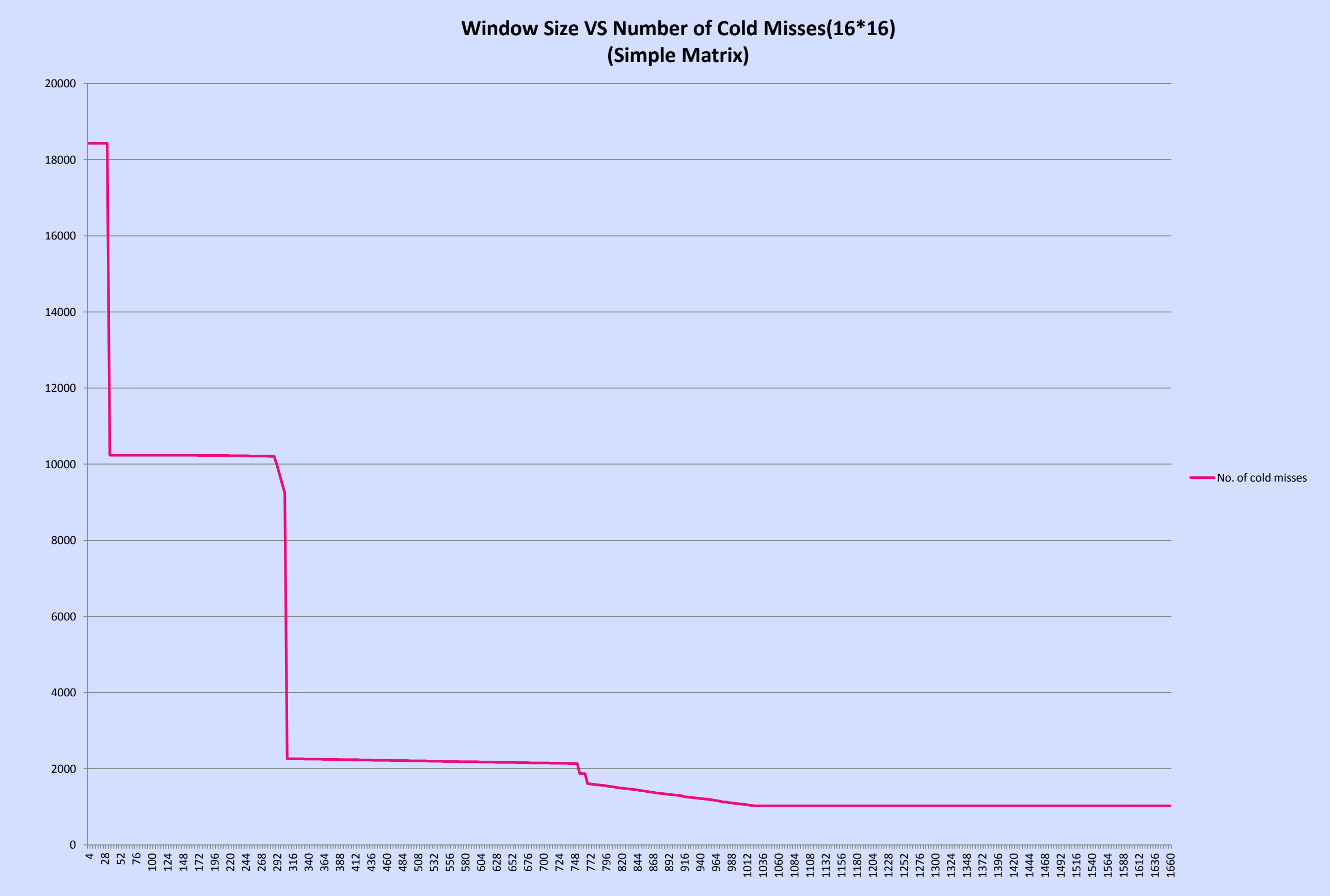


Figure1: In the above graph look-back window (window size) is 1024 at which the cold misses settles meaning all subsequent accesses to A, B, C and D arrays(4*16*16=1024) in the main matrix multiplication routine will already be in the 1024 window of addresses .Therefore, no increase in the cold misses. It does not matter whether you increase the look-back window beyond 1024 for the cold misses (cold miss is a miss that occurs when you see a memory reference for the first time).

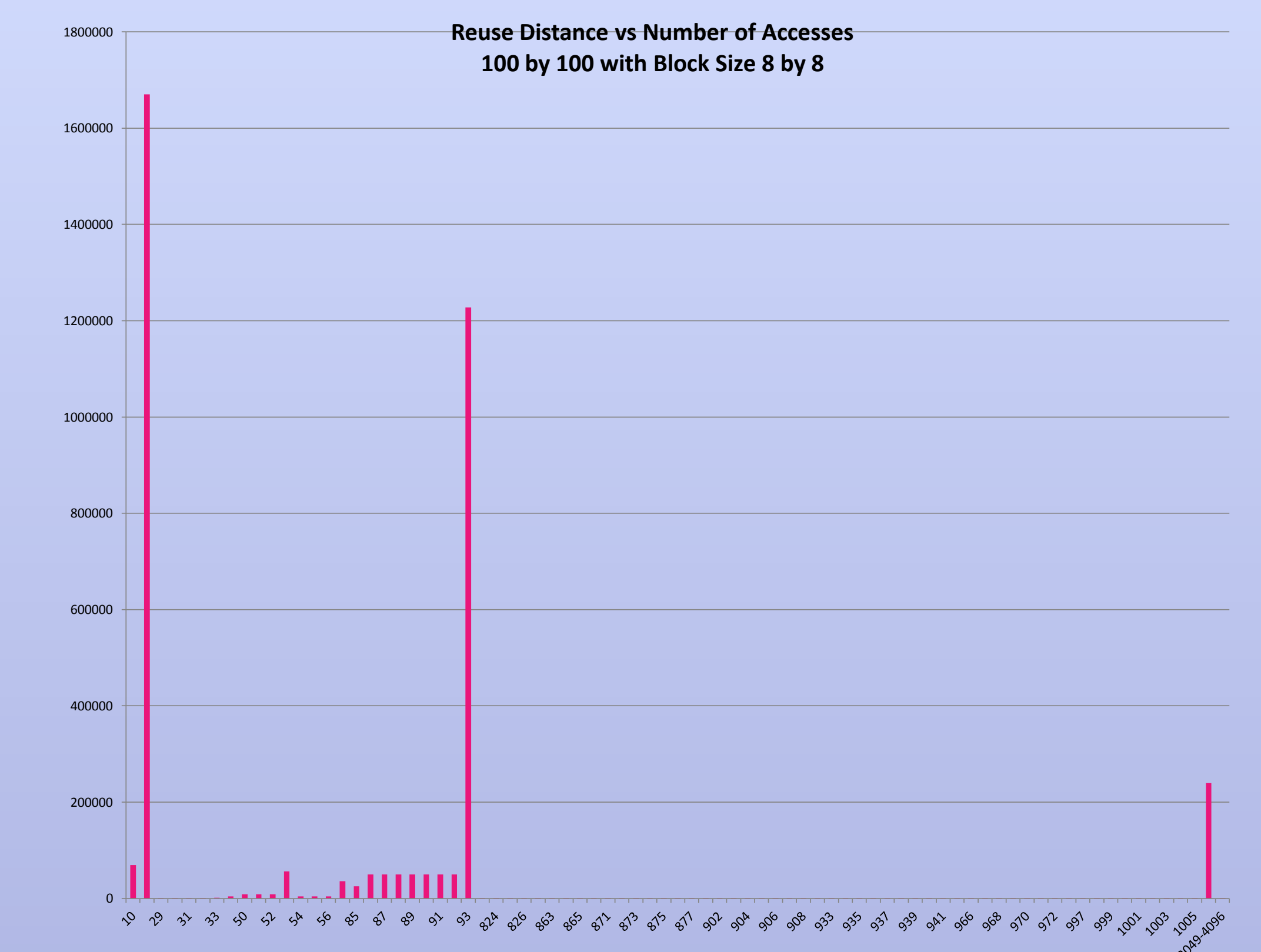


Figure2

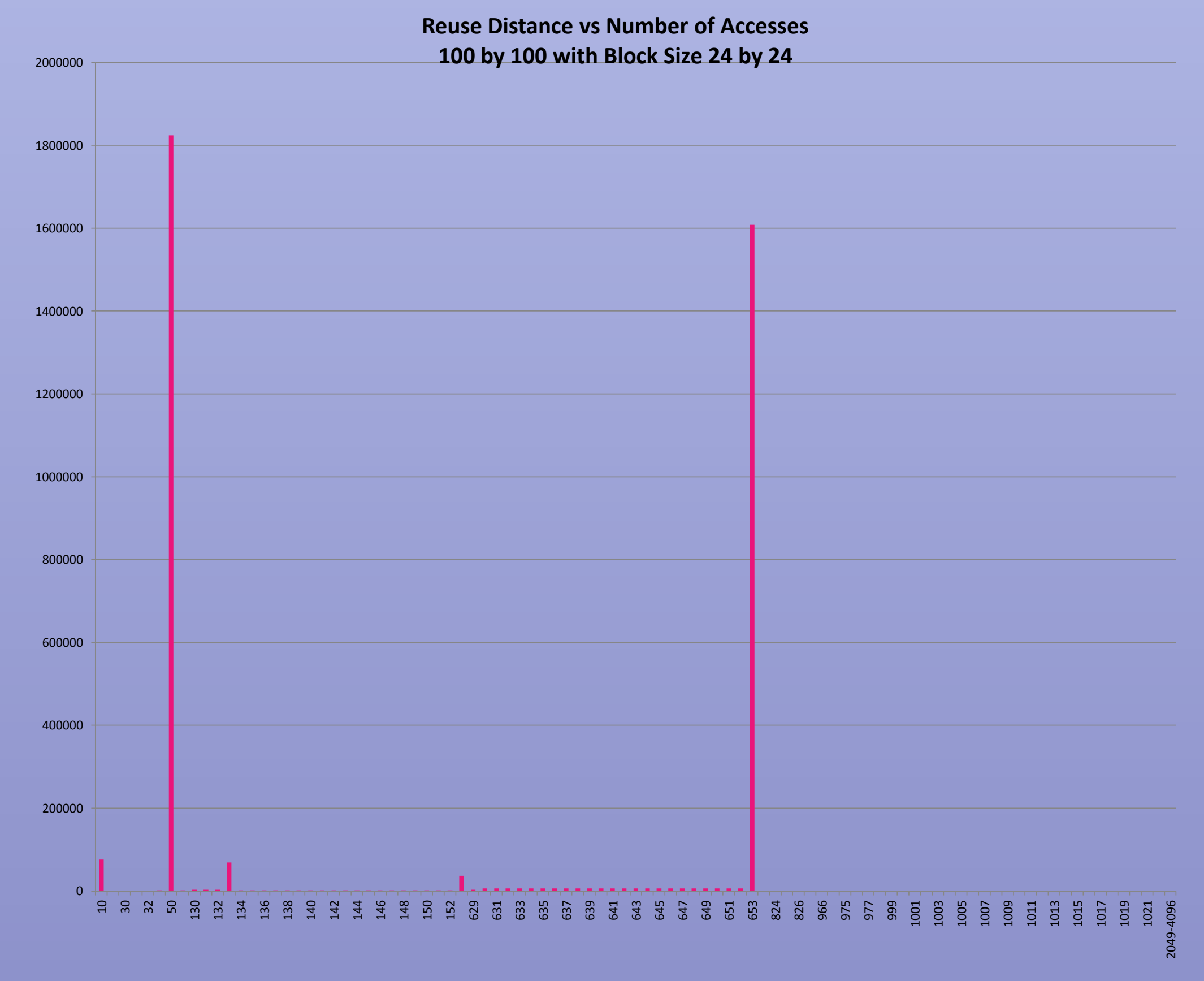
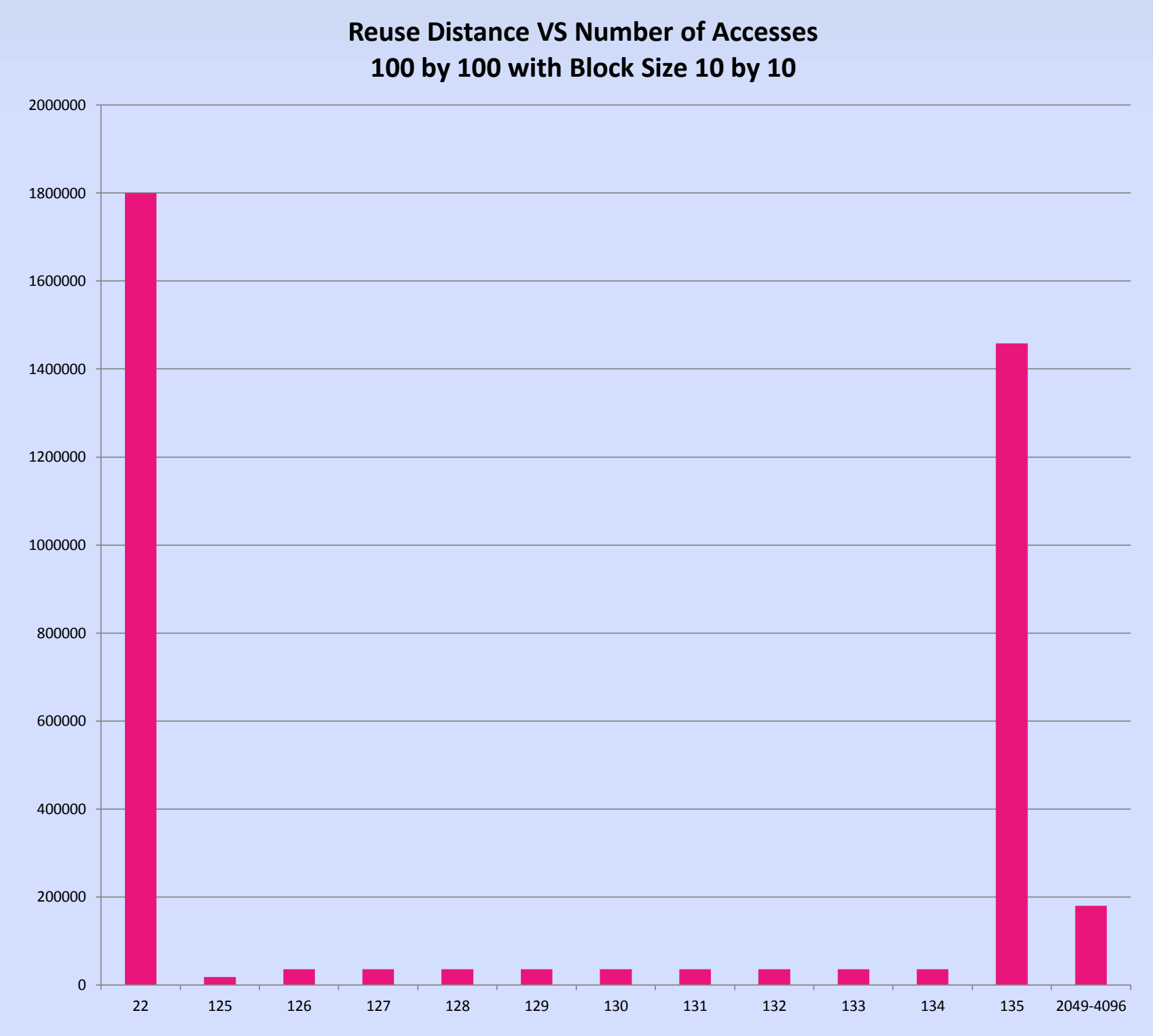


Figure3



Matrix Multiplication of 100 by 100 with the block size of 10 by 10 (Part of Sample File)

Window size=3000, Bin size=1024

		Number of Accesses	Number of Cold Misses
REUSEID	4222176190398464	4000000	219980

Lower Limit of Bin	Upper Limit of Bin	Number of Accesses
22	22	1800000
125	125	18000
126	126	36000
127	127	36000
128	128	36000
129	129	36000
130	130	36000
131	131	36000
132	132	36000
133	133	36000
134	134	36000
135	135	1458000
2049	4096	180020

Fig5: Overview of the Data generated and Plot

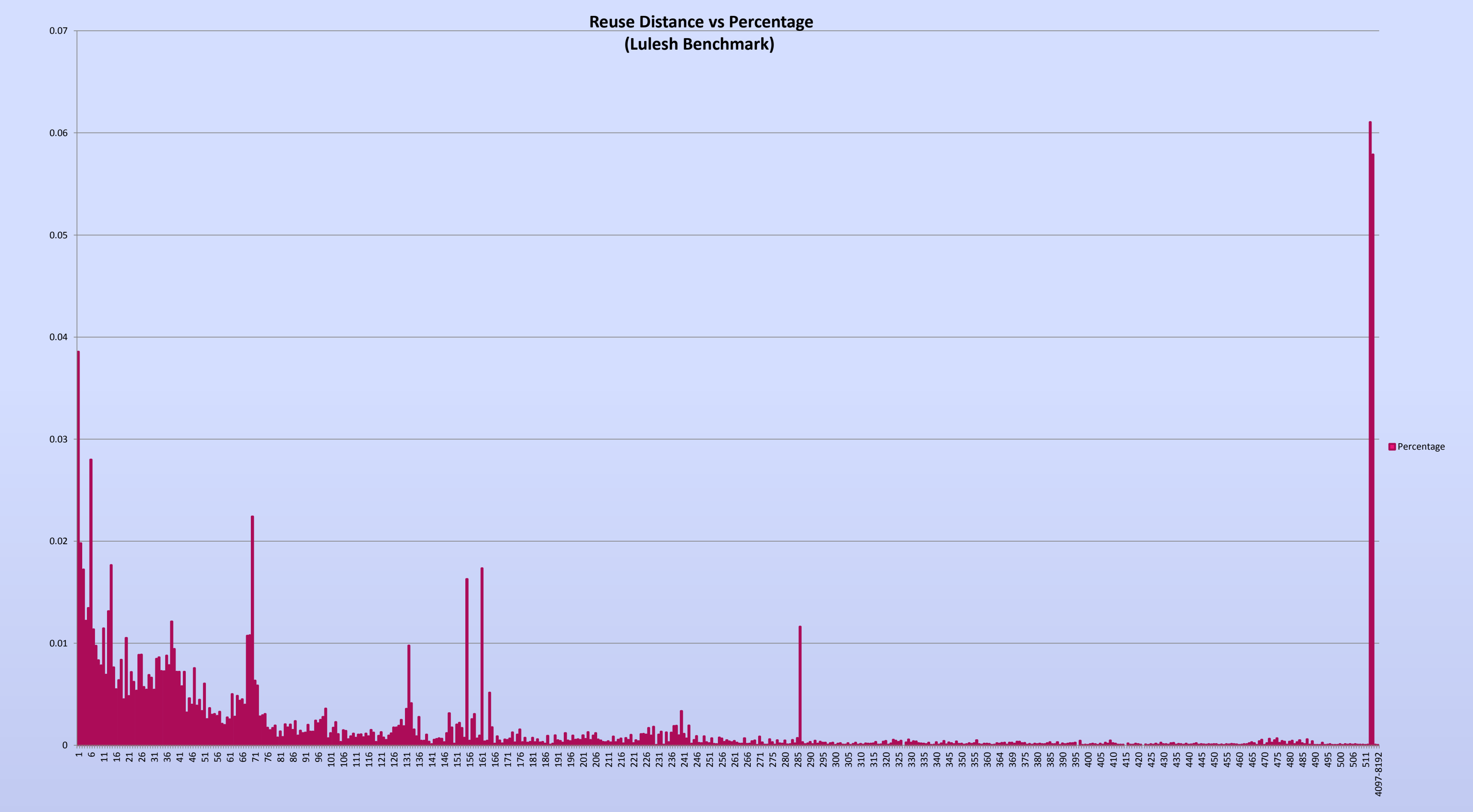


Figure6: LULESH BENCHMARK: Above graph presents the graph of percentage of number of accesses vs reuse distance for the window size 23900 and bin size 512. The bar indicates the percentage of accesses which have same reuse distance.

Figure 2,3,4 : The three graphs compare changes in reuse distances for various block sizes for the same matrix multiplication size 100*100. The graphs show that for a given block size, many accesses have the same reuse distance and that the specific distances change with change in block size. The bar showing the reuse distance at which most of the accesses occur shifts toward the right as the block size increases. This is because as the block size increases, the reuse distance also increases. These results validate our expected results for the SDSC tool.

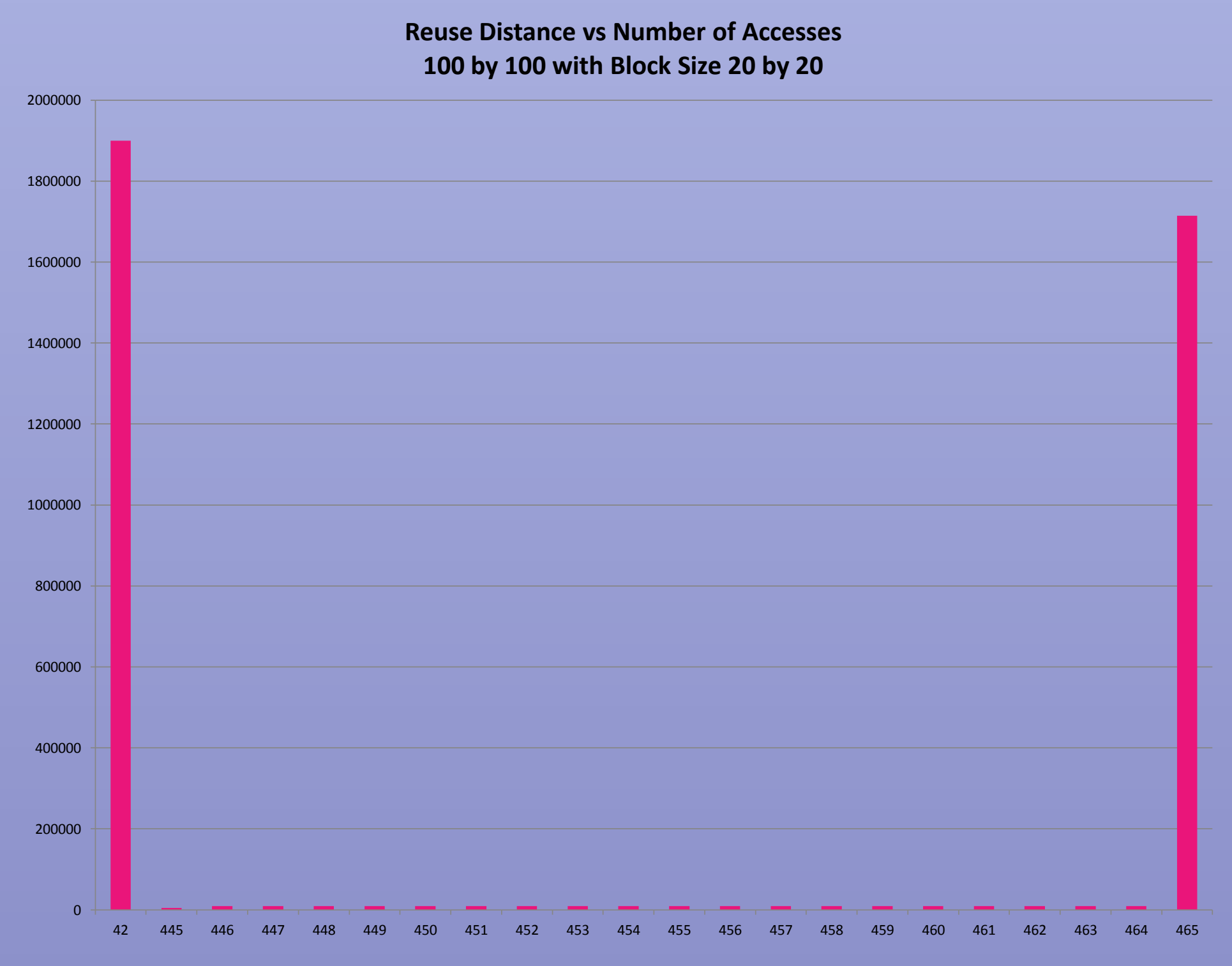


Figure4

CONCLUSION

In the poster, we report the results of our evaluations of locality measurement tool from SDSC. We also describe initial results from our characterization of the LULESH benchmark. Our next step will be to use this characterization to predict cache performance for different cache configurations.

ACKNOWLEDGEMENT

This material is based upon work supported by the Air Force Office of Scientific Research under AFOSR Award No. FA9550-12-1-0476