Using Statistical Models for Embedded Java Performance Analysis

Pradeep Rao[†] Kazuaki Murakami
Department of Informatics, ISEE, Kyushu University, Japan.
Institute of Systems and Information Technologies/Kyushu, Japan.
Email: {pradeep.rao, murakami}@i.kyushu-u.ac.jp [†]Student Author

***Abstract*—This paper proposes and evaluates rigorous statistical regression techniques for Java performance analysis on a high-end embedded processor (MPC7447A). Our models relate overall Java system performance to various microarchitecture metrics and their interactions. We show that the models we develop in this paper are easy to construct, are interpretable and have high prediction accuracies.**

I. Introduction

Java$^{\text{TM}}$applications are prevalent on a wide variety of processors, many of which provide on chip performance monitoring units (PMU) to measure and track system performance. Due to additional complexities associated with running Java applications on a virtual machine (JVM), it is necessary to clearly understand the influence of the various factors that affect performance. As with any complex system, a thorough performance analysis of Java applications is critical to optimize current systems and to aid design and development of future platforms for Java.

However, complex microarchitecture features in contemporary processors obfuscate interpretation of data measured using performance counters. Moreover, the *one factor at a time* approach, adopted by reported Java performance analyses[1], inherently ignore higher order interactions that may exist between factors that contribute to performance. Consequently, the methodology skews results and may result in incorrect conclusions drawn from performance analyses.

We propose and illustrate the use of rigorous statistical techniques for Java performance analysis. Specifically, we build regression models that relate overall Java system performance to various microarchitecture metrics and their *interactions*. An interaction is said to exist when two events occur at the same time and the performance impact attributed to one event depends on the occurrence of the other event. For example, the performance impact of cache misses would depend on the magnitude of TLB misses since software page-table walk routines tend to pollute the cache. Our approach teases out several such nuances of production environments and quantifies their effect on overall performance. Moreover, these models are entirely constructed using measurements on a production system running a commercial Java runtime.

Georges, *et.al.*[1] identify flaws in existing methodologies used in Java performance evaluation and advocate the use of statistical rigor in performance evaluations. An earlier paper [2] identifies significant processor parameters for simulation using the Plackett-Burman (PB) experimental design. However, the disadvantage with this approach is that PB designs inherently ignore all factor interactions. Recent studies on modeling software performance [3], [4], try to relate observed workload performance to key microarchitecture events using model trees, but, none of these studies consider Java workloads. Other studies on processor performance modeling [5], [6] are based on software simulation of general purpose workloads using a processor model. Due to the complexity of the workload, these studies use partial traces and reduced input data sets respectively.

We perceive several drawbacks while applying the methodology adopted by these papers to model Java performance: Contemporary processors are far too complex to be modeled accurately in a simulator and thus, the generated models may not be representative of actual performance. Moreover, the use of reduced input data sets with Java workloads has been shown [7] to poorly represent Java execution on a real machine. In contrast, our work (a) uses measurements from real machine execution using a recent production JVM, (b) uses large data inputs and all Java applications are run to completion. Further, to the best of our knowledge, this is the first study that extends rigorous statistical modeling to advance accurate Java workload performance analysis and characterization.

II. Preliminaries

Regression analysis is a statistical tool that expresses the relationship between a *response* variable and its *predictors* over a sample space. In its simplest generalized form, the response variable y may be related *linearly* to p independent predictors or *regressor* variables $(Z_1, Z_2, .., Z_p)$ as expressed by the model in Eq.1.

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i Z_i + \epsilon \qquad (1)$$

The parameters $\{\beta_i | 0 \leq i \leq p\}$ are called the *regression coefficients* and represent the expected change in response y per unit change in $\{Z_i | 1 \leq i \leq p\}$ when all the remaining variables $\{Z_j | j \neq i\}$ are held constant. The *residual* or the error due to lack of fit is denoted by ϵ. The model describes a hyperplane in the k-dimensional space of the regressor variables Z_i and β_0 can be interpreted as the intercept of the response surface with the y-axis. Linear regression models are commonly used to estimate parameter significance and also to predict the response variable at arbitrary points in the design space.

A. Interactions

Often, the predictors interact, *i.e.*, the response of y to a change in Z_i depends on the value of Z_j. In such cases the model in Eq.1 can be easily extended to model such two-factor interactions as follows:

$$y = \beta_0 + \sum_{i=1}^{p} \beta_i Z_i + \sum_{i=1}^{p} \sum_{j=i}^{p} \beta_{ij} Z_i Z_j + \epsilon \qquad (2)$$

It is easy to see that Eq.1 can be extended to model third and higher order interactions in a similar fashion. Using substitution, the complete linear regression model can be represented as a sum of p terms in the generic form:

$$y = \beta_0 + \beta_1 X_{n1} + \beta_2 X_{n2} + ...\beta_p X_{np} + \epsilon_n \qquad (3)$$

The subscript n identifies the *trial* and the subscript p denotes the predictor in $\mathbf{X}$. In matrix terms, the above equation and its solution can be written as:

$$\mathbf{y} = \mathbf{X}\beta + \epsilon \quad \Rightarrow \quad \hat{\beta} = (\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'\mathbf{y} \qquad (4)$$

Hence, given the linear model with p terms, the data set from the design matrix $\mathbf{X}$ and the response vector $\mathbf{y}$, the *least squares* estimates of the partial regression coefficients $\hat{\beta} = (\beta_0, \beta_1, ..., \beta_{p-1})$ can be computed using Eq.4.

B. Model Building and Simplification

We fit a model consisting of all appropriate main effects and their second order interactions. This model is used as the base model for *goodness of fit* comparisons for pruned and simplified models developed next.

Overfitting refers to a model with too many parameters which fits the data well, but is unable to generalize beyond that. To avoid overfitting, we prune this base model further (by dropping unnecessary terms) using stepwise search based on *Bayesian Information Criterion* (BIC) which penalizes overfit models. The criterion is defined by the following relation:

$$BIC = \frac{n + p(ln(n) - 1)}{n(n-p)} SSE \qquad (5)$$

where, $SSE = \sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ is the error sum of squares, p is the number of parameters in the model, and n is the number of samples. The model with the minimal BIC tries to find an optimal compromise between model fit and model complexity.

C. Model Diagnostics

We use the following measures to evaluate our models:

(1) The *residuals* $\{\mathbf{r} | \mathbf{r} = \mathbf{y} - \mathbf{X}\hat{\beta}\}$, are the differences between the data and the fitted values. As a consequence of least squares estimation of β, the residuals will be uncorrelated with all predictors (and intercept, if any) and can be used to diagnose problems with the model.

TABLE I
7447A PROCESSOR PARAMETERS

Parameter	Description
Pipeline	7 stage, 4way superscalar
Frontend	12entry fetch queue; 128entry, 4way BTIC; 2048entry BHT, 2bit prediction, 8entry RAS
ITLB/DTLB	128entry, 2way, LRU replacement
L1 Caches	Separate 32KB, 8way, 32B blocks, PIPT, 1cycle read/write, PLRU replacement
L2 Cache	Unified 512KB, 8way, 64B line, LRU
Latencies	1/9/172 cycles (L1/L2/Mem)
Functional Units	4 IALU, 32 GPR; 1 FP-ALU, 32 FPR(64bit)

(2) The fit of the model can be summarized by the residual standard deviation, $\hat{\sigma} = \sqrt{\sum_{i=1}^{n} r_i^2/(n-p)}$ and R^2, the fraction of variation 'expressed' by the model:

$$R^2 = 1 - \frac{\hat{\sigma}^2}{s_y^2} \qquad (6)$$

where, s_y is the standard deviation of data and $(n-p)$ is referred to as the *degrees of freedom*.

(3) The *goodness of fit* of a reduced model (with R^2) with respect to an original model (with R_o^2) can be estimated with the *F-test* [8] using the statistic F, computed as:

$$F_{k,n-p-1} = \frac{(R^2 - R_o^2)/k}{(1-R^2)/(n-p-1)} \qquad (7)$$

where, k is the difference between the degrees of freedom of the original model and that of the reduced model. Given that the statistic F follows the F-distribution with $(k, n-p-1)$ degrees of freedom, the *p-value* is defined as the probability $P(F > |c|)$, where c is a constant. Thus, a small *p-value* for an F-test leads us to reject the *null-hypothesis*, suggesting that additional predictors in the larger model are statistically significant in predicting the response. We use this test to show the statistical significance of higher order interactions between predictors.

(4) Finally, a good measure of model quality is its prediction accuracy. We determine the prediction accuracy for a given model using *cross validation* as in other modeling studies [3]. Cross validation involves randomly partitioning the observations into n sets ($n = 10$ here). The models are then built using $n - 1$ sets and the last set is used for testing the prediction accuracy. This process is repeated n times, using a different set each time. Overall accuracy is determined by averaging the prediction metrics across all sets/folds. The prediction metric we use is the prediction error and its 95% confidence interval.

III. Experimental Framework

A. Measurement Methodology

The models we build are based on measurements on the Freescale 7447A processor – a popular high-end embedded processor for the PowerPC instruction set architecture (ISA). Our experimental platform features a 1.5GHz 7447A processor, 768MB memory and

TABLE II
PROCESSOR EVENTS MEASURED

Processor Cycles	*cycles*
Instructions Dispatched	*instd*
Instructions Completed	*inst*
Branch Mispredictions	*br.mpred*
Load Instructions	*ld*
Store Instructions	*st*
FPU Instructions	*fpu*
Branch Instructions	*br*
Other Instructions *(= inst - ld - st - br - fpu)*	*other*
DTLB miss	*dtlb.miss*
ITLB miss	*itlb.miss*
L1 I-cache miss	*il1.miss*
L1 D-cache miss	*dl1.miss*
L2 cache hits	*l2.hits*
L2 cache miss	*l2.miss*
L2 cache miss (data)	*l2.dmiss*
L2 cache miss (instruction)	*l2.imiss*
Prefetch Engine Request	*prefetch.req*
Prefetch Engine Full	*prefetch.full*

runs the Linux operating system kernel v2.6.17. The processor microarchitecture parameters are outlined in Tab.I and further details may be found in [9].

The 7447A processor features a performance monitoring unit (PMU) that can be configured to measure various microarchitecture events such as cycles elapsed, cache misses, branch mispredictions, *etc*. The processor features six 32bit counters and can be configured to count from among 200+ events specified in [10]. The Linux kernel is patched to support access to the processor PMU using the *perfctr*[11] package v2.6.28. The *perfctr* patch also extends the limited 32bit physical counters into 64bit virtual counters. Since the Java virtual machine executes Java threads as native threads, we extend the *perfctr* package to aggregate counts over all threads as the original *perfctr* package only supports measuring single-threaded processes.

We choose 18 processor microarchitecture events to measure and model embedded Java applications. These metrics are similar to those used in earlier Java performance analysis studies [3], [6]. The events and their acronyms used in this paper are shown in Tab.II and can roughly be categorized based on instruction mix, TLB, cache and branch predictor parameters.

Measurement noise due to extraneous factors are reduced by (1) performing all experiments in single user mode with all unnecessary processes and services turned off, and (2) disabling processor voltage/frequency scaling in the Linux kernel. Since multiple runs of each benchmark are required to capture all events due to limited physical counters, we test for the statistical significance of errors across multiple measurements. This is especially important in the case of Java applications since the JVMs employ dynamic optimization techniques to enhance performance. Our results indicate that the measurement error interval averages $\pm 1.53\%$ with 95% confidence, when each experimental configuration (benchmark-heap combination) is replicated five times.

TABLE III
BENCHMARK APPLICATIONS

Benchmark	Description	H_{min}(kB)
compress	file compression	8193
jess	expert shell system	2177
raytrace	raytracing	4097
db	database	8193
javac	Java compiler	8193
mpegaudio	MPEG-3 audio decompression	2177
mtrt	multithreaded raytracer	6145
jack	Java parser generator	2177
ecm	Embedded CaffeineMark v3.0	2177

B. Virtual Machine

For our model construction experiments we use the recently released Sun JavaSE runtime environment v5.0 for embedded[12] PowerPC. We use the default *mixed mode* execution and all our experiments consider a per-benchmark heap size as in [13]. We vary heap size starting from the minimum heap size (H_{min}) and increase the heap size in steps of $0.5H_{min}$ upto a maximum heap size of $6H_{min}$. The minimum heap size is determined empirically to be the least heapsize at which the Java application can execute without any `OutOfMemory` errors, subject to the caveat that the least heap allowed by the Sun^{TM}JVM is 2,177kB.

C. Benchmarks

Our workload is meant to be representative of the embedded domain and hence the applications are drawn from the SPEC JVM98[14] and Embedded CaffeineMark[15] (ECM 3.0) suites. ECM consists of five tests: *sieve, loop, logic, method and float.* Since each of these tests are basic, the composite result of all five tests are used for ECM. All SPEC JVM applications use the largest data set (`-s100`), unless mentioned otherwise. Tab.III lists the benchmark applications, a brief description and the minimum heap size required to run the benchmark on our platform.

IV. EVALUATION RESULTS

We use the statistical computing package *R*, to build and analyze our models. We begin with a qualitative examination of processor performance data to determine the relevant predictors to use for modeling.

A. Statistical Analyses

The interrelation between the key events is illustrated in the dendrogram in Fig.1. These are arrived at using hierarchical variable clustering based on squared Spearman's rank correlation coefficients as similarity measures[8]. A larger ρ^2 indicates higher correlation between the predictors connected on the figure and the horizontal line connecting the predictors marks the value of ρ^2 when extended to the vertical axis.

In cases where overfitting is a concern, redundant predictors can be eliminated by choosing only one predictor from a pair of highly correlated predictors. This is also effective when the predictors are collinear, since *multicollinearity*[8] influences effect estimation

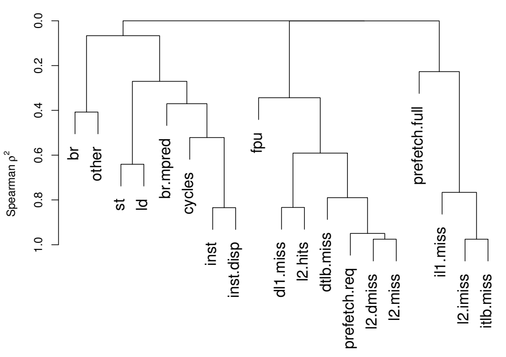

Fig. 1. Predictor clustering

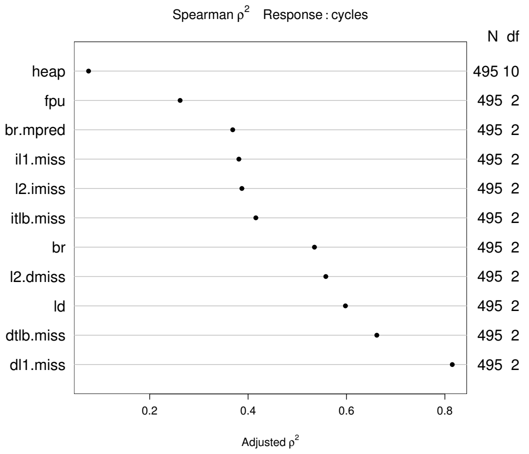

Fig. 2. Strengths of marginal relationships

by artificially inflating p-values even though the variable is significant. However, multicollinearity doesn't pose a problem when predicting and the predictions will still be accurate with R^2 (Sec.II-C2) capturing how well the model predicts the response y (processor *cycles*).

Fig.1 also shows a few obvious correlations, *e.g.*, between dispatched and completed instructions and between load and store instructions. The events *l1.dmiss* and *l2.hits* are also highly correlated suggesting that most L1 data cache misses are serviced by the L2 cache. The dendrogram also indicates a high correlation between *l2.dmiss* and *l2.miss*, suggesting that L2 cache misses are predominantly due to data accesses. This is indeed so, with over 90% of L2 cache misses due to data misses. Furthermore, we observe that these misses are also highly correlated to prefetch requests. This is because the the prefetch engines on 7447A prefetch the second block of an L2 cache line even if data in the second block is not required. Finally, the L2 cache misses due to instructions are predominantly correlated to *itlb.miss* and to a lesser degree to L1 instruction cache misses.

Since our research focusses on performance analysis, we use the total execution time, measured in processor *cycles*, as the response. In order to estimate the strength of relationship between the predictors and the response, we compute and plot the non-monotonic (quadratic in ranks) Spearman's rank correlation coefficient separately for each of the predictor variables in Fig.2. A lack of fit would be more consequential for predictors that exhibit higher ρ^2. In cases where it is difficult to assess how the predictors bear on overall performance based on microarchitectural insights, it is helpful to prioritize inclusion of predictors with higher ρ^2 in the model. Fig.2 indicates that the impact of a lack of fit would be higher (in descending order of importance) for *dl1.miss, dtlb.miss, l2.dmiss*, *etc*.

TABLE IV
MODEL DIAGNOSTICS

Diagnostic	Model		
	I-order	II-order	II-order pruned
R^2	0.907	0.992	0.992
F-test	reject	-	accept
Prediction error (%)	12.02±32.34	2.89±11.27	2.91±11.25

B. Model Evaluation

We evaluate the effect of of higher order interactions and the efficacy of the model pruning process (Sec.II-B) using metrics described in Sec.II-C. We designate a model with main effects only as a I-order model, a model with main effects and interactions between any two predictors as a II-order model, and so on. Since not all interactions are meaningful, based on microarchitectural insights (*e.g.*, l2.imiss*l2.dmiss, dl1.miss*l2.imiss *etc.*), we remove these terms from our II-order models.

The model diagnostic results are summarized in Fig.4 and Tab.IV. We observe that, though the I-order model has a reasonably high R^2 value, it exhibits high prediction errors and high prediction error intervals at 95% confidence. Also, the F-test (Eq.7), to check for the goodness of fit with the II-order model rejects the null hypothesis with 99% confidence indicating strong statistical evidence of a lack of fit from 2-way interaction terms. Tab.IV also indicates that the pruned II-order model performs comparably with the complete II-order model, while using fewer terms. This can have a significant impact in cases where the number of predictors are large. A similar experiment comparing II and III-order models indicate a lack of fit due to some third order interaction(s), but we do not investigate them further, since the second order models already achieve accuracies close to measurement error.

To ease interpretation and facilitate effects estimation, we standardize the predictors by subtracting the mean and dividing by the standard deviation to yield a *z-score*. This changes our interpretation of the

```
Coefficients              Estimate    Std. Error
------------------------------------------------
(Intercept)              1.919e+10    3.457e+08
l2.dmiss                 1.286e+10    1.530e+09
l2.imiss                -8.088e+09    6.100e+08
il1.miss                 1.261e+10    7.246e+08
dl1.miss                -4.111e+09    1.094e+09
dtlb.miss                1.962e+09    7.472e+08
br.mpred                -3.462e+09    1.198e+09
l2.dmiss*dl1.miss        3.823e+09    4.794e+08
l2.dmiss*dtlb.miss      -4.616e+09    6.143e+08
l2.dmiss*itlb.miss       1.582e+10    1.544e+09
l2.dmiss*br.mpred       -1.147e+10    9.400e+08
l2.imiss*il1.miss        2.003e+10    2.018e+09
l2.imiss*itlb.miss      -7.566e+09    1.590e+09
l2.imiss*br.mpred        2.649e+09    8.828e+08
il1.miss*dl1.miss        1.359e+10    3.166e+09
il1.miss*dtlb.miss      -9.680e+09    2.220e+09
il1.miss*itlb.miss      -1.529e+10    7.344e+08
il1.miss*br.mpred       -8.416e+09    1.918e+09
dl1.miss*dtlb.miss       2.412e+09    6.100e+08
dl1.miss*itlb.miss      -1.786e+10    2.182e+09
dl1.miss*br.mpred        1.033e+10    1.070e+09
dtlb.miss*itlb.miss      1.375e+10    2.017e+09
dtlb.miss*br.mpred      -4.079e+09    5.125e+08
```

Fig. 3. Pruned II-order model with significant interactions

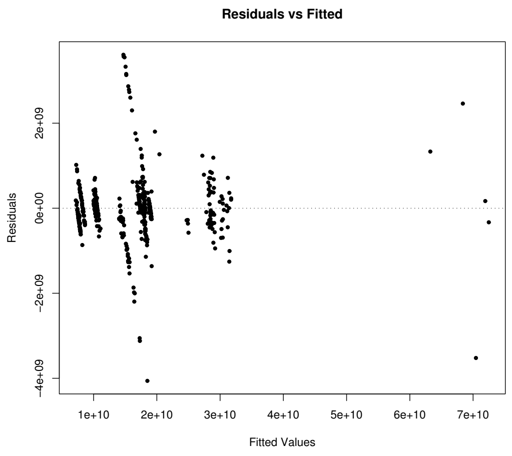

Fig. 4. Diagnostic residual plot (Sec.II-C1) for the II-order pruned model (Fig.3). The plot does not indicate any anomalies.

intercept to the mean of y when all predictor values are at their mean values. We note that the residual standard deviation and R^2 do not change since linear transformations of the predictors do not affect the fit of a regression model [16]. The resulting model is shown in Fig.3. Two interacting terms are indicated using an asterisk '*' between them.

Based on the model in Tab.3, we observe that cache events, specifically L2 cache events and their interactions, significantly contribute to execution cycles. Hence, Java applications may benefit from proactive prefetch mechanisms that improve L2 cache misses.

Due to the additional support required to study Java applications in a simulated environment [17], several studies resort to trace based simulation to understand Java characteristics. This is especially true with studies that seek to characterize and optimize cache/memory hierarchy performance for Java workloads and report improvements in terms of miss-rates. The results from our experiments can be used to extend such studies to estimate the impact of their proposals on overall performance by simply plugging in their measured metrics into our models.

V. Conclusion

This paper highlights the importance of accounting for higher order interactions in performance analyses and illustrates how that can be achieved using a framework based on statistical modeling. As a case study, we propose, illustrate and evaluate the use of linear regression models for Java performance analysis, using processor performance counter data. The models we develop are easy to interpret and achieve accuracies that are close to measurement errors. They do so by accounting for second order interactions between predictors that contribute to performance.

One can conceive a wide range of applications for these models and the system architect is advised to simultaneously tune the parameters for optimal performance. Though the modeling methodology was explored on a single platform, we feel that the methodology is generic enough to extend to other platforms. Our current research includes enhancing these models to be program phase aware and exploring other modeling techniques for performance analysis and prediction.

Acknowledgement

The authors wish to thank ISIT/Kyushu (JET Programme) for supporting this research, Prof. Nandy and Prof. Matthew Jacob (IISc) for their comments, Andy Georges for tips on extending *perfctr* and Antoine Trouve for access to the PowerPC 7447A platform.

References

[1] A. Georges, D. Buytaert, and L. Eeckhout, "Statistically rigorous Java performance evaluation," in *OOPSLA'07*.
[2] J. Yi, D. Lilja, and D. Hawkins, "A statistically rigorous approach for improving simulation methodology," in *HPCA'03*.
[3] E. Ould-Ahmed-Vall, J. Woodlee, C. Yount, K. Doshi, and S. Abraham, "Using model trees for computer architecture performance analysis of software applications," in *ISPASS'07*.
[4] E. Ould-Ahmed-Vall, K. Doshi, C. Yount, and J. Woodlee, "Characterization of SPEC CPU2006 and SPEC OMP2001: Regression models and their transferability," in *ISPASS'08*.
[5] P. Joseph, K. Vaswani, and M. Jacob, "Construction and use of linear regression models for processor performance analysis," in *HPCA'06*.
[6] B. Lee and D. Brooks, "Accurate and efficient regression modeling for microarchitectural performance and power prediction," in *ASPLOS'06*.
[7] Eeckhout et.al, "How Java programs interact with virtual machines at the microarchitectural level," in *OOPSLA'03*.
[8] F. Harrell, "Regression modeling strategies." Springer, 2001.
[9] Freescale Semiconductor, "MPC7447A RISC microprocessor hardware specifications," 2006.
[10] ——, "MPC7450 RISC microprocessor family reference manual," 2006.
[11] Perfctr, http://user.it.uu.se/~mikpe/linux/perfctr/.
[12] Sun Microsystems, http://java.sun.com/javase/embedded/.
[13] S M Blackburn et.al., "Myths and realities: The performance impact of garbage collection," in *SIGMETRICS'04*.
[14] Standard Performance Evaluation Corporation, www.spec.org.
[15] Embedded Caffeine Mark, http://www.benchmarkhq.ru/cm30/.
[16] A. Gelman and J. Hill, "Data analysis using regression and multilevel/hierarchical models." Cambridge Univ. Press, 2007.
[17] P. Rao and K. Murakami, "A sampling microarchitecture simulator for Java workloads," in *Proceedings of TIMERS-1 held in conjunction with IEEE ISPASS'08.*