Detection of False Sharing Using Machine Learning

By

Sanath Jayasena*, Saman Amarasinghe**, Asanka Abeyweera*, Gayashan Amarasinghe*, Himeshi De Silva*, Sunimal Rathnayake*, Xiaoqiao Meng#, Yanbin Liu#

*Dept of Computer Science & Engineering, University of Moratuwa, Sri Lanka

**Computer Science & Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, USA, #IBM Research, Yorktown Heights, New York, USA

A PAPER REVIEW

Submitted to

Oregon State University

As part of assignment

For course Introduction to Parallel Programming Sp'19

Reviewed on June 10, 2019

By Aashwin Vats

About the paper:

This paper was published in the Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13) November 1, 2013. Twenty-four other papers in similar research area have cited it.

The paper, as the title suggests tries to address the issue of performance bugs in parallel applications by targeting a major class of bugs, false sharing. The paper proposes an efficient and effective approach to detect the false sharing based on machine learning. The implemented algorithm detects published false sharing regions in the experiments with benchmarks with a performance penalty of less than 2%. The detection of false sharing is challenging, as it doesn't change the semantics of the program. This paper speaks about how theses challenges were addressed with the use of machine learning.

About the authors:

The authors of this paper belong to three different organizations, as listed.

Dept. of CS & Engg, University of Moratuwa, Sri Lanka: Sanath Jayasena, is the Associate Professor in the Dept. of Computer Science in the university. His research interests include High performance computing, parallel computing, ML and NLP). He was the supervisor for this research project. **Asanka Abeyweera and Gayashan Amarasinghe** contributed to this paper as their final year project, having similar research interests. Other contributors, **Himeshi De Silva and Sunimal Rathnayake** are currently computer science PhD candidates at the National University of Singapore.

Massachusetts Institute of Technology, Cambridge, USA: Saman Amarasinghe: He leads the Commit compiler research group in MIT's Computer Science & Artificial Intelligence Laboratory (CSAIL), which focuses on programming languages and compilers that maximize application performance on modern computing platforms. He was the supervisor for this research project.

IBM Research, Yorktown Heights, New York, USA: Xiaoqiao Meng: Currently working as Engineer Manager at Facebook. **Yanbin Liu:** Currently, software architect at Yale University, previously Advisory Soft Engineer at IBM Watson Research.

The what-why-how-wow:

False sharing is common issue that hurts the performance and subsequent speedup in a parallel execution. It occurs when the threads running on different cores with local caches modify unshared data that happen to occupy same cache line.

There are available methods that help resolve the same issue. However, they are expensive and are seldom used in practice. False sharing can't be detected by application analysis or localized analysis within a single core (as it would require multiple cores to access different parts of cache line). False sharing problem worsens with multi-threaded versions and the recent works have tried techniques that trace data movement across multiple cores. This, however, requires excessive data gathering and heavy instrumentation incurring a big overhead. They are also limited in application due to dependency on various libraries.

The approach that this paper proposes looks for telltale signs created by false sharing; induced memory access patterns on multiple cores. A pattern is detected by looking at the performance event counts of cores by using supervised learning and training the classifier with sample kernels-with and without false sharing. The trained classifier then analyzes patterns in other programs.

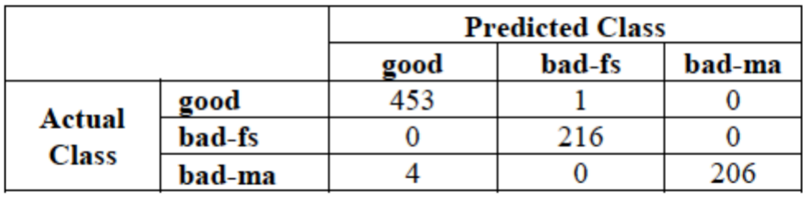
The proposed approach relies on hardware performance event counts collected from running programs that help understand how the hardware is being utilized by program and provide potential hints on issues with performance. This information is tightly coupled with system's architecture and as a result the data is difficult to handle and process for humans. Hence, the authors resorted to machine learning techniques for analysis of this data.

With this, the machine classifier was trained from a set of kernels, while running each with and without false sharing turning it on or off. To distinguish the potential problems from others, the inefficient form of memory access were included in these programs. These can be classified into 3 modes: good- no false sharing, no bad memory access; bad- false sharing, ‘bad-fs’; bad- inefficient memory access, ‘bad-ma’. Hence the classification became a three-way classification, resulting iff false sharing was considered. After running the mini-programs in all possible modes, each instance was labeled as good, bad-fs or bad-ma. The classifier was trained based on these labels as training data.

The two sets of mini-programs used were: multi-threaded program set and sequential program set (linear resulting in ‘good’ mode and strided-access resulting in ‘bad-ma’ mode) and within these sets different versions perform the same computation while the way data in memory is accessed is different.

These mini programs are used to identify a set of relevant events from the candidate list of performance events for the hardware platform. Adding Instructions_Returned event to the existing list of 15 selected events did required normalization. Smaller amount of events were considered as they were thought to be desirable for better accuracy of results. However, we know that large set of relevant performance events is potentially desirable from a ML point of view.

The training dataset consisted of 880 instances: A+B, where A was the set of multi-threaded mini-programs and B was the set of training data from sequential mini-programs. Decision tree classifier was constructed with 6 leaves and 11 nodes. Below is the confusion matrix for training data.



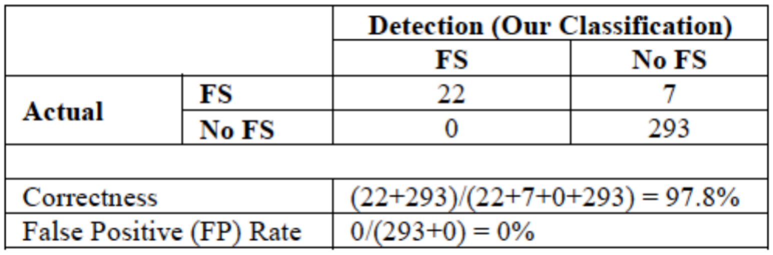
		Predicted Class		
		good	**bad-fs**	**bad-ma**
Actual Class	**good**	453	1	0
	bad-fs	0	216	0
	bad-ma	4	0	206

Experiments performed for detection of false sharing:

The trained classifier was applied to the PARSEC and Phoenix benchmark programs.

The results were compared with the two most recent works referenced in the paper. The programs were ran with provided input sets, each with different no. of threads and compiler optimizations.

Two very bold claims were made after evidences from these two benchmarks. There were zero false positives, as the trained classifier did not incorrectly classify any program as having false sharing. The instrumentation cost was found to be minimal where performance penalty was less than 2%. The approach is easily portable and easy to apply without specialized tools, hence a practical method for detection of false sharing. Following is the result of performance of selection of false sharing, based on results from verification with Phoenix and PARSEC benchmarks:



		Detection (Our Classification)	
		FS	**No FS**
Actual	**FS**	22	7
	No FS	0	293
Correctness		(22+293)/(22+7+0+293) = 97.8%	
False Positive (FP) Rate		0/(293+0) = 0%	

Conclusions from the paper:

The proposed approach to detect false sharing is efficient and effective, based on machine learning to detect false sharing. It is portable provided performance event counts can be collected and is independent of specialized tools or access to source code. The performance overhead is less than 2% and has been applied on well-known benchmarks with an observation of 0 false positives. With respect to the recent works referenced in the paper, the proposed approach has been claimed to be lightweight and have a better accuracy.

New insights gained from the paper:

Apart from what was covered in the course of intro to parallel programming, by reading this paper I gained a more detailed insight on how false sharing adversely impacts the performance. The events of Intel micro-architectures Nehalem, Westmere and Sandy Bridege (referenced in paper) observed that there is no single event, or even a small subset of events that can indicate the presence of false sharing. However, the authors worked impressively to leverage the performance event counts data in order to make predictions on the detection of false sharing. Also, I learned how overwhelming the data could be for human processing. The different kind of mini-programs/kernels chosen for the training dataset had their own way of accessing data from memory and the classifier built was trained on a normalized data. Furthermore, I learnt more about the existing benchmarks present in the field while the authors used them to assess their trained classifiers and performed a detailed analysis on the acquired results. A more detailed comparison was done later in comparison to two recent approached referenced in the paper comparing results with the same benchmarks.

Discussion of shortcomings in the author's methods/conclusions:

Machine learning models usually have a much larger set of relevant data however in this paper, in order to get desirable accuracy the number of events chosen were merely 15. This is not a desirable dataset to make predictions from a machine learning point of view. This is one of the concerns I had while going through the methodology of their approach.

Future Scope:

There are a few directions to where this project can be extended. One of the suggestions in the paper is to detect false sharing in shorter time slices. Or, detecting it at function-level, unlike the work done in this paper that considers whole duration of the program. The proposed approach can further be tested on more real applications to find out the effectiveness and its dependence on the number and types of performance events & mini-programs across different hardware platforms.