Project III

Neil P. Parmar
Dept. of Electrical Engineering and Computer Science
Oregon State University, Corvallis, USA
parmarn@oregonstate.edu

I. Introduction

The particular project's primary purpose is to understand false-sharing while using Fix#1 and Fix#2 with the available number of cache lines. This project report is categorized into System Configurations and Load, Various Implementation specifications, Performance Evaluation and Analysis and conclusion.

II. System Configurations & Load

This project had been implemented using the CentOS operating System Release 6.7, at OSU Computer labs which has about 8 CPU cores. Fig 1 below shows the CPU average load for 1, 5 and 15 minutes to be 0.71, 0.17 and 0.66 respectively.

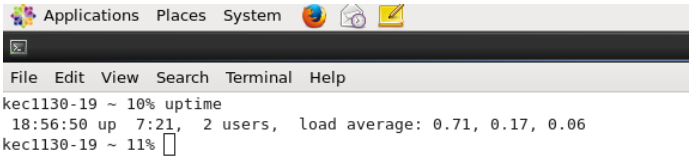

Fig 1. System configurations – System Load

III. Implementation Specifications

The implementation code is contained in the file Project3_main.cpp, where the false sharing fixes in references to Fix#1 and Fix#2 has been implemented. The computation is shown in the file FinalImpContent.txt file.

Moreover, additional computations with rabbit and had been performed for the project example, however, the results tested did not produce satiable output and therefore these computations have not been recorded in this report. But one could find all the computation details with rabbit in the files "Tmp.txt". Kindly ignore any other files as they have been recorded just for the purpose of reference and analysis with rabbit.

IV. Performance Evaluation & Analysis

TABLE I. PERFORMANCE

		Number of Pads															
	Threads	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Fix #1	1	420.16	419.98	417.9	420.26	419.02	418.16	419.67	419.8	417.65	418.58	417.48	418.41	415.36	420.36	419.31	420.45
	2	129.59	341.34	377.27	384.25	430.32	477.65	473.9	824.76	830.86	827.96	830.43	825.01	823.26	828.61	831.33	834.78
	3	331.51	362.99	391.98	830.85	377.26	488.77	465.59	834.88	833.07	830.26	829.27	826.78	823.31	833.23	829.64	833.78
	4	227.58	225.32	297.19	401.47	305.29	373.93	373.6	508.3	724.38	723.59	713.85	662.35	679.03	1587.87	1579.52	1587.34
Fix#2	1	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72	419.72
	2	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27	829.27
	3	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43	834.43
	4	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58	1590.58

Firstly, Table I is categorized into two Fixes. Fix #1 represents the performance for number of threads being 1,2,3, and 4, and the various NUMPADS (Number of Padding) which in the particular case is 0-15. Fix #2 is represented with respect to the Number of Threads only, as Fix #2 observes a constant behavior and does not require padding as it uses a local variable.

The graph in Fig 3 shows Number of Pads plotted with respect to the performance. Also comments for Fix#2 has been plotted specifically on the graph in order to differentiate the particular from Fix#1.

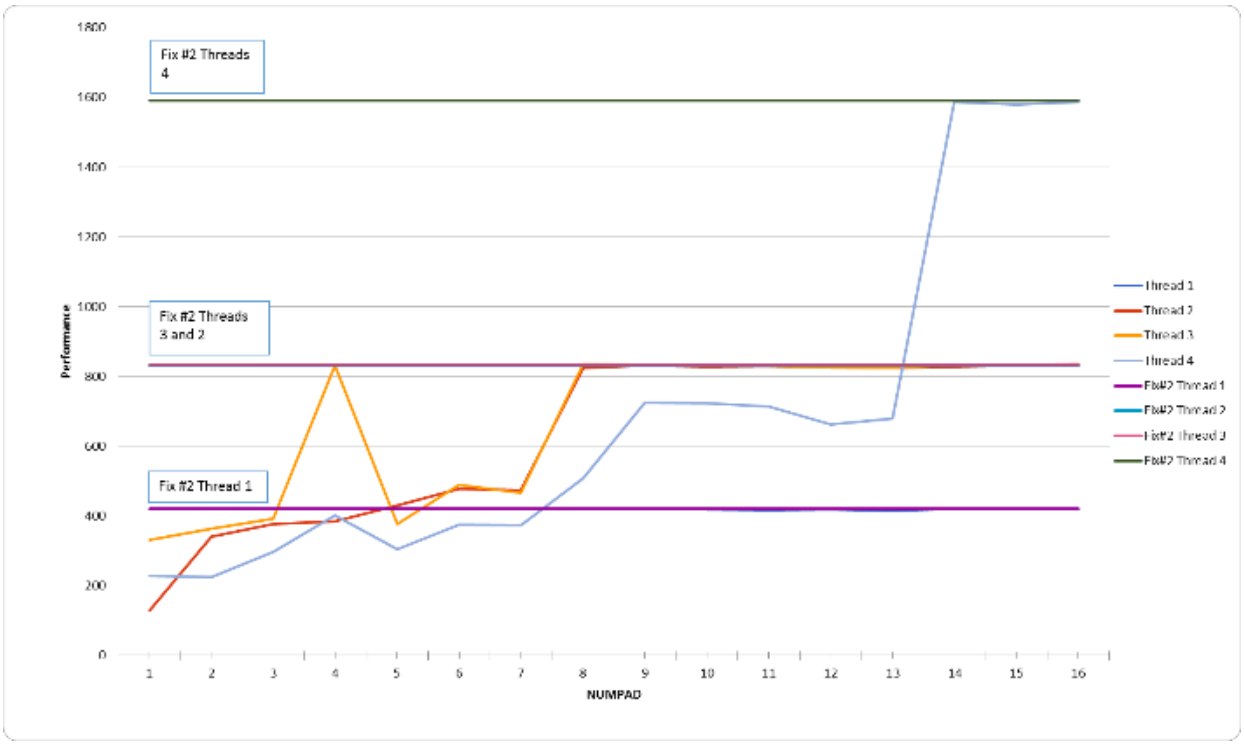

Fig 3. Performance

As can be seen from Figure 3, the performance for Fix#2 is comparatively better and shows a constant performance without having to observe the number of padding. While evaluating performance with respect to Fix #1 one could see corresponding irregular spikes, which may be observed due to the system load at the particular time period. Fix#1 while

using the NUMPAD as 15 observes the best performance with number of threads as 4. The similar performance is observed for Fix #2 with 4 threads.

The Fig 4, 5 and 6 shows the transpose of the graph plotted in Fig 3. The transpose has been distributed in the particular manner for better visual understanding. In the Fig 4 below, he performance is plotted with respect to the number of threads using Fix#1. Here as can be seen from the figure, here the performance is better with respect to the NUMPAD 15 as there is more padding and therefore lesser false sharing. However, with NUMPAD 0, with any number of threads the performance is unstable and low. The performance almost remains low but constant for the NUMPADs from 2 -10, while observing constant and steady decrease (after reaching a particular peak), as can be seen in the graph below.

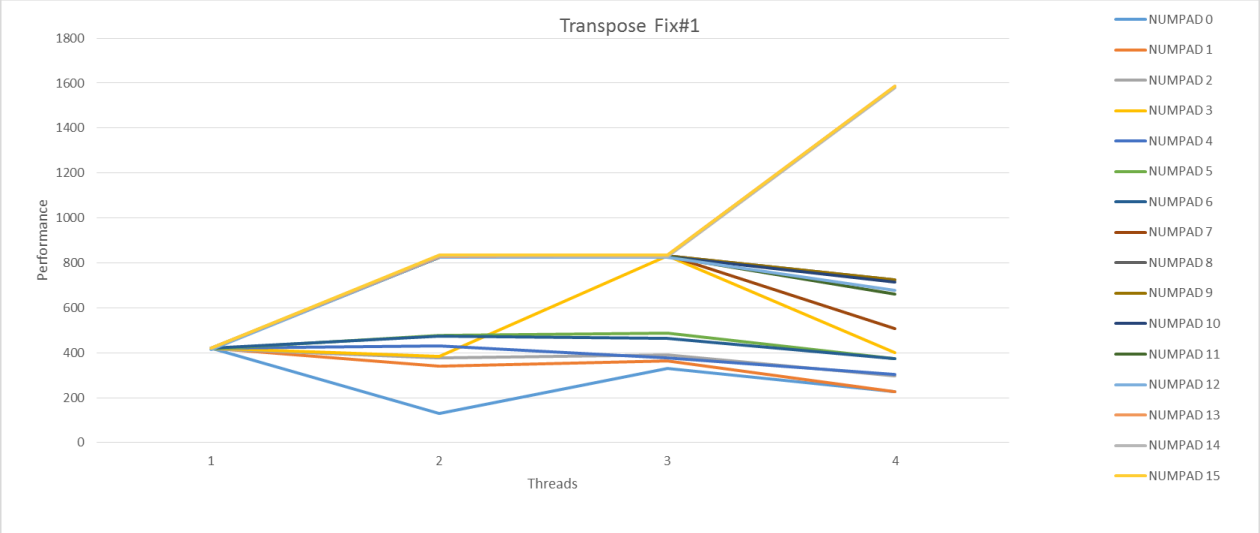

Fig 4. Transpose for Performance Fix#1

However, with Fix#2, the performance almost remains constant with any number of threads as there is no available padding. In this particular scenario, Fix#2 shows a better and steady performance.

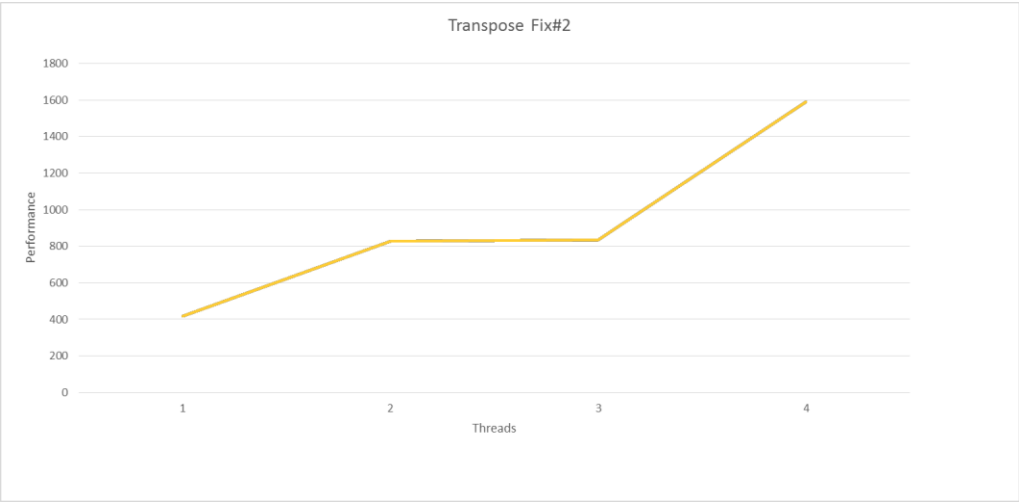

Fig 5. Transpose for Performance Fix#2

The figure below (Fig 6) represents the graphical performance for both Fix#1 and Fix#2. Here the horizontal axis represents the performance is plotted with a set of 1-4 threads which are combined on the same graph i.e., 1-4 for fix #1 and 1-4 for fix#2.

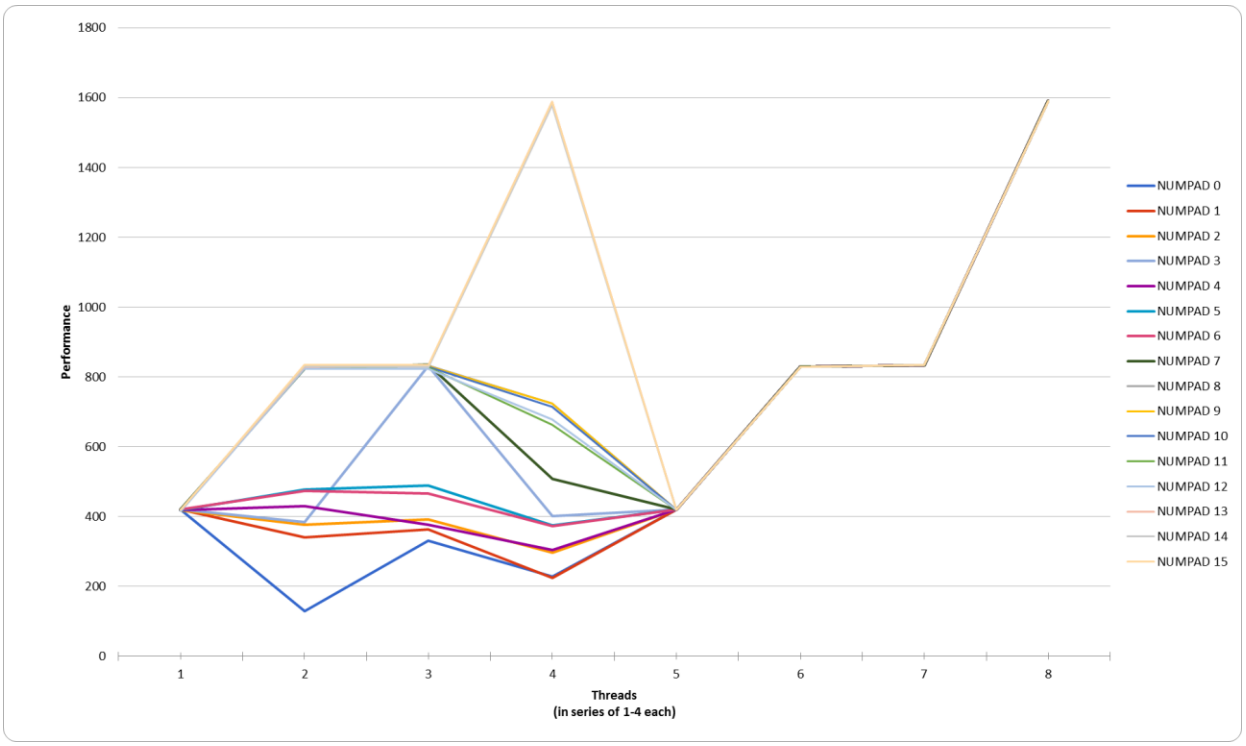

Fig 6. Transpose for Performance Combined Fix#1 and Fix#2

V. Conclusion

In this particular project scenario, better performance is observed using Fix#2 with the number of threads being 4. Padding does not considerably solve the performance in the particular scenario. Moreover, padding gives a very unsteady terrace results, except while using thread 1 (which has a constant output). Also, upon increasing the number of pads, the performance considerably rises as more space is available in the cache lines while using higher number of threads.