Online Linear Regression of Sampling Data from Performance Event Counters*

Qiang Wu and Oskar Mencer

Department of Computing, Imperial College London,
South Kensington, London SW7 2AZ, UK
{qiangwu,oskar}@doc.ic.ac.uk
http://comparch.doc.ic.ac.uk

Abstract. Sampling is a common way to collect execution information with performance event counters. However, the sampling data generated from performance event counters tend to be massive if sampling with high frequencies. Storing and processing a large amount of sampling data require much disk space and computing time. In this paper, we propose the online linear regression method to reduce the size of the sampling data. Our idea is to fit the sampling data with a series of straight lines. Each line represents the variation trend of the sample values within the corresponding section. Then we store the parameters of the lines instead of the sample values, resulting in a reduction of the sampling data size. The SPEC CPU 2006 benchmarks are tested to verify the proposed online linear regression method. The experimental result shows the online linear regression method can reduce the sampling data size effectively with a low overhead, and retain the variation characteristic of the sampling data with a normalized estimated standard deviation less than 0.1.

Key words: sampling data compression, performance event counters, online linear regression

1 Introduction

Performance event counters provide a low overhead facility to collect runtime information of the programs. A common way to use performance event counters in performance analysis is to sample the running program at the overflow of the specific performance event counter and store the runtime context including the count values of other performance event counters for further investigations [1][3][2]. Various tools based on the performance event counters provide such sampling functionality with additional features like sampling period randomization, sampling data aggregation and histograms [10][11][13][15].

However, the sampling with performance event counters tends to generate a large amount of data for the analysis tools to process. Fig. 1 shows the sampling data size for SPEC CPU 2006 benchmarks with different input sets at a sampling frequency of 1 Mega unhalted core cycles.

* This work is supported by EPSRC grant - Liquid Circuits: Automated Dynamic Hardware Acceleration of Compute-Intensive Applications

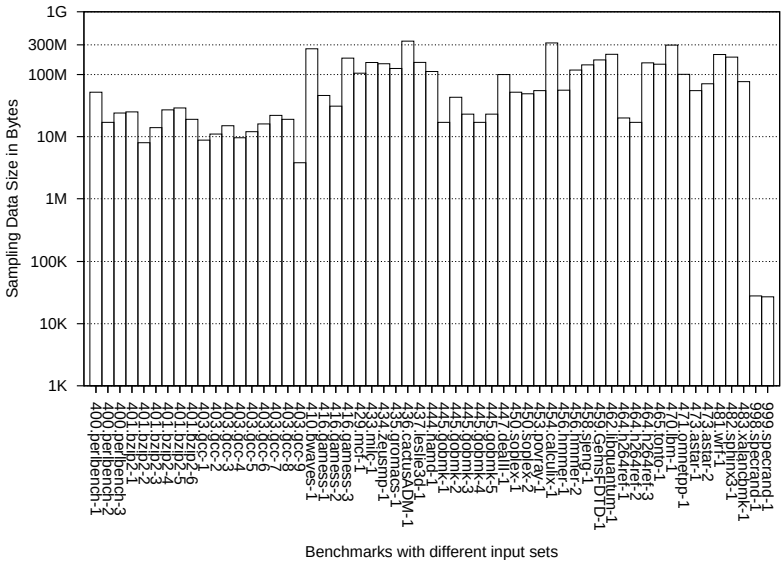

Fig. 1. Sampling data size for SPEC CPU 2006 benchmarks with different input sets at a sampling frequency of 1 Mega unhalted core cycles

We can see in Fig. 1 that the sampling data size of most benchmarks are from 10M to 100M bytes. Some of the benchmarks even produce a sampling data size around 300M bytes. The test machine has 2 GHz Xeon processors installed. So 1 Mega unhalted core cycles means up to 2,000 sampling interrupts per second, or up to 2,000 samples per second. Higher sampling frequency means larger sampling data size. A method to reduce the sampling data size is preferable especially for high sampling frequencies.

One possible approach is to apply the data compression algorithms to the sampling process. But the data compression algorithms are always computation intensive, they are more often a performance analysis object than a performance analysis tool. Alternative approach is to omit some of the samples in the whole record set, leading to a lossy compression of the sampling data. This is possible since the sampling data of a little lower sampling frequencies are somehow similar to those of higher frequencies. So if we omit some samples during the sampling, we may still get a profile of the program close to the original figure. Furthermore, instead of omitting the samples blindly, we may be able to pick the samples according to the characteristics of the sampling data. Since the sampling data are basically the counts of some performance events, the change in the count values is apparently one of the characteristics of the sampling data. If we capture the variation of the count values rather than record each piece of the sampling data, we are able to the reduce the sampling data size effectively while retaining the variation characteristic of the data.

In this paper, we address the issue of runtime sampling data size reduction by proposing the online linear regression to transform the sampling data to regression lines. The resultant regression lines are expected to fit the original data well so as to reflect the variation characteristic of the sampling data. Main contributions of our work are outlined as follows:

1. A method to do online linear regression for the sampling data;
2. A module based on pfmon[15] tool implementing the proposed online linear regression method.
3. Experiment on SPEC CPU 2006 benchmarks to verify online linear regression method;

2 Related Work

Performance event counters are commonly used in performance analysis. Model based processor bottleneck analysis is discussed in [1], [2], [3] and [4]. Some researchers investigate the data mining techniques to interpret the sampling data as in [5] and [6]. In [7], the performance counters are used to guide the compiler optimization options. Workload characterization with performance counters is also a focus of study like in [8] and [9]. Despite the various research work with performance event counters, the sampling data reduction is not specifically studied.

Many tools based on performance event counters are developed. Intel and AMD have dedicated tools for their processors [10][11]. PAPI [12] aims to provide a portable interface to utilize performance counters on various platforms. OProfile [13] is an open source system profiling tool based on performance counters for Linux. HPCToolKit [14] provides executable analysis with statistical sampling. The perfmon [15] used in this paper is also an open source tool with thread-specific counting and sampling functionalities.

3 Method

We notice that performance event counters count the occurrence of specified events during the performance monitoring. That is to say the performance event counter values always increase with the sampling process. So the idea arises that maybe we can use straight lines to fit the event count values against the sampling period. Since the event count values may change abruptly, the fitting result could be multiple lines to reflect the changes in the sampling data, leading to a piecewise linear regression problem [17]. We adopt the following method to solve it in an online situation.

3.1 Definitions

Suppose the sampling data is a series of the records,

$$(x_1, \mathbf{y_1}), ..., (x_i, \mathbf{y_i}), ..., (x_n, \mathbf{y_n})$$

where n is the number of samples. In practice, clock cycles or instructions are always used as the sampling period. So x_i may be clock cycle counts or instruction counts, or any other count values that used as the sampling period. $\mathbf{y_i}$, on the other hand, is the vector of count values sampled in the sampling interrupts. $\mathbf{y_i}$ can be cache misses, mispredicted branches, floating point operations and other count values interested. For most cases, $\mathbf{y_i}$ is a group of count values recorded with available performance event counters. To ease the demonstration of our online linear regression method, we assume $\mathbf{y_i}$ contains one data item, hence written in y_i in the succeeding sections. It is able to extend the online linear regression method to the case that $\mathbf{y_i}$ represents a vector of count values.

Let $y = kx + b$ represent a line. After online linear regression, the sampling data will be represented with a series of lines,

$$(x_1, k_1, b_1), ..., (x_i, k_i, b_i), ..., (x_m, k_m, b_m)$$

where k_i and b_i are the slope and intercept of the ith line respectively, and x_i indicates the start and end point of the $(i-1)$th and ith lines. y_i is omitted in the lines since we can calculate $\hat{y}_i = k_i x_i + b_i$ as the approximation of y_i.

Obviously each line has more parameters than the original sampling record. So if we want to reduce the sampling data size, we should make the number of lines below two third of the sample number, or nearly half of the sample number if y_i contains more than one data item.

3.2 Scaling

Since different programs and different events may produce count values far different in the actual value, scaling is needed to regulate the ranges of the count values. We scale the count values by dividing them with the initial value of the whole count value series. Let $(\dot{x}_1, \dot{y}_1), ..., (\dot{x}_i, \dot{y}_i), ..., (\dot{x}_n, \dot{y}_n)$ represent the scaled sampling data.

$$\dot{x}_i = \frac{x_i}{x_1}, \quad \dot{y}_i = \frac{y_i}{y_1}$$

Since the count values are seldom zero from the beginning, it is safe to use the division in the above expressions. In the following sections we assume that the x_i and y_i are scaled for simplicity.

3.3 Online Linear Regression

Linear regression of a group of paired data can be solved by least squares method. The formulas to calculate the slope k and intercept b for $y = kx + b$ are [18]:

$$k = \frac{n\sum_{i=1}^{n} x_i y_i - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{1}$$

$$b = \frac{\sum_{i=1}^{n} y_i \sum_{i=1}^{n} x_i^2 - \sum_{i=1}^{n} x_i \sum_{i=1}^{n} x_i y_i}{n\sum_{i=1}^{n} x_i^2 - \left(\sum_{i=1}^{n} x_i\right)^2} \tag{2}$$

Define the following variables to represent the sum terms in the formulas (1) and (2).

$$S_{x(n)} = \sum_{i=1}^{n} x_i\,, \quad S_{x^2(n)} = \sum_{i=1}^{n} x_i^2\,, \quad S_{y(n)} = \sum_{i=1}^{n} y_i\,,$$

$$S_{y^2(n)} = \sum_{i=1}^{n} y_i^2\,, \quad S_{xy(n)} = \sum_{i=1}^{n} x_i y_i$$

Substitute them in the formulas and add the subscripts for k and b, then we have

$$k_n = \frac{nS_{xy(n)} - S_{x(n)}S_{y(n)}}{nS_{x^2(n)} - \left(S_{x(n)}\right)^2} \tag{3}$$

$$b_n = \frac{S_{y(n)}S_{x^2(n)} - S_{x(n)}S_{xy(n)}}{nS_{x^2(n)} - \left(S_{x(n)}\right)^2} \tag{4}$$

If we get a new sample, (x_{n+1}, y_{n+1}), simply apply the new sample to the formulas (3) and (4):

$$\begin{aligned} k_{n+1} &= \frac{(n+1)S_{xy(n+1)} - S_{x(n+1)}S_{y(n+1)}}{(n+1)S_{x^2(n+1)} - \left(S_{x(n+1)}\right)^2} \\ &= \frac{(n+1)\left(S_{xy(n)} + x_{n+1}y_{n+1}\right) - \left(S_{x(n)} + x_{n+1}\right)\left(S_{y(n)} + y_{n+1}\right)}{(n+1)\left(S_{x^2(n)} + x_{n+1}^2\right) - \left(S_{x(n)} + x_{n+1}\right)^2} \end{aligned} \tag{5}$$

$$\begin{aligned} b_{n+1} &= \frac{S_{y(n+1)}S_{x^2(n+1)} - S_{x(n+1)}S_{xy(n+1)}}{(n+1)S_{x^2(n+1)} - \left(S_{x(n+1)}\right)^2} \\ &= \frac{\left(S_{y(n)} + y_{n+1}\right)\left(S_{x^2(n)} + x_{n+1}^2\right) - \left(S_{x(n)} + x_{n+1}\right)\left(S_{xy(n)} + x_{n+1}y_{n+1}\right)}{(n+1)\left(S_{x^2(n)} + x_{n+1}^2\right) - \left(S_{x(n)} + x_{n+1}\right)^2} \end{aligned} \tag{6}$$

3.4 Decision of Creating New Lines

If the count values increase smoothly, then we can fit the count values with only one line, leading to a maximal reduction of sampling data size. But this is a case rarely to happen. In reality, we need multiple lines to fit the sampling data. As a result, we have to decide when to end the current linear regression and start a new line.

A common approach is to compare the residual r_{n+1} of the new sample (x_{n+1}, y_{n+1}) with the standard deviation σ of the current line. If r_{n+1} is much greater than σ, i.e.

$$r_{n+1} > 3\sigma$$

where

$$r_{n+1} = |y_{n+1} - \hat{y}_{n+1}|, \quad \hat{y}_{n+1} = k_n x_{n+1} + b_n$$

then the new sample is regarded as an outlier of the current line. So we end the current line at (x_n, y_n), and start a new line from (x_n, y_n) to the new sample (x_{n+1}, y_{n+1}). Otherwise, if $r_{n+1} < 3\sigma$, we include the new sample (x_{n+1}, y_{n+1}) into the current line, and update the slope k_n and intercept b_n to k_{n+1} and b_{n+1} with the formulas (5) and (6).

The standard deviation σ is estimated from the residuals of the current line. Let the $\hat{\sigma}$ represent the estimator of the standard deviation σ.

$$\hat{\sigma} = \sqrt{\frac{1}{n-2}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{7}$$

Deduce the residuals to sums of the samples

$$\begin{aligned}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 &= \sum_{i=1}^{n}(y_i - kx_i - b)^2 \\ &= \sum_{i=1}^{n} y_i^2 + \sum_{i=1}^{n} k^2 x_i^2 + \sum_{i=1}^{n} b^2 - \sum_{i=1}^{n} 2kx_i y_i - \sum_{i=1}^{n} 2by_i + \sum_{i=1}^{n} 2kbx_i \\ &= S_{y^2(n)} + k^2 S_{x^2(n)} + nb^2 - 2kS_{xy(n)} - 2bS_{y(n)} + 2kbS_{x(n)}\end{aligned} \tag{8}$$

The k and b in the above formula (8) refer to the slope k_n and intercept b_n of the current line.

If $n = 2$ which is the case when a new line is just created, the σ is not able to be estimated with the formula (7). In this case, we adopt a heuristic that is

$$|y_{n+1} - \hat{y}_{n+1}| < \alpha|\hat{y}_{n+1}|$$

where we set $\alpha = 0.01$ in our experiment.

3.5 Implementation

Combine the above method together, the procedure to perform the online linear regression is outlined below.

Procedure Online Linear Regression

```
Online_linear_regression (alpha)
{
    Initialize n, k, b, Sx, Sy, Sx2, Sy2, Sxy to zero;
    for each new sample (x, y) {
        if( n > 1 ) {
```

```
            Calculate slope k and intercept b of the current line
                from Sx, Sy, Sx2, Sy2, Sxy;
            Estimate the standard deviation
                (using alpha for the case n==2);
            Compare residual of the new sample
                with the estimated standard deviation;
            if( the residual of the new sample is too big ) {
                Print out the current line;
                Create a new line which starts from
                    the last sample of the current line;
                n = 1;
                Reset Sx, Sy, Sx2, Sy2, Sxy regarding the
                    last sample of the current line;
            }
        }
        Update Sx, Sy, Sx2, Sy2, Sxy with the new sample (x, y);
        n = n + 1;
    }
    Calculate k and b;
    Print out the current line;
}
```

The procedure processes the samples one by one, without storing them and tracing back. This means only a constantly bounded computation time is added to the processing time of each sample, which is suitable for online situations. We implement the above procedure as a custom sampling module in the pfmon [15] tool. The user can use the command in below under Linux to invoke the online linear regression sampling.

```
pfmon --smpl-module=linear-regression ...
```

The online linear regression sampling module works as other built-in sampling modules in pfmon [15]. When the execution finishes, the module will print out the information of the lines ever created during the sampling.

4 Experimental Evaluation

We perform the experiment on SPEC CPU 2006[16] benchmarks with the online linear regression method. All benchmarks with different input sets in SPEC CPU 2006 have been tested, 57 test cases in total.

4.1 Test Platform

The test platform is a workstation with 2 Intel Xeon quad-core processors running at 2 GHz. The size of the system RAM is 8 GB.

SPEC CPU 2006 benchmarks version 1.0 is installed on the system, and built with GCC and GFortran 4.1.2. Optimization switch for the GCC and GFortran is -O2.

Operating system is Ubuntu Linux 8.04 with kernel 2.6.26.2, running in 64-bit mode. The kernel is patched with perfmon[15] interface to the performance event counters. The version of perfmon kernel patch is 2.81 which is required for pfmon[15] tool version 3.52. pfmon is used to perform the sampling with the custom online linear regression module.

4.2 Test Results

We run the pfmon with online linear regression sampling module with the following options:

- The performance events to sample are Intel Core architectural events.
- The sampling period is set to 1 Mega unhalted core cycles with randomization.
- The event to perform the online linear regression is last level cache misses.

Choosing the last level cache misses to do online linear regression is based on the consideration that the last level cache misses are not easy to predict, thus should be a good testing event. It should be noted that the online linear regression method is not limited to the options we use in the experiment. Other performance events and sampling periods supported by pfmon [15] are able to apply to the online linear regression.

Data size reduction ratio Fig. 2 shows the reduction ratio of sampling data for SPEC CPU 2006 benchmarks with different input sets.

It can be seen from the Fig. 2 that most benchmarks result in a reduction ratio from 10 to 1000. Benchmark 453.povray with input set 1 gets the maximal reduction ratio at about 4000, while 450.soplex with input set 2 gets the minimal reduction ratio at about 6.

Fig. 3 and Fig. 4 show the sample points and the corresponding regression lines generated from these sample points. Apparently, benchmark 450.soplex with input set 2 has a better fit than benchmark 453.povray with input set 1. This is reasonable since benchmark 450.soplex with input set 2 has the minimal sampling data reduction ratio, while 453.povray with input set 1 has the maximal data reduction ratio, which means 450.soplex with input set 2 has more regression lines for its samples than 453.povray with input set 1 does, thus leading to a better fitting.

Data fitness Fig. 5 shows the maximal normalized estimated standard deviation of the lines for SPEC CPU 2006 benchmarks with different input sets (gammq[18] is not used due to its computation intensive iterations). The maximal normalized estimated standard deviation (MNESD) is calculated by

$$MNESD = \frac{\hat{\sigma}_{max}}{y_{max} - y_{min}}$$

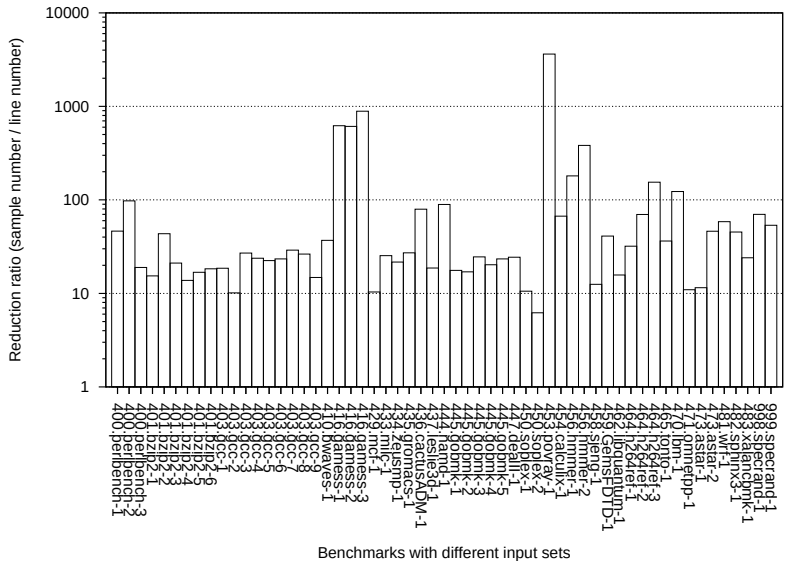

Fig. 2. Reduction ratio of sampling data for SPEC CPU 2006 benchmarks with different input sets

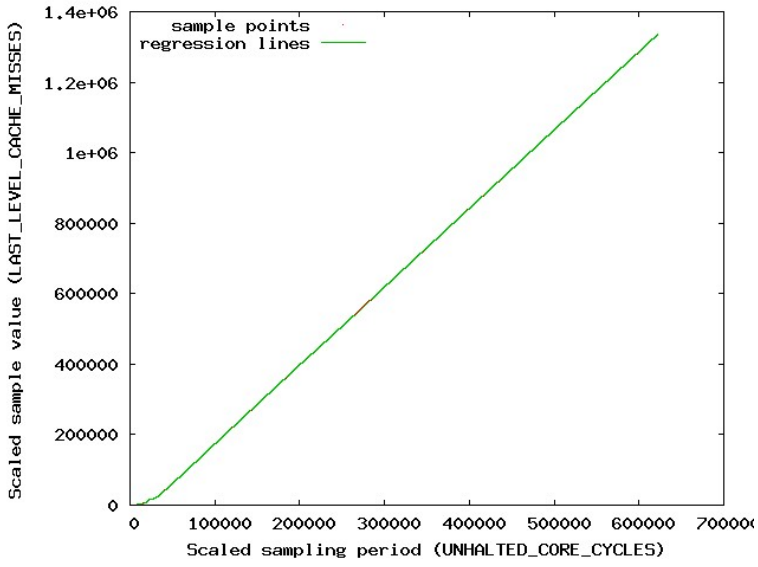

Fig. 3. Fitting result of the benchmark 450.soplex with input set 2 which has minimal sampling data reduction ratio

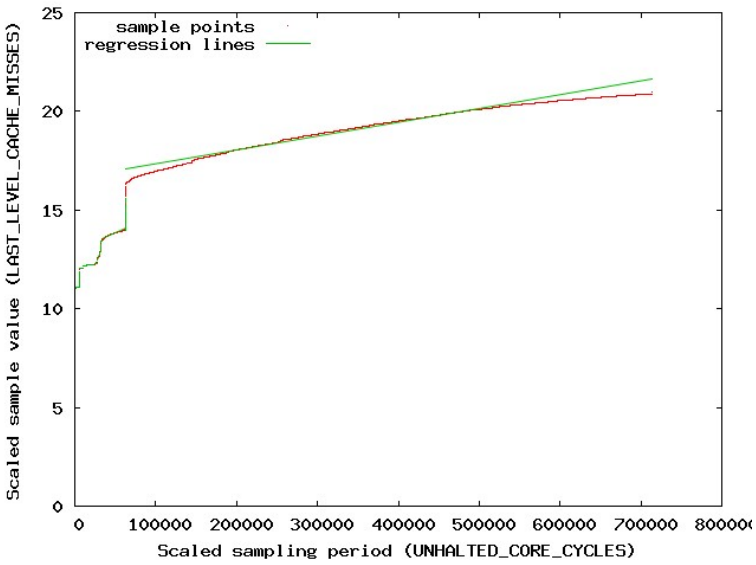

Fig. 4. Fitting result of the benchmark 453.povray with input set 1 which has maximal sampling data reduction ratio

where $\hat{\sigma}_{max}$ is the maximal estimated standard deviation, y_{max} and y_{min} are the maximal and minimal values of y in the sampling. The MNESD is used as a measurement of how well the resultant lines fit the original sampling data.

From the Fig. 5, we can find that benchmark 998.specrand has the largest maximal MNESD, followed by the 999.specrand and 436.cactusADM, while 470.lbm and 410.bwaves have the least maximal MNESD, all with input set 1. Fig. 7 and Fig. 6 show the fitting results of benchmark 998.specrand and 436.cactusADM (999.specrand is similar to 998.specrand hence omitted), while Fig. 8 and Fig. 9 show the fitting results of benchmark 410.bwaves and 470.lbm.

Obviously 998.specrand and 436.cactusADM have bad fitting results. In the Fig. 6, the last long regression line is far different from the curved part of the sample points. Fig.7 has the similar situation. A long regression line goes through the scattered sample points. This also happens in Fig. 4 where the last long regression line differs from the sample points.

The reason for this we think is that the long regression line is a result of accumulated errors from previous steps in online linear regression. Recall that the online linear regression is performed on the samples in a stepwise manner, there is a possibility that the succeeding samples are not very different from each other. So they are not regarded as outliers. But these samples may continuously move toward one direction for a period. Consequently, the estimated standard deviation may get larger and larger, allowing sample points further from the original line to be included without generating a new line. The final accumulated

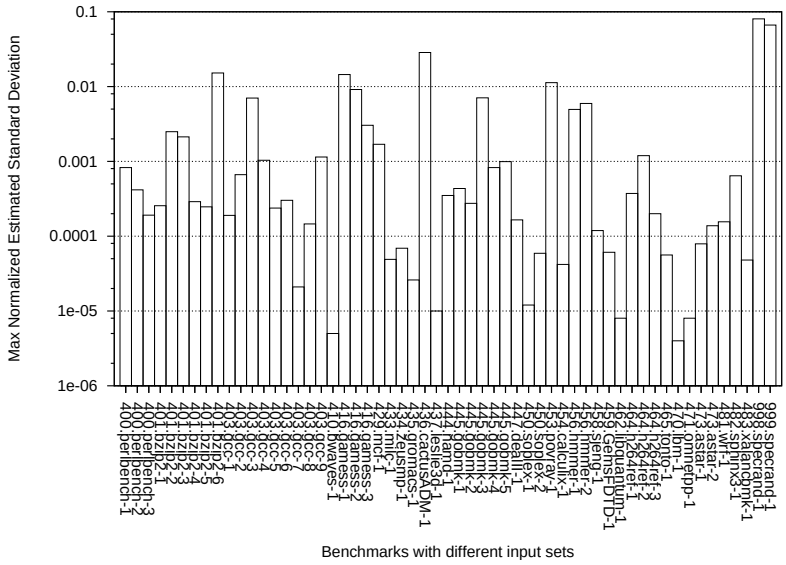

Fig. 5. Maximal normalized estimated standard deviation for SPEC CPU 2006 benchmarks with different input sets

effect is that all the samples afterwards may be fitted to one line as we have seen in Fig. 4, Fig. 6 and Fig. 7.

Fortunately, this is not a common situation in the experiment. Most test cases show a fit similar in Fig. 8 and Fig. 9 for both smooth sample points as in Fig. 9 and ragged ones as in the beginning part of Fig. 8.

Overhead We investigate the overhead of the online linear regression of the sampling data during the test. The overhead is recorded by counting the time stamps elapsed when performing the linear regression on the samples. The resultant count of time stamps is then divided by the total number of time stamps for the whole execution, showing the proportion of the online linear regression time over the total run time of the program. Fig. 10 depicts the calculated overhead of the online linear regression on SPEC CPU 2006 benchmarks. It can be seen in the figure that the overheads of all the tested benchmarks are below 1.7% at the sampling period of 1 mega unhalted CPU core cycles, or about 2,000 samples per second.

5 Conclusion

In this paper, we demonstrate the online linear regression method. The online linear regression is proposed to reduce the size of the sampling data generated by performance event counters, while keep track of the variation characteristic

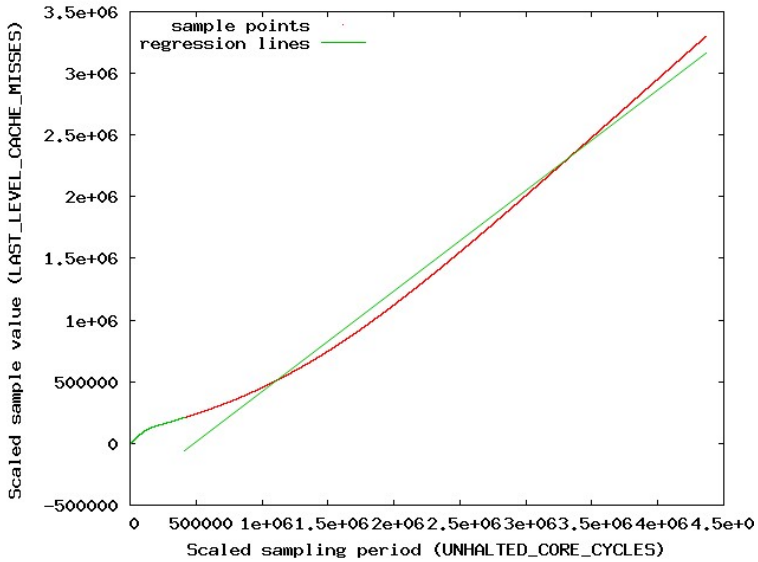

Fig. 6. Fitting result of the benchmark 436.cactusADM with input set 1 which has the highest maximal normalized estimated standard deviation

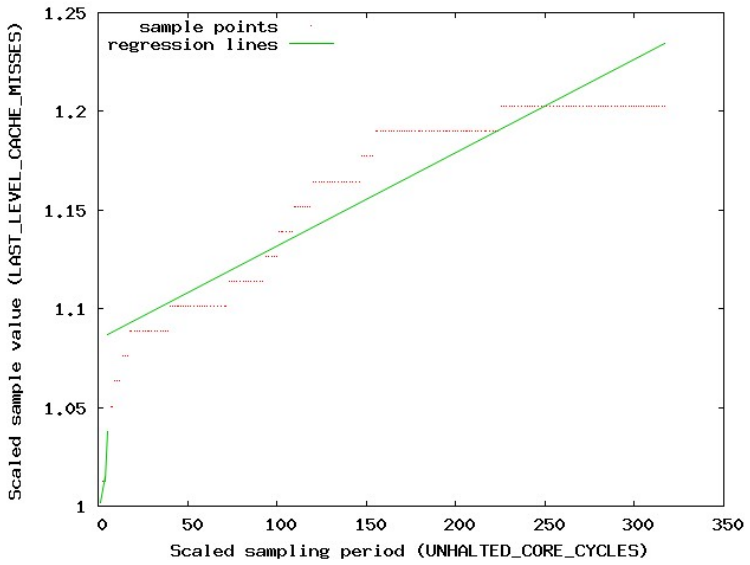

Fig. 7. Fitting result of the benchmark 998.specrand with input set 1 which has a high maximal normalized estimated standard deviation

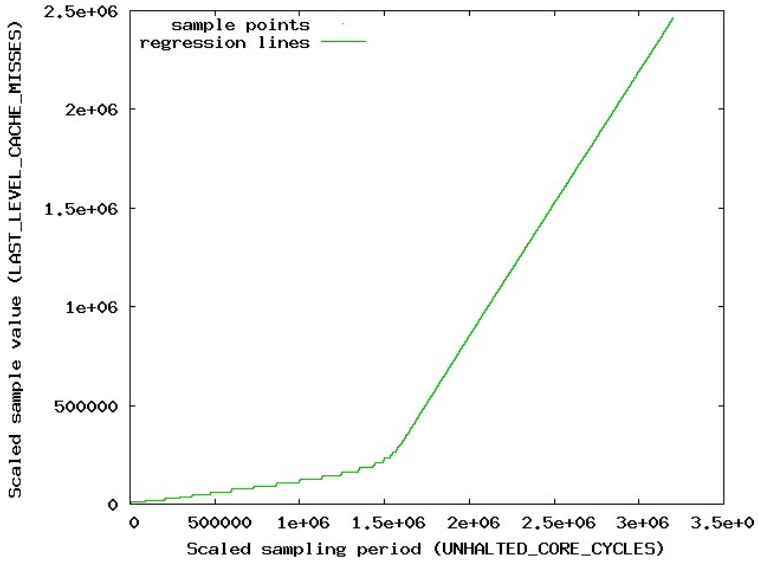

Fig. 8. Fitting result of the benchmark 410.bwaves with input set 1 which has a low maximal normalized estimated standard deviation

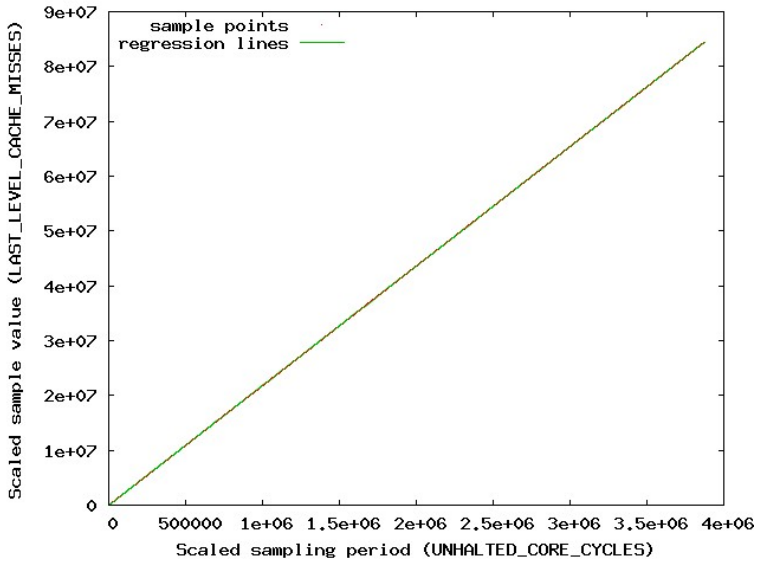

Fig. 9. Fitting result of the benchmark 470.lbm with input set 1 which has a low maximal normalized estimated standard deviation

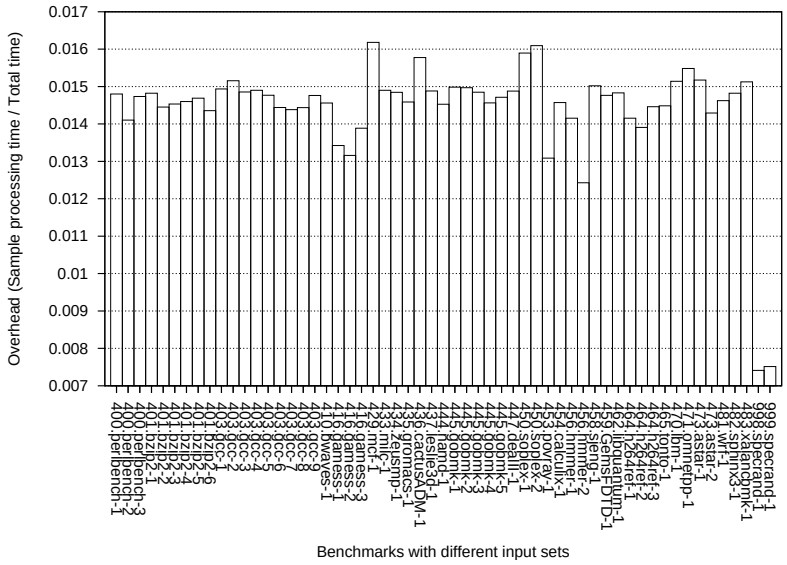

Fig. 10. Overhead of the online linear regression against the total execution time of the program, tested with SPEC CPU 2006 benchmarks at 1 Mega unhalted CPU core cycles or about 2,000 samples per second

of the sampling data. The linear regression is performed on the samples in a stepwise manner, with one sample getting processed at a time. This helps to make the linear regression operation suitable for online situation.

Experiment on the SPEC CPU 2006 benchmarks shows that the sampling data size reduction ratio ranges from 6 to about 4000 with typical ratios from 10 to 100, calculated by dividing the number of samples with the number of lines. The overheads of online linear regression for all benchmarks are less than 1.7%. And the lines fit the sample points well for most benchmarks.

However the experiment reveals that for a few cases the fitting result of linear regression is not as good as others due to the some subtle variation in the sampling data, though this is not a common situation seen in the experiment. This suggests that the online linear regression may be more suitable to use as a quick profiling method. One can sample a program with online linear regression to get an initial set of sampling data which outlines the variation of the performance event count values. Since this set of sampling data is relatively small, it is easy to analyze and identify interesting parts. Then the further investigations can be planned to focus on the interested parts for more details. Our future work is to employ the online linear regression in the program profiling framework based on performance event counters.

References

1. Marco Zagha, Brond Larson, Steve Turner, Marty Itzkowitz.: Performance analysis using the MIPS R10000 performance counters. In: Proceedings of the ACM/IEEE conference on Supercomputing, no. 16. IEEE Computer Society, Washington DC (1996)
2. Anderson, J. M., Berc, L. M., Dean, J., et al: Continuous profiling: where have all the cycles gone?. In: Proceedings of the 16th ACM symposium on Operating systems principles, pp. 1–14. ACM Press, New York (1997)
3. Azimi, R., Stumm, M., and Wisniewski, R. W.: Online performance analysis by statistical sampling of microprocessor performance counters. In: Proceedings of the 19th Annual international Conference on Supercomputing, pp. 101–110. ACM Press, New York (2005)
4. Glenn Ammons, Thomas Ball, James R. Larus: Exploiting hardware performance counters with flow and context sensitive profiling. ACM SIGPLAN Notices. 32(5), 85–96 (1997)
5. Dong H. Ahn, Jeffrey S. Vetter: Scalable Analysis Techniques for Microprocessor Performance Counter Metrics. In: Proceedings of the ACM/IEEE Supercomputing Conference, pp. 3. IEEE Computer Society, Washington DC (2002)
6. Kevin A. Huck, Allen D. Malony: PerfExplorer: A Performance Data Mining Framework For Large-Scale Parallel Computing. In: Proceedings of the ACM/IEEE Supercomputing Conference, pp. 41–52. IEEE Computer Society, Washington DC (2005)
7. John Cavazos, Grigori Fursin, Felix Agakov, Edwin Bonilla, Michael F.P. O'Boyle, Olivier Temam: Rapidly Selecting Good Compiler Optimizations using Performance Counters. In: Proceedings of International Symposium on Code Generation and Optimization, pp. 185–197. IEEE Computer Society, Washington, DC (2007)
8. Karthik Ganesan, Lizy John, Valentina Salapura, James Sexton: A Performance Counter Based Workload Characterization on Blue Gene/P. In: Proceedings of the International Conference on Parallel Processing, pp. 330–337. (2008)
9. Chang-Burm Cho, Tao Li: Using Wavelet Domain Workload Execution Characteristics to Improve Accuracy, Scalability and Robustness in Program Phase Analysis. In: Proceedings of IEEE Symposium on Performance Analysis of Systems and Software, pp. 136-145. (2007)
10. Intel Corporation. Intel VTune Performance Analyzer. `http://www.intel.com/cd/software/products/asmo-na/eng/239144.htm`
11. AMD Inc. AMD CodeAnalyst Performance Analyzer. `http://developer.amd.com/CPU/Pages/default.aspx`
12. Browne, S., Dongarra, J., Garner, N., Ho, G., Mucci, P.: A Portable Programming Interface for Performance Evaluation on Modern Processors. The International Journal of High Performance Computing Applications. 14(3), 189–204 (2000)
13. OProfile - A System Profiler for Linux. `http://oprofile.sourceforge.net`
14. HPCToolKit. `http://hpctoolkit.org`
15. perfmon project. `http://perfmon2.sourceforge.net`
16. Standard Performance Evaluation Corporation, `http://www.spec.org`
17. T. Hastie, R. Tibshirani, J. Friedman.: The elements of statistical learning. Springer, New York (2001)
18. Press, William H., Teukuolsky, Saul A., Vetterling, William T., Flannery, Brain P.: Numerical Recipes in C, 2nd ed. Cambridge University Press, Cambridge (1992)