International Journal of Computer Information Systems and Industrial Management Applications (IJCISIM)
ISSN: 2150-7988 Vol.2 (2010), pp.212-221
http://www.mirlabs.org/ijcisim

Characterizing L2 Cache Behavior of Programs on Multi-core Processors: Regression Models and Their Transferability

Jitendra Kumar Rai,[1#§] Atul Negi,[2#] Rajeev Wankar,[3#] K. D. Nayak[4§]

ANURAG[§]
Hyderabad, India
jk.anurag@yahoo.com[1]
anuragdir@satyam.net.in[4]

Department of Computer & Information Sciences[#]
University of Hyderabad, Hyderabad, India
atulcs@uohyd.ernet.in[2]
wankarcs@uohyd.ernet.in[3]

Abstract

Contention for shared resources on multi-core processors has been a performance bottleneck. A solution to manage contention would be to apply knowledge about the shared resource utilization behavior of programs running on multi-core processors. In our previous work we used machine learning techniques to predict solo-run-L2-cache-stress, which can be utilized as a metric to characterize such behavior of programs.
In this study we investigate the transferability of trained regression models that estimate solo-run L2 cache stress of programs running on multi-core processors. Machine learning techniques were used to generate the trained regression models. Transferability of a regression model is the utility of a regression model trained on one architecture to predict the solo-run L2 cache stress on another architecture. The statistical methodology to assess model transferability is discussed. We observed that regression models trained on a given L2 cache architecture are reasonably transferable to another L2 cache architecture and vice versa.

1. Introduction

Multi-core processors generally have level-2 caches (L2 caches) which are shared between cores or hardware threads [1][2][3]. Contention for shared L2 cache between programs running on multi-core processors is one of the performance bottlenecks. The solutions proposed by researchers to reduce the contention for shared L2 caches on multi-core processors [4][5][6][7][8] [9][10] need to know about the L2 cache related characteristics of the programs running on a multi-core processor.

1.1 Metric for characterizing L2 cache behavior of programs

L2 cache miss rate i.e. the number of L2 cache misses per instruction retired, is one of the metric used to characterize the L2 cache behavior of a program. The observed L2 cache miss rate of a program, while it shares L2 cache with another program running on a different core, is different from its solo-run L2 cache miss rate. This is so because the observed L2 cache miss rate results from the interactions of the cache access patterns of both the programs. The solo-run L2 cache miss rate is the L2 cache miss rate observed for a program when it runs *alone* without sharing L2 cache with another program running on other core.

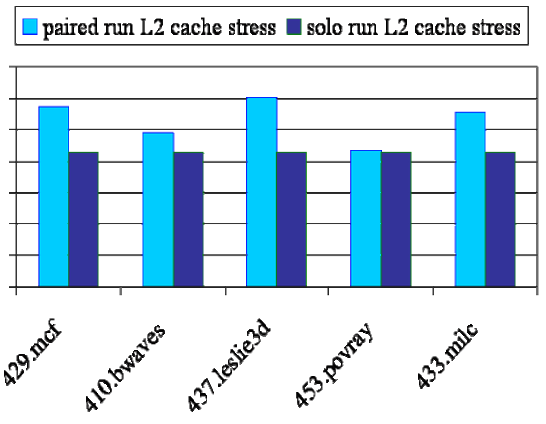

Figure1. Solo-run and paired-run L2 cache stress for 429.mcf on Intel Xeon X5482

We use solo-run L2 cache stress of a program to characterize its L2 cache behavior while running on multi-core processors. The solo-run L2 cache stress is the total number of L2 cache lines brought in due to miss and prefetch activities per Kilo (10^3) instructions

retired, when the program is running without sharing the L2 cache with programs running on other cores. The values the solo-run as well as paired-run L2 cache stress of 429.mcf (one of the SPEC cpu benchmark) measured on our Intel quad-core Xeon X5482 processor based experimental platform (described in section 4.1) are shown in fig. 1. The names along the horizontal axis are benchmark program names from SPEC cpu2006, with which the 429.mcf was paired. It can be seen that the paired run L2 cache stress for the 429.mcf differs between various runs based on co-runner benchmark.
An operating system scheduler can use the solo-run L2 cache stress of programs to intelligently schedule them to reduce the contention for shared L2 cache.

1.2 Consideration of the contention for other shared resources

Apart from last level (L2) caches, there are various other resources which are shared between processes running on a system based on multi-core processors. In a recent work [36] Sergey Zhuravlev et al observed that along with contention for shared last level cache, other factors like memory controller contention, memory bus contention and prefetching hardware contention also combine in complex ways to cause degradation in performance for processes running on multi-core processors. They performed experiments on two socket server with two Intel X5365 “Clovertown” quad-core processors. On that system the two sockets shared the memory controller hub, which includes memory controller. The four cores on each socket shared a Front Side Bus (FSB). Each pair of cores on a single socket shared last level (L2) cache.

On this system when the two processes run on different sockets, they contend for memory controller. When they run on the same socket, on the cores not sharing last level (L2) cache, they contend for Front Side Bus (FSB), in addition to memory controller. When the two processes run on the same socket, on the cores sharing last level (L2) cache, then they contend for all the four resources i.e. last level (L2) cache, prefetching unit, Front Side Bus (FSB) and the memory controller. In their work [36] Sergey Zhuravlev et al. used solo last level cache (LLC) miss rate as a metric to study the performance degradation due to contention for shared resources on multi-core processor.

The metric used in our study i.e. solo-run L2 cache stress includes the total number of L2 cache lines brought in due to miss as well as prefetch activities. It represents the amount of traffic happening between memory and last level (L2) cache of the processor. Thus by definition itself our metric takes into consideration the contention for all the four resources i.e. last level (L2) cache, prefetching unit, Front Side Bus (FSB) and the memory controller.

We measured the solo-run last level (L2) cache miss rate as well as solo-run last level (L2) cache miss and prefetch rate (i.e. solo-run L2 cache stress) for programs of SPEC cpu2006 suite on our Intel quad-core Xeon X5482 processor based experimental platform described in section 4.1. The values of the same for some of the SPEC cpu2006 benchmarks are shown in fig. 2, where names along the x-axis are benchmark program names. The large difference between the two indicates towards better suitability of solo-run last level (L2) cache miss and prefetch rate (i.e. solo-run L2 cache stress) as a metric to study the shared resource contention on multi-core processors, as it represents the total traffic happening between last level (L2) cache and memory.

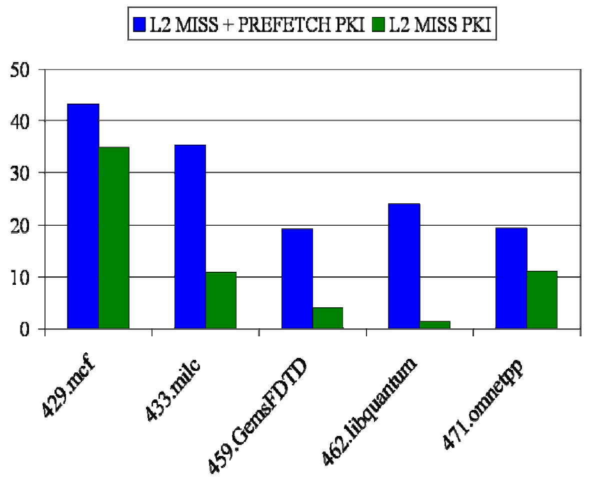

Figure2. L2 Miss+Prefetch and L2 Miss per Kilo Instructions retired on Intel Xeon X5482

1.3 Regression models and their transferability

In our previous work [11] we observed that the regression models generated by training machine learning algorithms can be used to predict solo-run L2 cache stress of programs running on multi-core processors. Transferability of a regression model means how useful is a regression model (which is trained on one scenario e.g. architecture) to predict the dependent variable i.e. solo-run L2 cache stress on another scenario (e.g. architecture). A regression model trained on architecture A is considered transferable to architecture B if it can be used to accurately predict the solo-run L2 cache stress on architecture B.

We plan to use machine learning to train the regression model offline and later on use the model online to predict solo-run L2 cache stress of running programs, to assist intelligent scheduling decisions on multi-core processors. Transferability of regression model is an important property for utilization of the model across various scenarios. It amortizes the efforts involved in training. Here scenario includes interactions between architecture and application with reference to L2 cache.

1.4 Organization of the paper

In this study we investigate the transferability of trained regression models for predicting the solo-run L2 cache stress. Run time data collected from hardware performance counters of Intel quad core Xeon X5482 and Intel dual core Core2 6300 processors were used to train the regression models. We used statistical methods to assess the transferability of generated regression model across quad core Xeon X5482 and dual core Core2 6300 processors.

Rest of the paper is organized as follows. Section 2 gives brief account of related work. In section 3 we give brief overview of the machine learning algorithms used. Section 4 gives brief of the experimental setup and data collection process used to train the models. In section 5 the statistical methodology adopted for assessing the transferability is described along with results. Finally in section 6 we conclude.

2. Related work

There are mainly two kinds of approaches to manage the contention for shared L2 caches on multi-core and multi-threaded processors: hardware based approach and software based approach.

The software based approach involve scheduling by operating systems [9][10] which is based on analytical models for cache behavior of running programs. Most of the hardware based approaches [4] [5] [6] [7] [8] rely on additional hardware support and analytical modeling to perform cache partitioning. Analytical cache models based on stack simulation [15] and probability [16] have been developed for chip multiprocessor systems. David Tam et al. [17] used memory access trace of running programs for getting L2 cache miss rate curves, which can be used for partitioning the cache. They used a Performance Monitoring Unit (PMU) feature called continuous data address sampling, available on IBM POWER5 processor only.

Neural networks have been used [18] by Richard M. et al. for workload characterization. They characterized a 3-tier web service in terms of functional characteristics of the application. Kumar and Negi [19] [20] used machine learning to characterize the workload and improve scheduling in Linux operating system. In their work various attributes from ELF executables and the previous execution history of the processes were used to characterize the workload. In [21] machine learning algorithms were used to study the performance in terms of Instructions Per Cycle (IPC) using the event data collected from hardware performance counters. In [22] model trees have been used in performance evaluation and in [23] the transferability of regression models is discussed to predict the performance across various work-loads in term of their IPC. T Ramazan B. et al. [24] proposed implementation of artificial neural networks based framework in hardware to coordinate the management of multiple interacting resources in chip multiprocessors.

The objective of our work differs from the previous work. We are characterizing the L2 cache behavior of programs on multi-core processors. The knowledge about the process characteristics in this respect can also be used for mitigating the contention for other shared resources on multi-core processors based systems. The focus of the present study is to investigate the transferability of trained regression models to predict shared L2 cache related behavior of programs across two multi-core processors having different L2 cache organization.

3. Machine learning algorithms used

Different machine learning algorithms correspond to different concept description spaces searched with different biases. Some problems are served well by different description languages and biases, while others are not served well or even served badly. This entails the study of various machine learning algorithms belonging to different families to check their efficacy to solve a given problem across various scenarios [31]. The machine learning algorithms used in the study are: Linear Regression (LR), Artificial Neural Networks (ANN), Model Trees (M5'), K-nearest neighbors classifier (IBK), KStar (K*) and Support Vector Machines (SVM).

Linear regression [31] performs least-squares linear regression.

Artificial neural networks [26] are based on the mechanism of co-operative processing of information, as done by neurons in the brain. In a multilayer neural network, there is an input layer, an output layer and a number of hidden layers. Each layer has a number of neurons (nodes) organized in it. The input layer takes the information to be processed as input. The first hidden layer takes the results from input layers as its inputs and forwards its results as inputs to the next layer. The output layer takes the results of the last hidden layer as inputs and produces the prediction result. In this study back-propagation was used to train feed-forward multilayer neural network.

K-nearest neighbors classifier (IBK) and KStar (K*) [26][27][28] are lazy or instance-based learning algorithms. In this case a new regression equation is fitted each time, when the model needs to predict on a new instance (i.e. a new query point).

Model trees are a kind of regression tree [14]. We chose M5' algorithm [12], which is an improved version of original M5 algorithm invented by Quinlan [13]. Model trees recursively partition the input space until the linear models at the leaf nodes can explain the remaining variability in those partitions.

Support Vector Machines (SVM) [29] tries to find instances that are at the boundary of the classes. These instances are called support vectors. Then it generates functions that discriminate those vectors as widely as possible. For training the support vector machine, a generalization of Sequential Minimal Optimization algorithm (SMO) by Shevade et al. [30] is used.

In this study we used weka-3.6.1 machine learning workbench [31]. We used default settings of most of the parameters to the above-mentioned machine learning algorithms in weka-3.6.1 except few changes described above.

4. Experimental setup

This section gives brief of the experimental setup and the methodology used to collect data to train the machine learning algorithms. The methodology used is similar to our pervious work [11].

The data were collected from hardware performance counters in two phase i.e. solo-run experiment and paired-run experiment. Hardware performance counters are special purpose registers provided on modern processors for measuring various performance events such as number of instructions retired, L2 cache misses etc.

4.1. Platforms

We performed experiments on two platforms. The first platform is a dual-socket DELL Precision T7400 workstation with two Intel quad-core Xeon X5482 processors (3.2 GHz) and 32 GB of memory. The other platform is a single socket DELL Precision 390 workstation with one Intel dual-core Core2 6300 processor (1.86 GHz) and 2 GB of memory. The operating system kernel on both platforms was Linux-2.6.28. For collecting data from the hardware performance counters [1] of both processors, we used perfmon [32] interface.

4.2. L2 Cache Organization on Xeon X5482 and Core2 6300

It is necessary to note the difference between the two processors with respect to their L2 cache organization to have a view of differences in the scenarios.

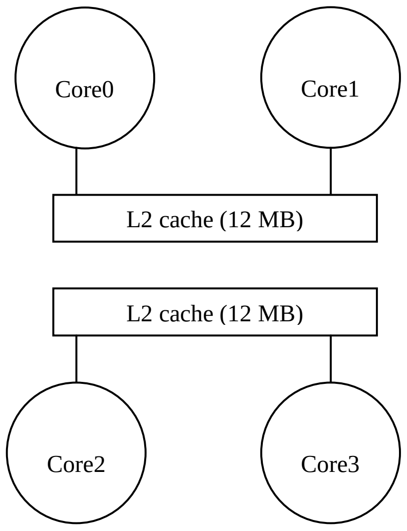

Figure3. L2 cache sharing on Intel Xeon X5482

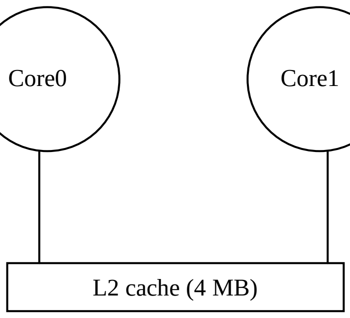

Figure4. L2 cache sharing on Intel Core2 6300

On both quad-core Xeon X5482 and dual-core Core2 6300 processor there is separate level-1 (L1) instruction and data cache, each of size 32KB per core.

Both of the processors have unified level-2 (L2) cache shared between two cores. The sharing of L2 caches between cores of Xeon X5482 and Core2 6300 processors are shown in fig. 3 and fig. 4 respectively. Table 1 shows the L2 cache related data [1] for both the processors.. There is difference in the L2 cache organization of the two processors in terms of cache size and ways of associativity.

Table 1. L2 cache related data of Xeon X5482 and Core2 6300

Processor	No of cores sharing L2 cache	Size (MB) shared between two cores	Ways of associativity
Xeon X5482	2	12	24
Core2 6300	2	4	8

4.3. Workload

We used benchmarks from SPEC cpu2006 suite [33]. The benchmarks suit consists of 12 integer programs and 17 floating point programs. We used reference inputs. With reference inputs the total workload consists of 35 integer programs and 20 floating point programs i.e. total 55 programs.

4.4. Solo-run-experiment

We ran each program from SPEC cpu2006 suite on a core and disallowed scheduling programs on other core sharing L2 cache with the previous core. The class variable solo-run L2 cache stress was calculated from hardware performance counter data collected for complete run of each program.

4.5. Paired-run-experiment

The programs from SPEC cpu2006 benchmark suite were run on the cores sharing L2 cache and the hardware performance counter data for each program were collected. Each of these programs takes different amount of time for completion. Hence we allowed one of the co-scheduled programs to run till completion. The other program was run in an infinite loop to execute it again as it finished and was stopped only as the first program finished. The attributes were generated from data collected from this phase.

With 55 programs, we have 55x55 (i.e. 3025) pairs of programs to run on the system. Completion of one run of 55 pairs takes about 7-10 hours on Xeon based and 10-15 hours on Core2 processor based platform.

4.6. Attributes and class variables

The events collected from hardware performance counters are:

INSTRUCTIONS_RETIRED
LAST_LEVEL_CACHE_REFERENCES
L2_LINES_IN__SELF__ANY

Out of afore mentioned events the second event LAST_LEVEL_CACHE_REFERENCES measures the number of references to L2 cache. While event L2_LINES_IN__SELF__ANY measures the total number of L2 cache lines brought in as a result of encountering L2 cache misses as well as prefetching activities.

If two programs p1 and p2 are running on two cores of Xeon X5482, and sharing the L2 cache. The attributes and class variables are generated in following manner:

The **first attribute** (p1_L2_REF_PKI) is L2 cache references per kilo instructions retired

p1_L2_REF_PKI =
(p1_LAST_LEVEL_CACHE_REFERENCES*1000)/
p1_INSTRUCTIONS_RETIRED

The **second attribute** (p1_L2_IN_PKI) is L2 cache lines brought in (due to miss and prefetch) per kilo instructions retired. It is indicative of the stress put by a program on L2 cache.

p1_L2_IN_PKI =
(p1_L2_LINES_IN__SELF__ANY*1000)/
p1_INSTRUCTIONS_RETIRED

The **third attribute** (p1_L2_IN_PK_REF) is L2 cache lines brought in (due to miss and prefetch) per kilo L2 cache lines referenced. It gives an indication of re-referencing characteristic of the program. Lower value of this attribute indicate higher tendency of the program to re-reference the previously referenced L2 cache lines.

p1_L2_IN_PK_REF =
(p1_L2_LINES_IN__SELF__ANY*1000)/
p1_LAST_LEVEL_CACHE_REFERENCES

The **fourth attribute** (p1_L2_FO) is the fractional L2 cache occupancy of a program. It gives a ruff estimate of fraction of space occupied by a program in L2 cache, while sharing it with other program.

p1_L2_FO=
p1_L2_LINES_IN__SELF__ANY/
(p1_L2_LINES_IN__SELF__ANY+
p2_L2_LINES_IN__SELF__ANY)

The **class variable** to be predicted is “solo-run L2 cache stress”. It is similar as second attribute i.e. L2 cache lines brought in (due to miss and prefetch) per kilo instructions retired, but for solo-run of the same program p1. We represent it by s1_L2_IN_PKI (here s1_ denotes solo-run of first program p1 out of pair p1 and p2).

5. Transferability of trained regression models

The transferability of regression models is assessed across processors i.e. regression models trained using data from Intel Xeon X5482 were used to make predictions on Intel Core2 6300 and vice versa.
Please note that each data i.e. an instance refers to a benchmark paired with a unique benchmark form 55x55 pairs. We used statistical tests and prediction accuracy metrics to assess the transferability of trained regression models. Statistical tests try to make inference on population from a given sample, while prediction accuracy metrics are confined to sample only. Here population refers to the data generated from interactions of all real world applications with architecture. While data collected for the pairs formed by SPEC cpu 2006 are 55X55 (i.e. 3025), represents a sample from population of the real world data set. These methods are discussed in next subsections.

5.1. Statistical tests

We followed the statistical methods [25] [34] to compare two alternatives for assessing the transferability of trained regression models. These methods fall in two categories parametric and non-parametric. Parametric methods include t-test, where the data are assumed to be normally distributed. If there is any reason to doubt the assumption of normality of data, then we can use a distribution free test i.e. Wilcoxon test, which falls under non-parametric methods. We used R [34] for performing statistical computations.

The significance testing is done using p-values, where p-values less than the threshold (0.05) indicate towards rejection of null hypothesis [25] [35].

First we test about the normality of the data (i.e. class variable) and then test about the difference between predicted and actual values of the class variable.

Testing for Normality of data:

The class variable solo-run L2 cache stress was collected for all 55 benchmarks in solo-run experiment. We test the normality of class variable by three methods: quantile-quantile (Q-Q) plot, Kolmogorov-Smirnov test and Shapiro-Wilk test [34]. The Q-Q plot is shown in fig. 5, where the sampled data is shown as circles against theoretical quantiles (i.e from normal distribution) shown as straight line.
If the sampled data comes from normal distribution then it should follow the theoretical quantiles on quantile-quantile (Q-Q) plot. The calculated p-values from tests for normality are shown below:

Kolmogorov-Smirnov test:	1.050e-13
Shapiro-Wilk test :	1.542e-09

The p-values are much below threshold (0.05). The Q-Q plot (i.e. points shown as circles v/s straight line) also indicates towards rejection of the null hypothesis (i.e. the data follows normal distribution). Hence we observe that there is doubt to assume that the data come from normal distribution.
The number of pairs of benchmarks i.e. instances of data used in study is 418. The 418 instances collected from XeonX5482 and Core2 6300 are used to generate regression models for XeonX5482 and Coere2 6300

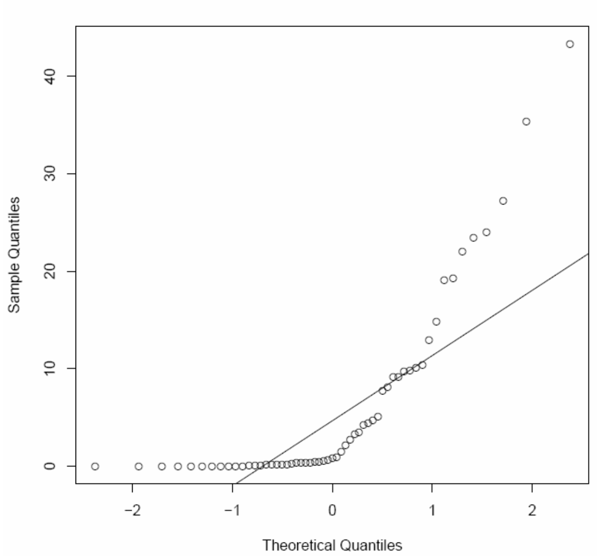

Figure5. Q-Q plot for class variable

respectively. The trained model for XeonX5482 was used to predict solo-run L2 cache stress for Core2 6300 and vice versa. To assess transferability we perform the statistical tests to check the difference between actual values and the predicted values.

Testing for difference between actual and predicted values:

We have two samples to compare against each other for presence of any significant difference between them. First sample consists of predicted and the other sample consists of actual values of solo-run L2 cache stress on a given processor. The parametric method (t-test) assumes the data to have normal distribution.

Table 2. p-values for t-test and Wilcoxon test across XeonX5482 and Core2 6300

Algorithm used	Trained using data from processor	Tested using data from processor	p-value for t-test	p-value for Wilco-xon test
LR	Xeon	Core2	0.7646	0.1947
LR	Core2	Xeon	0.7893	0.3319
ANN	Xeon	Core2	0.541	0.00018
ANN	Core2	Xeon	0.9977	0.3938
M5’	Xeon	Core2	0.7022	0.9486
M5’	Core2	Xeon	0.8162	0.8084
IBK	Xeon	Core2	0.4535	0.9556
IBK	Core2	Xeon	0.2067	0.05917
K*	Xeon	Core2	0.03878	0.2088
K*	Core2	Xeon	0.7028	0.0143
SVM	Xeon	Core2	0.6147	0.09743
SVM	Core2	Xeon	0.928	0.5135

The number of instances 418 is quite large, whereby the parametric method (t-test) becomes robust enough to be used for non-normal data [35]. Hence to assess the transferability we use both parametric method (t-test) as well as non-parametric method (Wilcoxon test). The acceptance of null hypothesis indicates the absence of significant difference between the predicted and actual values. In other words we can say that the given regression model used to predict solo-run L2 cache stress is transferable across the two processors viz. one on which it was trained and the other on which it was used to make predictions.

The results of p-values for t-test as well as Wilcoxon test are shown in Table-2 against the machine learning algorithm and processor used for training and testing. For most of the cases p-values for both tests is greater than threshold (0.05), indicating the acceptance of the null hypothesis, which says there is not significant difference between actual and predicted values of class variable solo-run L2 cache stress. There are few cases where one of the test gives p-values lower than threshold, we plan to investigate it in our future work.

5.2. Prediction accuracy metrics

The prediction accuracy of the trained models can be expressed in terms of the prediction metrics [31] given below. Here p_i and a_i refer to predicted and actual values of i^{th} instance and N is total number of instances.

Correlation Coefficient (C): It measures the extent of relationship between predicted (p_i) and actual (a_i) values. Its value ranges from –1 to 1, where 1 corresponds to ideal case.

Table 3. Prediction accuracy metrics of different algorithms across XeonX5482 and Core2 6300

Algori thm used	Trained using data from processor	Tested using data from processor	C	MAE	RMSE
LR	Xeon	Core2	0.9928	1.1549	1.869
LR	Core2	Xeon	0.9922	0.7285	1.1699
ANN	Xeon	Core2	0.9718	1.8035	3.7943
ANN	Core2	Xeon	0.9844	0.8603	1.6337
M5’	Xeon	Core2	0.9766	1.3619	3.4289
M5’	Core2	Xeon	0.9844	0.8603	1.6337
IBK	Xeon	Core2	0.9247	2.5865	5.7782
IBK	Core2	Xeon	0.9485	1.6731	5.0481
K*	Xeon	Core2	0.8754	2.4519	7.3883
K*	Core2	Xeon	0.9524	1.0873	2.7824
SVM	Xeon	Core2	0.9938	1.0508	1.6873
SVM	Core2	Xeon	0.9930	0.6135	1.1400

It is given by:

$$C = \frac{S_{PA}}{\sqrt{S_P S_A}} \text{ , where}$$

$$S_{PA} = \frac{\sum_{i=1}^{N}(p_i - \overline{p})(a_i - \overline{a})}{N-1},$$

$$S_P = \frac{\sum_{i=1}^{N}(p_i - \overline{p})^2}{N-1}$$

$$S_A = \frac{\sum_{i=1}^{N}(a_i - \overline{a})^2}{N-1}$$

$\overline{p}$ and $\overline{a}$ are mean of predicted and actual values.

Mean Absolute Error (MAE): For calculating this, the absolute value of error between predicted and actual values is used. In ideal case it should have 0 value. It is calculated as shown below:

$$MAE = \frac{\sum_{i=1}^{N}|p_i - a_i|}{N}$$

Root Mean Squared Error (RMSE): It is measured in the same unit as that of the measured quantity (for this case solo-run L2 cache stress). Its value ranges from 0 to infinity, where 0 indicates ideal case. It is calculated in following manner:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(p_i - a_i)^2}{N}}$$

The prediction accuracy metrics for different algorithms used in the study across XeonX5482 and Core2 6300 processors are shown in Table 3. The prediction accuracy metrics shown indicate that most of the trained regression models perform reasonably well across two different architectures, where one is used for training and other is used for testing. The instance based classifiers IBK and K* seem to be on lower side with respect to performance accuracy metrics. In their case a new regression equation is fitted each time, for making prediction on a new instance. This may be attributed to lower performance metrics across different architectures.

6. Conclusions and future work

In this study we assessed the transferability of trained regression model across two multi-core processors viz. Intel quad-core Xeon X5482 and Intel dual-core Core2 6300. It was observed that at 0.05 level of significance, most of the regression models under study are transferable across the L2 cache architecture of these two processors. The prediction accuracy metrics indicate that most of the trained regression models are transferable across the two processors used in the study.

The model transferability is an important property for the trained regression models for predicting solo-run L2 cache stress of programs running on multicore processors. A transferable regression model can later be used on various multi-core processors to predict the solo-run L2 cache stress of running programs. The predicted solo-run L2 cache stress of a program provides information about the L2 cache related behavior of that program on given multi-core processor. One of the potential uses of the predicted solo-run L2 cache stress is to help in intelligent scheduling of programs on multi-core processors. Implementing the intelligent scheduler is part of our future work.

7. References

[1] Intel® 64 and IA-32 Architectures Software Developer's Manuals. http://www.intel.com/products/processor/manuals.

[2]] Poonacha Kongetira, Kathirgamar Aingaran, Kunle Olukotun, “Niagara, a 32-way Multithreaded Sparc Processor” in *IEEE Micro*, March-April 2005 (pp. 21-29).

[3] Niagara2:A Highly Threaded Server-on-a-Chip http://www.opensparc.net/pubs/preszo/06/04-Sun-Golla.pdf.

[4] Yuejian Xie Gabriel H. Loh, “Dynamic Classification of Program Memory Behaviors in CMPs”, in *Proceedings of CMP-MSI: 2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects* in conjunction with the 35th International Symposium on Computer Architecture (ISCA-35) Beijing, China, June 22nd, 2008.

[5] D. Chandra, F. Guo, S. Kim, and Y. Solihin, “Predicting Inter-Thread Cache Contention on a Multi-Processor Architecture”, in *Proceedings of the 12th International Symposium on High Performance Computer Architecture HPCA*, 2005 (pp. 340-351) .

[6] N. Rafique, W. T. Lim, and M. Thottethodi, “Architectural Support for Operating System-driven CMP Cache Management”, in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Seattle, Washington, Sept. 2006 (pp. 2-12).

[7] G. E. Suh, S Devadas, and L. Rudolph, “A New Memory Monitoring Scheme for Memory-Aware Scheduling and Partitioning”, in *Proceedings of the 8th International*

Symposium on High Performance Computer Architecture HPCA, 2002 (pp. 117-128).

[8] Lisa R., Steven K., Ravishankar I. and Srihari M., "Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource", in *Proceedings of the 15th International Conference on Parallel Architectures and Compilation Techniques*, Seattle, Washington, Sept. 2006 (pp. 13-22).

[9] A. Fedorova, M. Seltzer and M. D. Smith, "Improving Performance Isolation on Chip Multiprocessors via an Operating System Scheduler", in *Proceedings of the Sixteenth International Conference on Parallel Architectures and Compilation Techniques*, Brasov, Romania, Sept. 2007 (pp. 25-38).

[10] A. Fedorova, M. Seltzer, C. Small and D. Nussbaum, "Performance Of Multithreaded Chip Multiprocessors And Implications For Operating System Design", in *Proceedings of USENIX 2005 Annual Technical Conference* Anaheim, CA, April 2005 (pp. 395-398).

[11] Jitendra Kumar Rai, Atul Negi, Rajeev Wankar, K.D. Nayak: "On Prediction Accuracy of Machine Learning Algorithms for Characterizing Shared L2 Cache Behavior of Programs on Multicore Processors", in *Proceedings of The 2009 IEEE International Conference on Computational Intelligence, Communication Systems and Networks (CICSyN2009)*, July 23-25 2009, Indore, India (pp. 213-219).

[12] Y. Wang and I. Witten, "Inducing model trees for continuous classes", in *Proceedings of the 9th European Conf. on Machine Learning*, Poster Papers, 1997.

[13] R. Quinlan, "Learning with continuous classes", in *Proceedings of the 5th Australian Joint Conference on Artificial Intelligence (AI'92)*, 1992.

[14] L. Breiman, J. Friedman, R. Olshen, and C. Stone, *Classification and Regression Trees*. Wadsworth International Group, 1984.

[15] Xudong S., Feiqui S., Jih-kwon P., Ye Xia and Zhen Yang, "CMP Cache Performance Projection: Accessibility vs. Capacity", in *Proceedings of dasCMP2006,* 2006.

[16] Pavlos P., Georgios K., Hakan Z., Stefanos K. and Erik Hagersten,"Modeling Cache Sharing on Chip Multiprocessor Architectures", in *Proceedings of IEEE International Symposium on Workload Characterization (IISWC)*, 2006 (pp. 160-171).

[17] D. K. Tam, R. Azimi, L. B. Soares and M. Stumm, "RapidMRC: Approximating L2 Miss Rate Curves on Commodity Systems for Online Optimizations", in *Proceedings of the 14th International Conference on Architectural Support for Programming Languages and Operating Systems*, Washington, DC, USA. March, 2009 (121-132).

[18] Richard M. Y., Han L., Kingsum C. and Hsien-Hsin S. L., "Constructing a Non-Linear Model with Neural Networks for Workload Characterization", in *Proceedings of the IISWC*, 2006 (pp. 150-159).

[19] K. K. Pusukuri and A. Negi, "Applying machine learning techniques to improve GNU/Linux process scheduling", in *Proceedings of IEEE Tencon Conference*, Australia, Dec., 2005 (pp. 393-398).

[20] K. K. Pusukuri, A. Negi, "Characterizing process execution behavior using machine learning techniques" in *Proceedings of HiPC International Conference, DpROM'4 workshop,* Bangalore, India, Oct., 2004.

[21] ElMoustapha O. and James W., Charles Y. and Kshitij A. D., "On the Comparison of Regression Algorithms for Computer Architecture Performance Analysis of Software Applications", in *Proceedings of the First Workshop on Statistical and Machine learning approaches applied to ARchitectures and compilaTion (SMART'07)***,** Ghent, Belgium, January 27 2007.

[22] ElMoustapha O. and James W. and Charles Y., Kshitij A. D. and Seth A., "Using Model Trees for Computer Architecture Performance Analysis of Software Applications", in *Proceedings of the IEEE International Symposium on Performance Analysis of Systems & Software, ISPASS 2007*. (pp. 116-125).

[23] ElMoustapha O.., Doshi, K.A., Yount, C., Woodlee, J., "Characterization of SPEC CPU2006 and SPEC OMP2001: Regression Models and Their Transferability",. in *Proceedings of the 2008 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS* 2008 (pp. 179-190).

[24] Ramazan B., Engin I. and Jose F. Martinez, "Coordinated Management of Multiple Interacting Resources in Chip Multiprocessors: A Machine Learning Approach", in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, Lake Como, Italy, Nov. 2008 (pp. 318-329).

[25] R. E. Walpole, R. H. Myers, S. L. Myers and K. Ye, *Probability and statistics for engineer and scientists*, 8th Edition, Pearson Education, Delhi, 2007 (pp. 413-467).

[26] T. Mitchel, *Machine Learning*, McGraw Hill 1997.

[27] D. Aha, D. Kibler (1991). Instance-based learning algorithms. Machine Learning. (pp. 6:37-66).

[28] John G. Cleary, Leonard E. Trigg: K*: An Instance-based Learner Using an Entropic Distance Measure. In: 12th International Conference on Machine Learning, 1995 (pp. 108-114).

[29]Alex J. Smola, Bernhard Scholkopf (1998), "A Tutorial on Support Vector Regression", NeuroCOLT2 Technical Report Series - NC2-TR-1998-030.

[30] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, "Improvements to SMO Algorithm for SVM Regression", Technical Report CD-99-16, Control Division Dept of Mechanical and Production Engineering, National University of Singapore.

[31] Ian H. Witten and Eibe Frank (2005) *Data Mining: Practical machine learning tools and techniques*, 2nd Edition, Morgan Kaufmann, San Francisco, 2005.

[32] S. Eranian "Perfmon2: The Hardware-based erformance Monitoring Interface for Linux", in *Proceedings of the 2006 Linux Symposium*, Vol. I (pp. 269-288).

[33] Standard performance evaluation corporation. SPEC CPU2006. http://www.spec.org/cpu2006/.

[34] R Development Core Team.: R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. (2004)

[35] Non parametric test with small and large samples. http://www.graphpad.com

[36] Sergey Zhuravlev, Sergey Blagodurov and . Alexandra. Fedorova, "Addressing Shared Resource Contention in Multicore Processors via Scheduling", in *Proceedings of the*

15th International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS'10, Pittsburgh, Pennsylvania, USA March 13-17, 2010 (pp. 129-142).