Leveraging Strength-Based Dynamic Information Flow Analysis to Enhance Data Value Prediction

WALID J. GHANDOUR, HAITHAM AKKARY, and WES MASRI,
American University of Beirut

Value prediction is a technique to increase *parallelism* by attempting to overcome serialization constraints caused by true data dependences. By predicting the outcome of an instruction before it executes, value prediction allows data dependent instructions to issue and execute speculatively, hence increasing parallelism when the prediction is correct. In case of a misprediction, the execution is redone with the corrected value. If the benefit from increased parallelism outweighs the misprediction recovery penalty, overall performance could be improved. Enhancing performance with value prediction therefore requires highly accurate prediction methods. Most existing general value prediction techniques are local, that is, future outputs of an instruction are predicted based on outputs from previous executions of the same instruction.

In this article, we investigate leveraging *strength-based dynamic information flow analysis* to enhance data value prediction. We use dynamic information flow analysis (DIFA) to determine when a specific value predictor can perform well and even outperform other predictors. We apply information theory to mathematically prove the validity and benefits of correlating value predictors. We also introduce the concept of the *linear* value predictors, a new technique that predicts a new value from another one using a linear relation. We finally present a variant of stride predictor that we call *update stride*.

We then conduct an empirical analysis using *Pin*, a dynamic binary instrumentation tool, and *DynFlow*, a dynamic information flow analysis tool, that we apply to programs from the SPECjvm2008 and Siemens benchmarks. Our empirical measurements support our mathematical theory and allow us to make important observations on the relation between predictability of data values and information flow. Our analysis and empirical results show that the values of a set of selected variables can be predicted with a very high accuracy, up to 100%. Such prediction is based on the previous history and/or the values of one or more other *source* variables that have strong information flow into the predicted variable. Using our selection criteria, we show that a DIFA-directed predictor outperforms hardware value prediction for all subject programs, and sometimes by a significant margin. This was observed even when using an ideal tagged hardware value prediction table that does not suffer from aliasing or capacity misses.

Categories and Subject Descriptors: C.5.3 [**Computer System Implementation**]: Microcomputers—*Microprocessors*; C.1.4 [**Processor Architectures**]: Parallel Architectures

General Terms: Design, Performance, Theory

Additional Key Words and Phrases: Computer architecture, correlation, dynamic information flow analysis, information flow strength, information theory, instruction level parallelism, program dependence analysis, value prediction

---

This article is an extension of the conference paper Ghandour et al. [2010] presented at *PACT'10*.
This work has been supported by grants from Intel Corporation, the American University of Beirut Research Board, and Dar El-Handasah in Beirut.
Authors' address: W. J. Ghandour, H. Akkary, and W. Masri, Electrical and Computer Engineering Department, American University of Beirut, Beirut, Lebanon 1107 2020; emails: ghandour@utexas.edu, ha95@aub.edu.lb, wm13@aub.edu.lb.

DOI 10.1145/2133382.2133383 http://doi.acm.org/10.1145/2133382.2133383

ACM Reference Format:
Ghandour, W. J., Akkary, H., and Masri, W. 2012. Leveraging strength-based dynamic information flow analysis to enhance data value prediction. ACM Trans. Archit. Code Optim. 9, 1, Article 1 (March 2012), 33 pages.
DOI = 10.1145/2133382.2133383 http://doi.acm.org/10.1145/2133382.2133383

1. INTRODUCTION

Until recently, microprocessor performance had been doubling every two years. This continuous improvement in performance resulted in a fundamental transformation and vast improvement in every aspect of modern society. Unfortunately, this performance scaling has finally reached an end, forcing industry to pursue parallelism in the form of multicore processors as an alternative to single-core processor performance scaling.

Developing parallel applications, however, have proved to be very difficult. Understanding the nature of data dependences and data correlation in programs at a fundamental level can help improve the performance of single-thread as well as parallel applications in the future.

Data dependences in programs impose a fundamental limit on parallelism. In single-thread applications, data dependences define the critical execution path and consequently the total execution time. In parallel programs, data dependences between tasks require synchronization that often lead to serialization of tasks and consequently reduced parallelism. Data value prediction [Lipasti et al. 1996] has been proposed as one way to exceed this data flow limit on parallelism. However, this speculative technique requires highly accurate value prediction. With the low prediction accuracy of current data value predictors, the cost of misprediction penalty often outweighs the benefit from increased parallelism.

Value locality refers to the property that instructions in programs often execute multiple times, producing the same output values as previous executions. Empirical studies with unbounded hardware resources [Sazeides and Smith 1997b] have shown the amount of value locality in typical programs to be very high. These studies indicate that, at least theoretically, highly accurate value predictors are possible. Current value predictors, however, fall short of exploiting the full potential of value locality shown to exist in theoretical studies. New methods and insights are needed to achieve such objective.

Current data value predictors mostly exploit local correlation; that is, they predict the value of the next execution output of an instruction based on previous execution results of the same instruction. Local value prediction methods can be either computational or context based [Sazeides and Smith 1997b]. A computational predictor yields a predicted next value by performing an operation on previous values such as last value (LV) predictor [Lipasti et al. 1996] and stride value (STR) predictor [Sazeides et al. 1996]. A context based predictor makes value prediction based on its observation of previous patterns. An order k Finite Context Method (FCM) predictor uses k preceding values [Sazeides and Smith 1997a]. Global correlation between different instructions is left unexploited, except for limited cases, such as predicting the return value of a procedure based on its input arguments [Hu et al. 2003; Pickett and Verbrugge 2004a, 2004b].

In this article we use *dynamic information flow analysis* (DIFA) [Masri et al. 2004; Masri and Podgurski 2009a] to achieve the following objectives: (1) gain new insight into the nature of data value correlation between different executions of the same instruction as well as between different instructions, (2) identify highly predictable variables[1] to improve the accuracy of current local value predictors, and (3) explore the possibility of introducing new value prediction methods for exploiting global correlation between variables.

Dynamic Information Flow Analysis (DIFA). DIFA monitors information flow between variables in a program at runtime. Dynamic information flow analysis was originally used in computer security to check for information leakage and tampering. In this study, we employ a DIFA profiling tool, called *DynFlow* [Masri and Podgurski 2009a, 2009b; Masri et al. 2004], and leverage the *Pin* dynamic binary instrumentation tool [Luk et al. 2005] to measure the strength of information flows between variables at the bytecode level and instructions at the assembly level, respectively. First, a test suite is executed on the subject application to collect profiles that capture the induced information flows along with their respective source and target values. Second, instances of each flow are gathered from across all runs in order to compute various measurement metrics characterizing its strength. We use classical information theoretic methods, such as *entropy*, *conditional entropy*, and *normalized mutual information* measures, and correlation methods, such as the *product moment* correlation coefficient (aka *Pearson's r* or simply *standard r*) and *correlation ratio* (aka *eta coefficient*) to determine the strength of information flow in a chain of dependences. The dependence chain could be between two instances of the same variable or between two different variables. *Standard r* ranges in value between −1 and +1, and a larger absolute value indicates a higher degree of linear association between correlating variables. The *eta coefficient* yields the same value as *standard r* when two variables are linearly related and a greater absolute value when the variables are nonlinearly related. That is, the difference between *eta* and *standard r* is a measure of the degree of nonlinearity of the relationship between two variables. Conditional entropy measures the remaining uncertainty about a random variable X after knowing the value of another random variable Y. Mutual information is a measure of the amount of information one random variable contains about another. The value of normalized mutual information ranges between 0 and 1. A value of 0 indicates that the two variables are totally independent. While a value of 1 indicates that knowing the value of one of them, we can know the value of the other with no uncertainty.

An empirical DIFA cost analysis study is provided in Section 5.7 in Masri et al. [2007]. The study, which was conducted in the context of test case filtering, shows the average execution times of the instrumented subject programs, the average sizes of the collected profiles, and the average analysis times. The results indicate that profile collection was the primary contributor to the cost of DIFA.

We pursue two approaches in performing our study on the potential of dynamic information flow analysis in value prediction: analytical and simulation.

Analytical Approach. We make use of concepts from information theory such as Fano's inequality to prove that we can potentially achieve a higher prediction accuracy using a correlating global predictor rather than a local predictor. We also derive mathematical proofs to provide selection criteria for the best type of predictor to use under specific conditions.

Simulation Approach. We use *DynFlow* and *Pin* to measure information flow between variables and calculate their strength. We also measure predictability of variables for three types of known value predictors, last value [Lipasti et al. 1996], FCM and stride [Sazeides and Smith 1997b], to study the relation between strength of information

[1]Throughout the article, we use the term *variable* to refer to a variable at the Java level, and to an instruction result at the assembly level.

flow and predictability. The empirical study also involves new proposed predictors, for instance, linear and update stride predictors. We use *PTLsim* [Yourst 2007], a cycle accurate microprocessor simulator for x86 and x86-64 instruction sets, to show the potential of the proposed predictors. We conduct our study using the Siemens benchmark suite [Do et al. 2005] and SPECjvm2008.

1.1 Analytical Article Contributions

This article makes the following theoretical contributions.

— We introduce for the first time Dynamic Information Flow Analysis (DIFA) as a tool to study the nature of data correlation in programs and to improve the accuracy of value prediction.
— We apply Fano's inequality to (1) prove that using the previous history of a target variable and the value of another variable to predict the new value of the target variable can result in a lower bound on the prediction error than using its previous history only; (2) introduce criteria to select the best predictor to employ: instruction-based, correlating or hybrid, and the variables to utilize in the correlating predictor from the set of source variables; (3) derive an upper bound on variable prediction accuracy.
— Potential usage of standard *r*. We prove that when two different variables have a strong linear association, as indicated by absolute value of *standard r* equals 1, both source and target follow the same local value prediction pattern. If the source is predictable using a stride or FCM prediction pattern, so will the target.
— We introduce the concept of the linear value predictor. Linear value prediction can be used to predict the values of a variable whose successive values are related via strong linear flow as determined by absolute value of standard *r* equals 1. It can also be used to predict a target variable value from a source value, in situations where the two distinct variables are related via strong linear flow, and the target neither follows a linear nor FCM local prediction pattern.
— Finally we show that a nonlinear relation between a source variable and a target variable, as indicated by absolute value of eta coefficient and/or normalized mutual information equal 1, preserves FCM local prediction pattern but does not preserve a stride pattern. In other words, if the source is predictable with FCM predictor, so will the target variable. However, if the source variable is predictable with a stride predictor, the target may not be predictable with a stride predictor.

1.2 Empirical Article Contributions

Using *DynFlow*, *Pin*, and *PTLsim*, and simulating last value, linear, FCM, and stride data value predictors on benchmarks from the Siemens and SPECjvm2008 benchmark suites, we make the following contributions.

— We observe that for variables characterized by a strong linear relation from the variable to itself, as measured by standard *r*, at least one of the stride or FCM local predictors achieves very high prediction accuracy.
— We observe that for variables characterized by a strong linear relation measured by standard *r* from a source variable into a target variable, where source and target are different, the global linear predictor achieves up to 100% prediction accuracy.
— For variables characterized by a strong linear relation from a source variable into a target variable, where source and target are different, the predictability accuracies of stride, linear and FCM local predictors at the source variable are preserved, within small deviation, at the target variable.

— We observe that some of the variables characterized by a strong nonlinear relation from the variable to itself, as measured by eta coefficient, can achieve high prediction accuracy using local FCM predictor. However, all of such variables have low prediction accuracy using the local linear and stride predictors.
— For variables characterized by a strong nonlinear relation from a source variable into a target variable, as measured by eta coefficient, where source and target are different, we observe that stride and linear patterns at the source are not preserved at the target. However, if the source follows a FCM pattern, the target also follows a FCM pattern.
— We show that using DIFA to select which data values to predict outperforms current hardware value prediction on all applications we have run, sometimes by significant amount, even with an ideal tagged hardware value prediction table that does not suffer from aliasing or capacity misses.
— We show that DIFA-directed value prediction can enhance processor performance.

1.3 Article Overview

The rest of this article is organized as follows. Section 2 provides background information and reviews related work. Section 3 describes the methodology we follow. Section 4 presents our analytical insights. Section 5 introduces our newly proposed predictors. Section 6 discusses the experimental work that we conducted using DIFA. Section 7 presents our implementation of value prediction using DIFA and compares it to an implementation that employs hardware only. Section 8 shows the potential of DIFA-directed value prediction in enhancing processor performance. Section 9 proposes interesting research directions we plan to address in future work. Finally, we conclude in Section 10.

2. BACKGROUND AND RELATED WORK

2.1 Dependences and Their Impact on Parallelism

Dependences between instructions present a limiting factor on their parallel execution. There exist three types of dependences: name dependences, control dependences and data dependences.

Name dependences are due to the reuse of the same storage locations. Write-After-Write (output dependence) and Write-After-Read (antidependence) are two types of name dependences. Static or dynamic renaming techniques could easily remove name dependences.

Control dependences are introduced by conditional branch instructions. The outcome of the branch determines the next instruction to be fetched. This presents a problem for maintaining a large window of useful instructions and hence a limiting factor on the amount of Instruction Level Parallelism (ILP) that can be exploited. Generally, there are five approaches to handle the conditional branch problem: (i) branch prediction, (ii) predicated execution [Smith et al. 2006; Warter et al. 1993], (iii) multipath execution [Ahuja et al. 1998], (iv) parallel execution of control-independent code portions [Akkary and Driscoll 1998; Akkary et al. 2008; Franklin 1993], and (v) misprediction penalty reduction [Akkary et al. 2003; Al-Zawawi et al. 2007; Gandhi et al. 2004].

Like control dependences, *data dependences* present another limiting factor on ILP. Data dependence, also known as true dependence or Read-After-Write (RAW) hazard, occurs when an instruction depends on the result of a previous instruction. There are two types of data dependences: register dependences and memory dependences. Register dependence occurs when an instruction writes a register that is read by a subsequent instruction. Memory dependence occurs when a store writes to a memory location that is read by a subsequent load instruction. Register dependences are determined in the instruction decode stage. The determination of a memory dependence requires the calculation of the memory address, which cannot be done before the register operands are ready and the instruction is issued. In a superscalar out-of-order processor, a memory-order violation could occur if a load depending on a store reads a memory location before the store writes to it. Preventing loads from executing until all prior stores have executed avoids such memory-order violations. However, this solution decreases performance by unnecessarily delaying loads. Memory dependence prediction [Chrysos and Emer 1998; Moshovos 1998; Moshovos and Sohi 2002; Moshovos et al. 1997] is a solution that: (i) predicts load instructions that will cause memory-order violations if allowed to execute, and (ii) delays their execution as long as needed to avoid memory-order violations.

The following three techniques are generally used to overcome data dependences: (i) collapsing dependences by combining dependent instructions into one instruction using fused functional units [Montoye et al. 1990], (ii) executing independent instructions in parallel such as in ILP compilers [Hwu 1995] and dynamically scheduled processors [Smith and Sohi 1995], and (iii) speculating on the results of data-generating instructions. Limited parallelism can be achieved using the second technique as shown in [Austin and Sohi 1992; Butler et al. 1991; Lam and Wilson 1992]. The third technique, *data value prediction*, is the topic of this article. We provide more information about it in the following section.

2.2 Data Value Prediction

Long latency instructions can be either of variable latency (e.g., LD, DIV, and SQRT) or fixed latency (e.g., MUL and FP ADD). Data dependent instructions will stall behind long latency instructions, potentially creating one or more critical paths in the program. This may cause instruction buffers to fill up, which results in stalling the instruction fetch unit and hence, preventing new instructions from flowing into the pipeline. Data value prediction (DVP) is a technique to increase instruction level parallelism by attempting to overcome serialization constraints caused by true data dependences. By predicting instruction results early and passing them to dependent instructions, DVP reduces the length of critical paths through a program [Sazeides and Smith 1997b]. When a predicted instruction executes, the correct result is compared with the predicted one. If a mismatch occurs, the correct result is passed to the dependent instructions, which execute again with the correct result as input [Lipasti and Shen 1996]. If the benefit from increased parallelism outweighs the misprediction recovery penalty, overall performance could be improved. Enhancing performance with value prediction therefore requires highly accurate prediction methods.

Prediction mechanisms can be either computational or context based [Sazeides and Smith 1997b]. A computational predictor yields a predicted next value by performing an operation on previous values such as *stride value* (STR) predictor [Sazeides et al. 1996] and *last value* (LV) predictor[2] [Lipasti et al. 1996]. A *2-delta stride* predictor keeps track of two stride values (s_1 and s_2). S_1 and the last value are used to compute a prediction of the next value. The newly computed stride is stored in s_2. S_1 is only updated with s_2 when s_2 occurs twice in a row [Eickemeyer and Vassiliadis 1993].

A context is a finite ordered sequence of values. A context based predictor makes value prediction based on its observation of previous patterns. An order *k Finite Context Method* (FCM) predictor uses *k* preceding values [Sazeides and Smith 1997a]. *Differential Finite Context Method* (DFCM) hashes recent history of strides rather than values and looks up a stride in the hashtable to make a prediction. DFCM has the

[2]Last value predictor performs the identity operation on the previous value.

potential to be more space efficient and faster to warm up than FCM [Goeman et al. 2001]. The *register value* predictor always assumes that the target register of a load instruction contains the value to be loaded [Tullsen and Seng 1999]. The *last four value* predictor stores the four most recently seen values. It is composed of four independent last value predictors with a metapredictor that selects one of them [Burtscher and Zorn 1999]. A hybrid predictor combines distinct types of predictors, such as (STR, LV) [Wang and Franklin 1997] and (STR, FCM) [Rychlik et al. 1998] hybrid predictors.

We call the value prediction mechanisms described above local value predictors since they predict the next value of a variable based on the previous values of the same variable. On the other hand, global value predictors predict an instruction result based on results of other instructions. Some studies have addressed the correlation between control flow and value prediction. *Path identity correlation* based stride predictor maintains to each possible path the corresponding stride and last value [Mohan and Franklin 1992]. *Path-based last value* and *per-path stride* predictors improve regular last value and stride prediction techniques by incorporating path information [Nakra et al. 1999]. *Previous instruction-based* predictor correlates the prediction of the value of an instruction with that of the immediately preceding instruction [Nakra et al. 1999]. *Dynamic dataflow-inherited speculative context* (DDISC) predictor uses a context obtained from the predicted values of previous instructions in the instruction dataflow path [Thomas and Franklin 2001]. *GDiff* predictor exploits stride-based global value history. It employs the global stride form of locality expressed as $x_N = x_{N-k} + a_0$, where x_N is the predicted outcome of instruction N and x_{N-k} is the value of instruction $N - k$ in the global value history [Zhou et al. 2003]. In *gDiff*, there exists hardware limitation on the number of previously executed instructions that can be considered.

The possibility of using program profiling to support value prediction is explored in Gabbay and Mendelson [1997].

Value predictors usually incorporate a *confidence estimator* to recognize predictions that are highly probable to be wrong so that they are inhibited. A predictor with a high misprediction rate can result in slowing down the processor instead of speeding it up. Filtering techniques are proposed to select and only predict instructions of specific types, such as load instructions, as well as instructions on the critical path [Calder et al. 1999].

Many instructions are dynamically executed several times with the same inputs. There is no need to re-execute these instructions if their previous execution results can be remembered. A buffer can be used to store their results to reuse them again in subsequent execution. This concept is called *dynamic instruction reuse* (DIR) [Sodani and Sohi 1997]. The reader is referred to Sodani and Sohi [1998] for a detailed comparison between DIR and value prediction.

The previous techniques were developed in the framework of superscalar architectures. Several efforts have addressed the usage of value prediction in the framework of thread-level speculation and simultaneous multithreading [Akkary and Driscoll 1998; Marcuello et al. 1999, 2004; Oplinger et al. 1999; Steffan et al. 1999; Tuck and Tullsen 2005].

Future Execution (FE) is an approach that uses idle cores on multicore microprocessors to prefetch addresses for loads and compute the outcomes of instructions that are not directly predictable. FE uses value prediction to replace predictable instructions with load immediate instructions in the prefetching thread [Ganusov and Burtscher 2006]. Checkpointed Early Load Retirement uses register checkpointing and back-end load-value prediction to allow load dependent instructions to speculatively execute and prevent loads from disabling instructions retirement which may result in stalling the processor [Kirman et al. 2005].

Speculative method-level parallelism (SMLP) is a speculative multithreading (SpMT) variant in which speculative threads are spawned at method call boundaries. Return value prediction technique predicts functions returned values [Hu et al. 2003; Pickett and Verbrugge 2004a, 2004b; Singer and Brown 2006; Steffan et al. 1999]. SMLP can significantly benefit from accurate return value prediction [Hu et al. 2003]. Information theory was used to analyze theoretical limits on method return value predictability [Singer and Brown 2006].

2.3 Program Dependence and Dynamic Information Flow Analysis

There are two recognized basic types of program dependences, namely, *data dependence* and *control dependence*. Program dependence analysis is categorized as either *static* or *dynamic*. In the static context, it identifies dependences by analyzing the code. Whereas in the dynamic context, it identifies dependences by analyzing the code and the execution traces induced by one or more executions. Static dependence analysis is safer as it aims at identifying all potential dependences in a program, but might suffer from a high rate of false positives. On the other hand, dynamic dependence analysis is more precise as it identifies only dependences that actually occurred in the executions considered, but might suffer from a high rate of false negatives. In our work we leverage dependence analysis in its dynamic form.

Dynamic information flow analysis (DIFA) is a program dependence-based analysis technique that is typically used in computer security to enforce information flow policies in order to preserve the confidentiality and integrity of data. It entails analyzing the information flow that occurs between program variables during execution. It is based on the assumption that the occurrence of a chain of dynamic data and/or control dependences between two variables implies that information actually flows between them. A DIFA study that devises techniques to measure the strength of dynamic dependence chains[3], leading to what we call here *strength-based dependence analysis*, is presented in Masri and Podgurski [2006, 2009b].

DynFlow is a tool that supports strength-based dependence analysis of Java programs [Masri and Podgurski 2009a, 2009b; Masri et al. 2004]. It calculates flow strength at the bytecode level. *Pin* [Luk et al. 2005] is a dynamic binary instrumentation tool that supports linux, windows and MacOS executables. *Pin* is publically available, provides a rich API, and efficient instrumentation. We built a strength-based dependence analysis pintool[4] to calculate flow strength at the assembly level.

We use three techniques to measure information flow strength. The first employs classical information theory [Cover and Thomas 1991], and makes use of the entropy-based information flow characterization described in Denning [1982] and Bishop [2002]. The second technique uses the product moment correlation coefficient, which is also known as Pearson's r or simply standard r. This approach is good at measuring the strength between linearly related variables [Kachigan 1986]. The third technique uses the eta coefficient [Siegel 1956; Krus 2006], which provides the same value as standard r for linearly related variables and a greater absolute value for nonlinearly related variables. More details about information theory, standard r and eta coefficient are provided below in Sections 2.4, 2.5.1 and 2.5.3, respectively.

[3]Informally, the strength of a dynamic dependence chain is quantified by the amount of information that propagates from the source to the target of the chain, or alternatively, by the correlation between them. A formal definition can be found in Section 3 in Masri and Podgurski [2009b].
[4]A pintool is an instrumentation tool developed using Pin.

2.4 Information Theory Background

Information theory was originally developed by Shannon in the context of secure communication and cryptography. It has been applied in diverse fields such as: electrical engineering (communication theory), statistical physics (thermodynamics), computer science (Kolmogorov complexity or algorithmic complexity), mathematics (probability and statistics), philosophy of science (Occam's Razor) and economics (investment) [Cover and Thomas 1991]. Next, we define information theoretic terms that are relevant to our work.

2.4.1 Entropy. Entropy is a measure of the uncertainty of a random variable. It explicitly quantifies the information content in a given source of data. The entropy $H(X)$ of a discrete random variable X is defined as [Bishop 2002; Denning 1982]:

$$H(X) = -\sum_{i} P(X = x_i) \log_2 P(X = x_i).$$

Conditional entropy of a random variable X given that the random variable Y is known measures the remaining uncertainty about the value of X after knowing the value of Y. It is defined as:

$$H(X|Y) = -\sum_{j} P(Y = y_j) \sum_{i} P(X = x_i|Y = y_j) \log_2 P(X = x_i|Y = y_j).$$

2.4.2 Mutual Information. Mutual information is a measure of the amount of information one random variable contains about another. $I(X;Y)$ can be expressed in terms of $H(X)$ and $H(X|Y)$ as follows:

$$I(X;Y) = H(X) - H(X|Y).$$

Mutual information is symmetric, $I(X;Y) = I(Y;X)$. It can detect arbitrary nonlinear relationships between X and Y. $I(X;Y)$ can be normalized to a value between 0 and 1 by dividing it by $H(X)$.

2.4.3 Fano's Inequality. Let X and Y be two randomly correlated variables such that we know Y and wish to predict the value of X. Fano's inequality relates the probability of error in predicting the random variable X to the conditional entropy $H(X|Y)$. $H(X|Y)$ is zero if and only if X is a function of Y. In this case, we can deduce X from Y with zero probability of error.

Let $\widehat{X}$ be an estimate of X. The probability of error is defined as: $P_e = P_r(X \neq \widehat{X})$, where P_r is used as an abbreviation for probability.

(Fano's Inequality) For any estimator $\widehat{X}$ such that $X \rightarrow Y \rightarrow \widehat{X}$, with $P_e = P_r(X \neq \widehat{X})$, we have

$$H(P_e) + P_e \log|\mathcal{X}| \geq H(X|\widehat{X}) \geq H(X|Y)$$

where $\mathcal{X}$ is the alphabet of X. If the estimator $g(Y)$ takes values in the set $\mathcal{X}$, the inequality can be slightly strengthened by replacing $\log|\mathcal{X}|$ with $\log(|\mathcal{X}| - 1)$. The inequality becomes

$$H(P_e) + P_e \log(|\mathcal{X}| - 1) \geq H(X|Y).$$

The inequality can be weakened to

$$1 + P_e \log(|\mathcal{X}| - 1) \geq H(X|Y)$$

or

$$P_e \geq \frac{H(X|Y) - 1}{\log(|\mathcal{X}| - 1)}$$

2.5 Correlation-Based Techniques

In this section, we provide some information about correlation-based techniques used in this article.

2.5.1 Standard r. The Pearson product moment correlation coefficient, which is also known as Pearson's r or standard r, measures the strength of the correlation (linear dependence) between two variables. It ranges between −1 and +1 and is defined as [Kachigan 1986]:

$$r = \frac{Cov(x, y)}{\sigma_x \sigma_y},$$

where $Cov(x, y)$ is the covariance of x and y, σ_x is the standard deviation of x, and σ_y is the standard deviation of y. r^2 is called the coefficient of determination or proportion of explained variance.

2.5.2 Coefficient of Relation. In multiple regression analysis, the set of predictor (independent or explanatory) variables $X1, X2, \ldots$ is used to explain variability of the criterion (dependent or prediction) variable Y. The coefficient of multiple correlation R is a measure of the strength of the association between the independent variables and the one dependent variable. R can take any value between 0 and +1. The closer R is to +1, the stronger the linear association is. If R equals 0, then there is no linear association between the dependent variable and the independent variables. For two predictor variables, R is defined using the following formula [Allison 1998]:

$$R = \frac{\sqrt{r^2_{YX_1} + r^2_{YX_2} - 2r_{YX_1}r_{YX_2}r_{X_1X_2}}}{\sqrt{1 - r^2_{X_1X_2}}},$$

where r is the *standard r*. R^2 is called the coefficient of multiple determination.

2.5.3 Eta Coefficient. The correlation ratio or eta coefficient provides the same value as standard r for linearly related variables and a greater value for nonlinearly related variables. The difference between *eta* and r presents a measure of the nonlinearity of the relation between two variables. The eta coefficient is defined as [Krus 2006; Siegel 1956]:

$$eta = \frac{\sigma_{\overline{y}}}{\sigma_y},$$

where $\overline{y}$ is the mean of the category that y belongs to, $\sigma_{\overline{y}}$ is the standard deviation of $\overline{y}$, and σ_y is the standard deviation of y.

Now that we have reviewed background and previous relevant work, we discuss our methodology.

3. METHODOLOGY

We follow in our study two different approaches: analysis and simulation.

3.1 Analytical Approach

Following the analytical approach, we provide mathematical proofs of the relation between variables, correlation, and predictability. We make use of some concepts from information theory such as Fano's inequality [Cover and Thomas 1991; Fano 1961] to prove that we can potentially achieve a higher prediction accuracy using a correlating predictor rather than a local predictor.

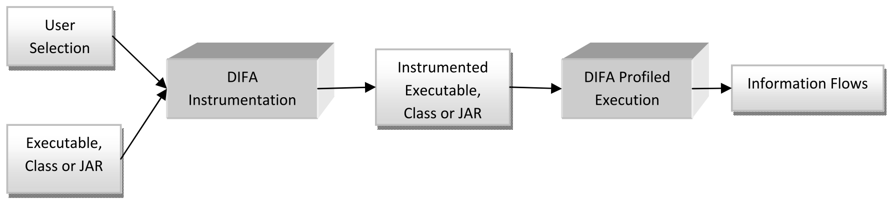

Fig. 1. Overview of DIFA tool operation.

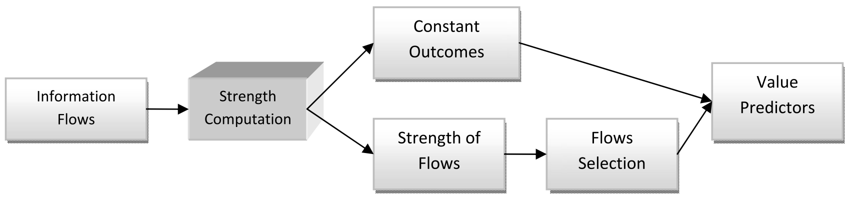

Fig. 2. Overview of strength computation and DIFA-directed value prediction.

3.2 Simulation Approach

We implement and simulate five predictors: differential finite context method (DFCM), finite context method (FCM), last value, 2-delta stride and stride. We also introduce and simulate the linear and the update stride predictors.

We use *DynFlow* [Masri and Podgurski 2009a; Masri et al. 2004] and *Pin* dynamic binary instrumentation tool [Luk et al. 2005] to determine information flow between variables and calculate their strength.

Figure 1 provides an overview of information flows generation using a DIFA tool. Our pintool allows the user to specify certain requirements such as: the variables to consider based on their type, the routines that include them or pointers that reference them, one or more arguments of a specific function, etc. A DIFA tool instruments its input, which can be an executable, class or JAR file, taken into consideration the user selection if any, to produce an instrumented output. A test suite is run on the instrumented output to collect profiles that capture the induced information flows along with their respective source and target values.

Figure 2 presents an overview of strength of flow computation and DIFA-directed value prediction. An analysis tool, that takes as input the information flows and their corresponding values obtained using the DIFA tool, identifies variables that always take the same value and computes the strength of the remaining identified dependences using information theoretic and statistical metrics applied on their associated values. Variables that always take the same value are predicted using a last value predictor. According to their strength of flow, as explained in Sections 6.2 and 6.3, the remaining variables are predicted using the appropriate value predictor.

We use *PTLsim* [Yourst 2007], a cycle accurate microprocessor simulator and virtual machine for x86 and x86-64 instruction sets, to model DIFA-directed value predictors. *PTLsim* simulates x86 codes after converting complex instructions into RISC-like micro-ops (uops), a technique used in Intel and AMD processors [Papworth 1996]. We modify *PTLsim* to support value prediction. *PTLsim* produces up to 16 micro-ops per x86 macro-instruction. We assign an ID that uniquely identifies uops that are obtained from the same x86 instruction. The value of a uop ID ranges from 0 to 15, and it is assigned in order to each uop. We obtain the index of the value prediction table using a hash function that takes as inputs the x86 instruction program counter (PC) and the uop ID.

Table I. Subset of SPECjvm2008 Benchmarks Studied in This Work

Program	Description
compress	compresses data using a modified Lempel-Ziv method (LZW)
crypto	encrypt and decrypt using several different protocols
mpegaudio	MPEG-3 audio stream decoder
scimark	floating point benchmark composed of FFT, LU, MonteCarlo, SOR and Sparse programs
serial	serializes and deserializes primitives and objects using data from the JBoss benchmark
xml	exercises the JRE's implementations and associated APIs of javax.xml.transform and javax.xml.validation

Table II. Siemens Benchmarks

Program	Lines of Code	Description
print_tokens	402	lexical analyzer
print_tokens2	483	lexical analyzer
schedule	299	priority scheduler
schedule2	297	priority scheduler
replace	516	pattern replacement
tcas	138	altitude separation
tot_info	346	information measure

We conduct our study using the Siemens benchmark suite [Do et al. 2005] and SPECjvm2008 (Java Virtual Machine Benchmark). The Siemens benchmark suite contains seven programs: print_tokens, print_tokens2, replace, schedule, schedule2, tcas and tot_info. SPECjvm2008 contains ten benchmarks: compiler, compress, crypto, derby, mpegaudio, scimark, serial, startup, sunflow, and xml. SPECjvm2008 is mainly used to measure the performance of a Java Runtime Environment (JRE). It is also used to measure operating system and hardware performance in the context of executing the JRE. In addition, it captures the performance of the processor and memory. We use all the programs from the Siemens benchmark suite, and the subset of the SPECjvm2008 benchmarks listed and described in Table I. We were unable to run some of the SPECjvm2008 benchmarks within the *DynFlow* tool. A description of the Siemens programs is provided in Table II. At the Java level, we study the predictability of all numeric variables in the programs we use. At the assembly level, we predict all value producing instructions.

4. ANALYTICAL INSIGHTS

In this section we provide upper bounds on value prediction accuracy using information theory, and mathematically study the potential usage of standard r, eta coefficient and normalized mutual information in value prediction.

4.1 Upper Bound on Prediction Accuracy

We are interested in measuring prediction accuracy limits of a variable if we use (i) previous history of the variable itself in the prediction, (ii) values of source variables that have strong flow into the predicted variable, or (iii) a combination of the two previous cases.

Consider a variable for which we may be interested in predicting its values. Each successive update to the variable may result in changing the machine state corresponding to it. Let the random variable Y represent the current state of the variable we are interested in predicting, and random variable X represent its state after the next execution. The conditional entropy $H(X|Y)$ measures the uncertainty we have in X given that we know Y.

(Conditioning reduces entropy) For any two random variables, X and Y, we have

$$H(X|Y) \leq H(X),$$

with equality if and only if X and Y are independent [Cover and Thomas 1991]. We can deduce from this theorem that $H(X|Y,Z) \leq H(X|Y)$.

(Fano's Inequality) For any estimator $\widehat{X}$ such that $X \to Y \to \widehat{X}$, with $P_e = P_r(X \neq \widehat{X})$, we have [Fano 1961]:

$$P_e \geq \frac{H(X|Y) - 1}{\log(|\mathcal{X}| - 1)}.$$

Having $H(X|Y,Z) \leq H(X|Y)$, we can extend Fano's inequality as follows:

$$P_e \geq \frac{H(X|Y) - 1}{\log(|\mathcal{X}| - 1)} \geq \frac{H(X|Y,Z) - 1}{\log(|\mathcal{X}| - 1)}.$$

We can deduce from these relations that using the previous history of a target variable and the value of another variable to predict the new value of the target variable can result in a lower bound on the prediction error than using its previous history only. This can result in a higher prediction accuracy. Both Y and Z used above may correspond to other variables, since, it is not necessary that we correlate over the previous history of the predicted variable. We can also infer from the above that increasing the number of variables involved in our prediction, may result in decreasing the lower bound on the prediction error.

When $H(X|Y) < H(X|Z)$, the probability of the error lower bound associated with $H(X|Y)$ is less than that associated with $H(X|Z)$. Thus, it is better to make use of Y instead of Z to predict X values in case we do not want to use both of them. This can be used as criteria to select the type of predictor to employ: instruction-based, correlating or hybrid, and to select the variables to utilize in the correlating predictor from the set of source variables.

The probability of prediction accuracy upper bound can be expressed as: $P_{amax} = 1 - P_{emin}$, where P_{emin} is the lower bound on the probability of error P_e. We define the theoretical prediction accuracy to be equal to P_{amax}. It represents the upper bound on the prediction accuracy.

4.2 Potential Usage of Standard r in Value Prediction

This section presents a mathematical analysis of the potential usage of standard r in value prediction.

4.2.1 Maximum r. A linear relation can be expressed as: $y = ax + b$. A value of $|r|$ equals 1, indicates that there exists a linear relation either between the values of two different variables x and y, or between consecutive values x_1 and x_2 of the same variable x. We now consider these two cases.

(1) The relation is between consecutive values of the same variable: we introduce a new predictor that we call *local linear predictor* that can be used to predict the values of the variable using the linear relation equation. When a equals 1, the predictor reduces to a stride predictor. Three occurrences of the variable are required to calculate the values of a and b. If the variable follows a stride pattern, which can be known after its third occurrence, there is no need to calculate a and b. On

the other hand, if a variable follows a nonstride pattern, the following equations can be used to calculate the values of a and b:

$$a = \frac{x_2 - x_3}{x_1 - x_2}, \text{ where } x_1 \neq x_2.$$

$$b = x_2 - ax_1 = x_3 - ax_2.$$

When consecutive values of the variables are identical, the predictor reduces to a last value predictor.

(2) The relation is between two different variables: let x and y be these variables where x is the source variable and y is the target variable. We next consider three patterns of the source variable values, stride, FCM and random, and prove that the target follows the same pattern as the source.

(a) THEOREM 1. *When the source variable follows a stride pattern of stride equal to δ; the target variable follows a stride pattern of stride equal to $a\delta$.*

PROOF. Let x_1 and x_2 be two consecutive values of source variable x, and y_1 and y_2 be the corresponding two consecutive values of target variable y. From the linear equation:

$$y_1 = ax_1 + b$$

$$y_2 = ax_2 + b$$

$$x_2 - x_1 = \delta$$

$$y_2 - y_1 = ax_2 - ax_1 = a\delta.$$ □

(b) THEOREM 2. *When the source variable follows a FCM pattern; the target follows a FCM pattern.*

PROOF. Let x_1, x_2 and x_3 be three consecutive values of source variable x, and y_1, y_2 and y_3 be the corresponding three consecutive values of target variable y. From the linear equation:

$$x_1, x_2, x_3 \ldots x_1, x_2, x_3 \ldots$$

$$ax_1 + b\,, ax_2 + b\,, ax_3 + b \ldots ax_1 + b\,, ax_2 + b\,, ax_3 + b \ldots$$

$$y_1, y_2, y_3 \ldots y_1, y_2, y_3 \ldots$$ □

(c) THEOREM 3. *When the source variable neither follows a stride nor a FCM pattern; the target variable can be obtained from the source variable using the linear relation equation. The linear equation can be defined after two occurrences of the source and target variables.*

PROOF. Let x_1 and x_2 be two consecutive values of source variable x, and y_1 and y_2 be the corresponding two consecutive values of target variable y. From the linear equation:

$$y_1 = ax_1 + b$$

$$y_2 = ax_2 + b$$

$$a = \frac{y_2 - y_1}{x_2 - x_1}, \text{ where } x_1 \neq x_2$$

$$b = y_1 - ax_1 = y_2 - ax_2$$ □

We conclude from Theorems 1, 2 and 3 that when two different variables are related with a linear equation, both source and target follow the same pattern. The global linear predictor can be used to predict the value of the target knowing the value of the source when the variables themselves neither follow a linear nor a FCM pattern.

4.2.2 Nonmaximum r. Now we consider the case where $|r| < 1$. We can use the regression line to predict the future value of the target variable, knowing the source variable value. The predicted value will be an approximation of the true value. In some Boolean and branch use expressions, it is sometimes safe to substitute an incorrect prediction. In the following code example below, if $x > 15$, any predicted value of x will result in correct execution as long as the predicted value is greater than 15. Similarly, if $x <= 15$, any predicted value of x will result in correct execution as long as the predicted value is less than or equal to 15.

```
if (x > 15) {
   // x == 16, 17, 18,...
} else {
   // x == 15, 14, 13,...
}
```

We do not evaluate this case empirically in this article, but will do this in future work.

4.3 Potential Usage of Eta Coefficient and Mutual Information in Value Prediction

In this section we consider the case where there exists a nonlinear relation between consecutive values of the same variable, or between a source variable and a target variable, captured via either eta coefficient or normalized mutual information. We note that there is no unique expression for all nonlinear relations.

THEOREM 4. *A nonlinear relation does not imply that stride pattern is preserved.*

PROOF. We prove the Theorem 4 using the following counter-example. Consider the nonlinear relation below where $a \neq 0$ and $\delta \neq 0$:

$$y = ax^2 + bx + c$$

$$x_2 - x_1 = x_3 - x_2 = \delta$$

$$y_2 - y_1 = 2ax_1\delta + a\delta^2 + b\delta$$

$$y_3 - y_2 = 2ax_1\delta + 3a\delta^2 + b\delta$$

$$y_2 - y_1 \neq y_3 - y_2$$

hence, the stride pattern is not preserved. □

We can show that a nonlinear relation preserves FCM pattern following the same reasoning provided in Section 4.2.1 for linear relation.

THEOREM 5. *The order of the repeating pattern of the target is always less than or equal to that of the source.*

PROOF. Since a nonlinear relation is a function, it associates each element in the domain with exactly one element in the codomain. Unless the relation is an injection[5],

[5]An injection or one-to-one function has the property that if $f(a) = f(b)$ then a must equal b.

more than one element in the domain can be associated with the same element in the codomain, hence, the order of the repeating pattern of the target is always less than or equal to that of the source. □

Notice that if the nonlinear relation can be defined, it can be used to compute the value of the target using that of the source or to compute the new value of a variable using its previous one.

We present in the next section our newly proposed predictors.

5. PROPOSED PREDICTORS

In this section we introduce two new predictors: *linear* and *update stride*.

5.1 Linear Predictor

We have two types of linear predictor, local and global.

5.1.1 Local Linear Predictor. The local linear predictor defines the equation of the linear relation, $y = ax + b$, that exists between successive values of the same variable and predicts its next value knowing its current value.

5.1.2 Global Linear Predictor. The global linear predictor defines the equation of the linear relation, $y = ax + b$, that exists between two different variables and predicts the target variable value knowing that of the source. Following are the values of two distinct variables that are related via a linear relation, where $a = 10$ and $b = 0$:

```
var1: 0 3 0 4 0 5 0 1 0 5 0 1 0 4 0 1 0 2 0 1
var2: 0 30 0 40 0 50 0 10 0 50 0 10 0 40 0 10 0 20 0 10
```

The parameters of the linear relation, a and b, are computed using the equations provided in Section 4.2.1. When a equals 1, the linear predictor reduces to a stride predictor.

There exists no constraint on the distance between source and target variables, since DIFA analysis considers flow of information among all variables in an executing program. Previous study has shown that the length of an information flow, that is, the number of direct data and/or control dependences in its dynamic dependence chain, is not indicative of its strength [Masri and Podgurski 2009b]. We use global linear relation, $y = ax + b$, between two distinct variables compared to the usage of global stride relation, $y = x + b$, in *gDiff* predictor [Zhou et al. 2003].

We intend to introduce, in future work, a general linear predictor that implements the linear combination, $y = a_n x_n + a_{n-1} x_{n-1} + a_{n-2} x_{n-2} + \ldots + a_1 x_1 + a_0$, where y is the predicted value of the variable of interest and x_i can be either the value of a previously updated variable or a previous value of the predicted variable, using the coefficient of multiple correlation described in Section 2.5.2.

5.2 Update Stride Predictor

Consider the following circumstances: 1) a branching statement condition is satisfied, and 2) execution reaches the beginning of a new iteration of a loop. Some variables that follow a linear pattern are set to a specific value when one of these two circumstances occurs. To accurately predict these variables, we use a modified linear predictor that we call *update linear*. The initialization of the variable and its regular update are identified using DIFA. A hash function is used to map both of them to the same entry in the prediction table. After the initialization of the variable, we update the last value of the predicted variable in the prediction table with the initial value. This prevents the miss that occurs if we are using regular linear predictor.

For variables that do not exhibit this behavior, update linear predictor works as a normal linear predictor and achieves the same prediction accuracy. Update linear predictor reduces to an *update stride* predictor when the variable follows a stride pattern.

For a repeated stride sequence, update stride predictor achieves 100% prediction accuracy. A regular stride predictor misses twice at the beginning of each new repetition of the sequence, while a 2-delta stride predictor misses once. The impact of these misses can be significant for short length sequences.

FCM predictor achieves 100% prediction accuracy for a repeated stride sequence; however, it takes a longer learning period and the number of entries required in the level-2 table[6] is equal to the length of the sequence. This requires large tables and can result in a lot of unnecessary interference in the level-2 table. DFCM requires much fewer entries than FCM; however, it still requires more entries and it is much more complicated than update stride.

Consider the following code example taken from SPECjvm *xml* benchmark:

```
private static final int XSD_NUMBER = 6;
int loops[] = {1,5, 3,52,647,419};
for (int i = 0; i < XSD_NUMBER; i++) {
for (int j = loops[i] - 1; j >= 0; j--) {
```

Variable j takes the following values:

```
0
4 3 2 1 0
2 1 0
51 50 ... 1 0
646 645 ... 1 0
418 417 ... 1 0
```

In this example, the initial value assigned to the variable *j* depends on the value of the variable *i*, and the length of the repeated sequence changes as the assigned value changes. FCM predictor is not suitable to predict the values of *j*; However, update stride predictor achieves 100% prediction accuracy.

In the following section, we empirically confirm our analytical results.

6. EMPIRICAL STUDIES USING DIFA

We conduct our study in this section at the Java level due to the availability of *DynFlow* that we use to identify dynamic information flow between variables and measure their strengths. *DynFlow* makes use of normalized mutual information, eta coefficient and standard *r* to measure the strength of dynamic information flow between variables. We have employed stride, linear, update stride and FCM predictors to predict the values of all numeric variables.

[6]A FCM predictor makes use of two levels of prediction tables, where the first level table stores the recent context of an instruction, and the second level table stores for each possible context the value which is most likely to follow it.

Table III. *schedule* Program Integer Variables Measurements

Variable Name	Stride Prediction Accuracy	FCM Prediction Accuracy	Normal Mutual Info.	Max Eta	Max r	Total Flows	Zero Flows
job.val	100	0	1	1	1	8	2
allocPocNum	100	0	1	1	1	9	3
numProcesses–	87	0	1	1	1	75	33
numProcesses++	87	0	1	1	1	76	34
newProcess.proc.priority	48	42	1	1	1	24	8
finishAllProcesses.i	89	3	1	1	1	77	35
finishAllProcesses.total	55	13	1	1	1	76	35
i	50	74	1	1	1	72	35
upgradeProcessPrio.n	57	73	0.19	0.93	0.6	72	33
unblockProcess.count	21	19	0.17	0.99	0.59	74	34
unblockProcess.n	94	93	0.18	0.99	0.41	72	43
unblockProcess.prio	48	49	0.84	0.96	0.96	74	36
quantumExpire.prio	71	33	0.86	0.94	0.96	72	34
initPrioQueue.i	73	13	1	1	1	9	4
marker	43	0	0.84	1	0.99	20	4
appendEle.memCount	52	28	no flows				
upgradeProcessPrio.count	83	78	0.25	0.94	0.71	72	33
main.prio	100	0	1	1	1	4	1
readInt.i	19	12	1	1	1	22	7
readFloat.i	89	0	1	1	1	21	7
findNth.i	86	14	1	1	1	72	33

6.1 The Relation between the Maximum Strength of All Paths Leading to a Variable and Its Predictability

For each integer variable from the *schedule* program, we provide in Table III the prediction accuracies obtained using stride and FCM predictors, the number of total flows into the variable, the number of flows that have zero strength using the three different measuring methods, and the strongest flow into the variable measured per method. We can tell from Table III that there is no direct relation between the maximum strength of flow into a variable and its predictability. Consider, for example, variables *job.val, numProcesses–, newProcess.proc.priority, and readInt.i*. These variables have the same values of the maximum strength of flow into them measured by the standard r, eta coefficient and normalized mutual information. However, there is significant difference in the predictability accuracies for all of these variables using both stride and FCM predictors.

To try to explain the prediction accuracies obtained using the predictors, we closely examined all the flows into each variable and how the variable is updated in the program. We concluded that the prediction accuracy of a target variable is affected by: (i) prediction accuracies of its source variables that have strong flow into it, and (ii) the program control flow, such as early return in a for loop before decrementing or incrementing the variable.

6.2 Variables Predictability and Strong Information Flow

We have provided in Sections 4.2 and 4.3 a mathematical analysis of the potential usage of standard r, eta coefficient and normalized mutual information in value

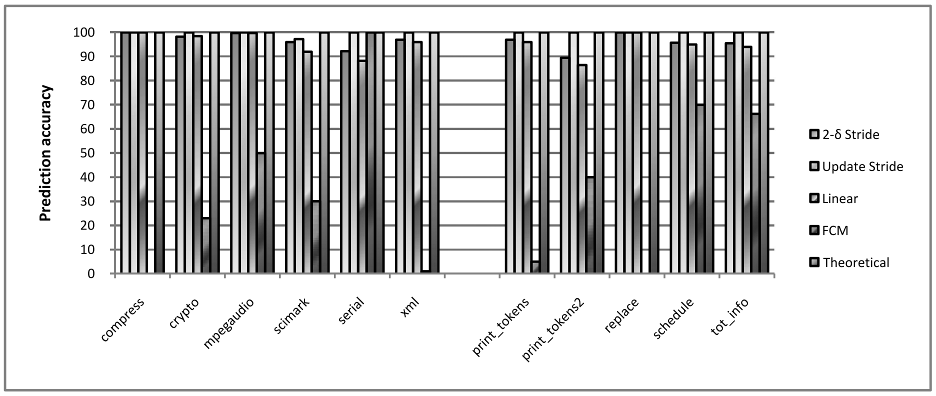

Fig. 3. Average prediction accuracies for variables of the self linear relation type.

prediction. We present in this section our empirical measurements that confirm our theoretical proofs. We have selected all numeric variables that have flows into them characterized by (a) $|r| = 1$, or (b) $|eta| = 1$ and $|r| < 1$, or (c) normalized mutual information equals 1, and both $|r|$ and $|eta| < 1$. We further categorize the flows as follows: (i) flow from the variable to itself at the same location in the program, or (ii) flow from a source variable into the target variable. We define:

— *self linear relation* type as the set of variables that satisfy both criteria (a) and (i);
— *distinct linear relation* type as the set of variables that satisfy both criteria (a) and (ii);
— *self eta nonlinear relation* type as the set of variables that satisfy both criteria (b) and (i);
— *distinct eta nonlinear relation* type as the set of variables that satisfy both criteria (b) and (ii).

6.2.1 Self Linear Relation Type. For the self linear relation type, Figure 3 shows the average prediction accuracies obtained using stride, update stride, linear, and FCM predictors. For all the benchmarks in Figure 3, at least one of the predictors used performed very well. This implies that variables of this type are highly predictable. Hence, this type can be used as a criterion to identify highly predictable variables.

6.2.2 Distinct Linear Relation Type. For the *distinct linear relation* type, it can be seen from Figure 4 that the global linear predictor, described in Section 5.1.2, achieves 100% prediction accuracy. The predictability accuracies at the source variables, obtained using local prediction techniques, are preserved within acceptable deviation at the target variables. This validates our mathematical analysis in Section 4.2.1.

Low prediction accuracies are achieved at the source and target variables using stride, linear and FCM predictors. We can explain this by the fact that variables that exhibit linear or repeating linear patterns, are part of the self linear relation type discussed in Section 6.2.1. In fact, if two variables are related with a linear relation and one of them follows a linear or repeating linear pattern, the second will follow the same pattern as proved in Section 4.2.1.

6.2.3 Self Eta Nonlinear Relation Type. For the self eta nonlinear relation type, Figure 5 presents the predictability accuracies using stride, linear, and FCM predictors. For *compress* benchmark, FCM predictor achieves very high prediction accuracies, close

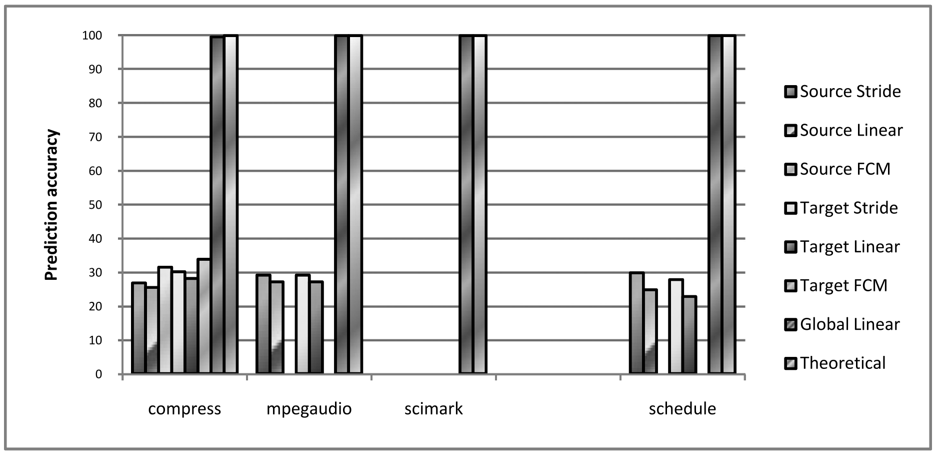

Fig. 4. Average prediction accuracies for variables of the distinct linear relation type.

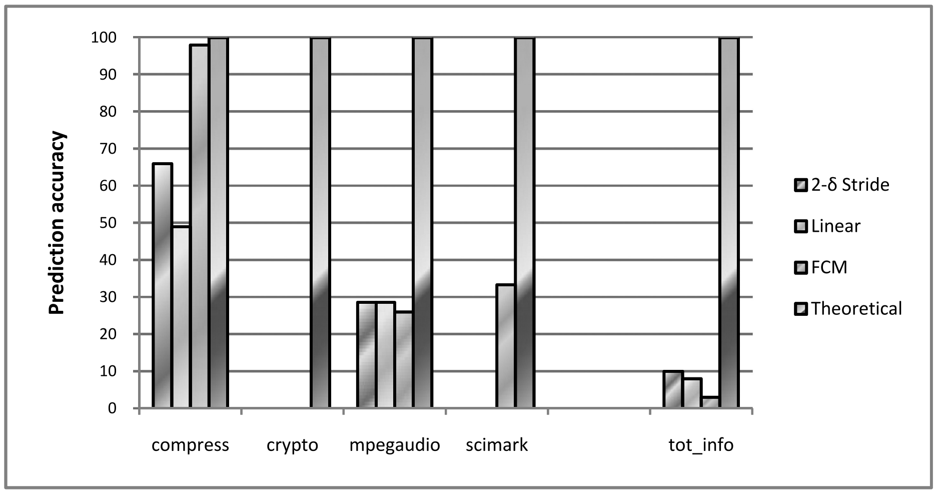

Fig. 5. Average prediction accuracies for variables of the self eta nonlinear relation type.

to 100%, and outperforms the other two predictors. Prediction accuracies measured using the three predictors are equal to 0 for *crypto* benchmark. The values taken by *crypto* variables that fall into this type are random. Stride and linear predictors perform slightly better than the FCM predictor for *mpegaudio*. This is due to the fact that some variables take random values interleaved by a sequence of the same value. FCM predictor accomplishes 33.33% for *scimark*, while the stride and linear predictors prediction accuracies are equal to 0. All three predictors do not perform well for *tot_info*.

6.2.4 Distinct Eta Nonlinear Relation Type. For the distinct eta nonlinear relation type, Figure 6 presents the predictability accuracies using stride, linear, and FCM predictors. It is clear from the results of benchmarks *compress* and *scimark*, that stride and linear patterns at the source are not preserved at the target. If the source follows a FCM pattern, such as in *scimark*, the target also follows a FCM pattern. This result confirms our analysis in Section 4.3.

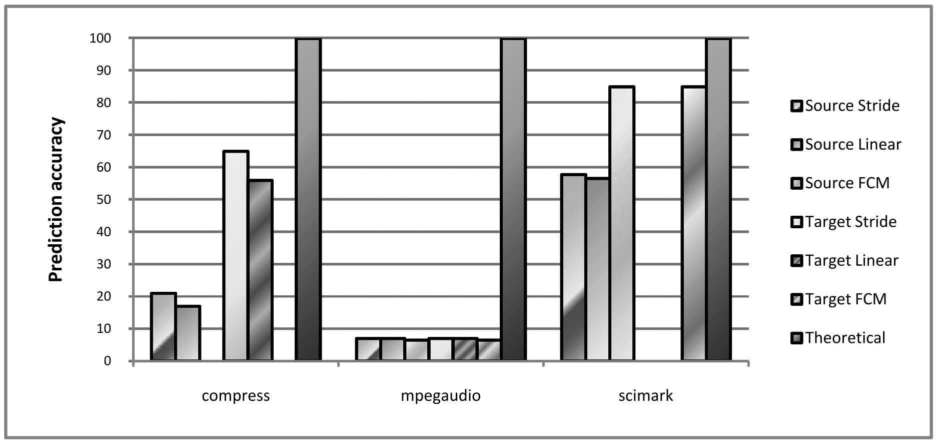

Fig. 6. Average prediction accuracies for variables of the distinct eta nonlinear relation type.

Each of Figures 3, 4, 5, and 6 reports only results for benchmarks that have variables that satisfy its corresponding type. Figures 3, 4, 5, and 6 show that 100% theoretical prediction accuracy is achieved. This is explained by the fact that the conditional entropy $H(X|Y)$ for variables of the corresponding types is 0. This implies that for:

— the self linear relation and self eta nonlinear relation types, knowing the value of the variable at the current state,we can predict its value at the next state with no uncertainty.
— the distinct linear relation and distinct eta nonlinear relation types, knowing the value of the source variable, we can predict the value of the target variable with no uncertainty.

6.2.5 Nonlinear Relation Captured via Normalized Mutual Information Only. We only observe dynamic dependence chains in *mpegaudio* benchmark that satisfy type (c)[7]. Most of the variables involved in these paths do not exhibit a known pattern; and hence they are not predictable using any of the existing value predictors. Had we known the nonlinear relation captured via normalized mutual information, we would have the potential to achieve high prediction accuracy.

6.3 Predictor Selection

We notice that random nonpredictable data results in zero strength flow measured by all of the three methods used. Linear and repeated linear patterns are caught by maximum strengths measured by all of the three methods used. FCM pattern, which is not a repeated linear pattern, is only caught by eta coefficient and normalized mutual information with maximum strength measured by both of them. We note that a nonlinear relation does not necessarily imply a FCM pattern.

Last Value Predictor. Variables that always take the same value which are identified using DIFA analysis and computation, as it can be seen from Figure 2, are predicted using a last value predictor.

[7]In most of the cases where normalized mutual information equals 1, $|r|$ and/or $|eta|$ are also equal to 1.

Update Linear Predictor. Variables of the *self linear relation* type are predicted using an update linear predictor. The update linear predictor reduces to an update stride predictor when the value of a in the linear relation equation equals 1.

Global Linear Predictor. Variables of the *distinct linear relation* type are predicted using a global linear predictor.

Finite Context Method (FCM) Predictor. When a nonlinear flow from the variable to itself is caught by absolute value of eta coefficient and/or normalized mutual information equal 1, a FCM predictor can be used to predict the variable values; though, it is not necessary that the variable follows a FCM pattern in this case.

For each variable, the above selection criterion allows the usage of the most appropriate predictor, which can potentially enhance the performance, for example, there is no need to employ a global linear predictor to predict a variable that can be predicted with a very high accuracy using one of the local predictors.

7. IMPLEMENTATION

DIFA identifies highly predictable variables and selects the appropriate predictor to use. This reduces the required hardware computations and achieves higher prediction accuracy. It also decreases the destructive impact of aliasing when it is present. Variables identification is done in software using DIFA instrumentation and profiling, and flow strength computation, and propagated to the hardware using special compiler added hints. Hint bits, encoded in the instruction, are used to communicate compiler analysis to the architecture. Several processors instruction set architectures (ISA) already support hint bits. As an example, Itanium ISA [Intel 2006] contains hint bits for branch prediction and memory locality. Researchers have proposed the usage of hint bits for value prediction [Zhao and Lilja 2004]. Implicit hint is an approach that encodes hints in the registers names without changing the ISA [Vandierendonck and Bosschere 2010].

In this section, we describe an implementation of the linear and update stride predictors, compare an implementation of DIFA-directed value predictor to conventional hardware only value predictor, and discuss the potential performance of DIFA-directed value predictors.

7.1 DIFA-Directed Local Prediction Techniques

A DIFA-directed local value predictor is a conventional predictor that predicts only variables selected by DIFA and identified using added hints. Besides the support required to encode the hints, which can be either explicit or implicit, a DIFA-directed value predictor does not entail any extra overhead compared to a similar conventional predictor.

7.2 Linear Predictor Implementation

7.2.1 Local Linear Predictor. Each entry in the prediction table contains five slots that hold the values of the coefficients a and b of the linear equation, and the last three values of the predicted variable. The predicted value is obtained from the previous value of the predicted variable using the linear equation.

7.2.2 Global Linear Predictor. Each entry in the prediction table contains six slots that hold the values of the coefficients a and b of the linear equation, and the last two values for both the source and target. When the source value is updated, the new value is stored in the corresponding entry in the table and the predicted value of the target is calculated using the linear equation. A given variable might be the source of more than one target variable.

For both global and local linear predictors, we initially check if the linear pattern is a stride pattern. In this case, a is set to 1, and b is assigned the value of the stride; otherwise, the value of a and b of the linear equation, $y = ax + b$, are calculated using the appropriate hardware as described in Section 4.2.1.

After it is resolved, the true value is stored in its corresponding entry in the table. If it matches with the predicted value, nothing is done. If it does not match, the values of a and b are re-calculated using the newly obtained value.

7.3 Update Stride Predictor Implementation

The implementation of the update stride predictor described in Section 5.2 is similar to that of a regular stride predictor with the addition of a hash function that maps the initialization of the variable to the table entry that corresponds to its regular update.

7.4 Metrics

We use the following three metrics in comparing the performance of using DIFA with that of hardware only.

(1) Prediction accuracy = number of successful predictions / total number of attempted predictions.
(2) Variables coverage = number of variables predicted / total number of encountered variables.
(3) Prediction coverage = number of a variable predictions / total number of a variable updates.

Variables coverage presents the percentage of the variables that are predicted. Prediction coverage presents the ratio of the number of a variable prediction to the number of its updates. It reflects the required learning period of a predictor, the set of variables that are not selected for confident prediction when confidence bits are used, and the aliasing effect since mapping distinct variables to the same table entry results in repeating the learning period each time the variable is allocated an entry in the table.

7.5 Implementation Issues

In this section, we explain the impact of aliasing and table size, and the usage of confidence bits. We also present our replacement policy.

7.5.1 Aliasing. At the assembly level, the lower bits of the program counter are used to access the prediction table. This can result in both constructive and destructive aliasing. At the Java level, we index the table using a unique id for each variable, and hence the aliasing effect is not present.

7.5.2 Confidence Bits. We associate with each table entry a two-bits saturating counter that is incremented on a hit and decremented on a miss. We only use the predicted value and account for the hit or miss when the value of the counter is greater than 1. Using the saturating bit counters, variables with 0 prediction accuracies are not considered for prediction since the value of the counter will always be less than 1. Also, variables with low prediction accuracies may not be considered for prediction, depending on the sequence of values taken by these variables.

7.5.3 Table Size. Table size affects prediction accuracy when aliasing is involved. The smaller the table size, the lower the prediction accuracy. In the absence of

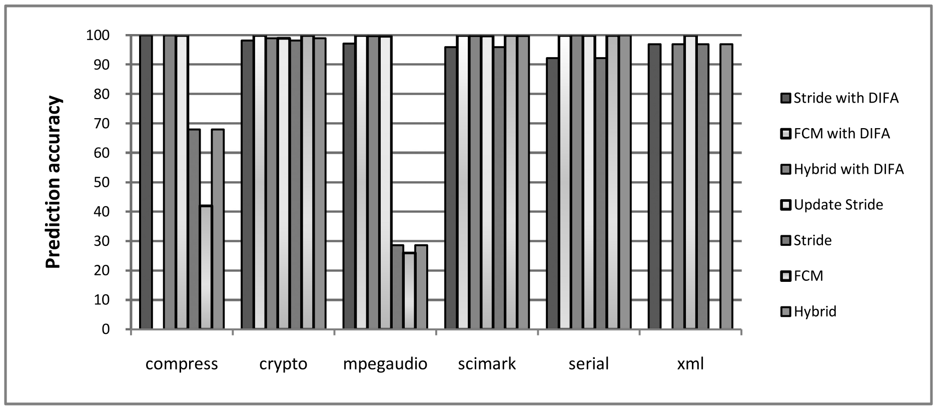

Fig. 7. Comparison of the average prediction accuracies with and without the usage of DIFA.

the destructive aliasing effect, the table size can affect the variables coverage and prediction coverage defined in Section 7.4. When DIFA is used, table size will have less impact, since only highly predictable variables are considered for prediction while using hardware only, all variables are considered.

7.5.4 Replacement Policy. At the assembly level, we use tagless direct mapped tables. At the Java level, we use tagged tables and a replacement algorithm that uses a combination of the confidence bits described above with least recently used (LRU) bits.

Replacing a highly predictable variable delays the prediction of the variable since after reencountering the variable and re-inserting it in the table, we need to account for the learning period of the predictor before we start predicting the value of the variable. This decreases the number of times the variable is predicted, and hence decreases its prediction coverage. In some situations, a highly predictable variable might be replaced while still in the warming phase, and hence its value is not predicted at all. This can decrease variables coverage defined in Section 7.4.

7.6 DIFA-Directed Value Predictors Results

7.6.1 Java Level Empirical Results. Taking into consideration the table size, we start incrementing the number of entries in the table and observe the changes in the values of variables coverage and prediction accuracy. Due to the absence of aliasing, the table size impact on the prediction accuracy is small. As we increase the table size, the variables coverage increases. For the same table size, higher variables coverage is generally achieved using DIFA than using hardware only, this is due to the fact that hardware only techniques consider all encountered variables which can result in removing variables from the table before being predicted to make place for newly encountered ones.

To provide a comparison of the accuracy of existing local value prediction techniques with and without the usage of DIFA, we select variables of the self linear relation type when DIFA is employed. As shown in Section 6.2.1, variables of the self linear relation type are highly predictable using local prediction techniques.

In Figures 7 and 8, using unlimited table size, we respectively present the average prediction accuracies and variables coverages obtained using stride, FCM and hybrid (stride, FCM) predictors with and without the usage of DIFA for SPECjvm2008 programs. We make use of the confidence bits described in Section 7.5.2. For each variable prediction, the hybrid predictor selects the appropriate predictor to use, either stride

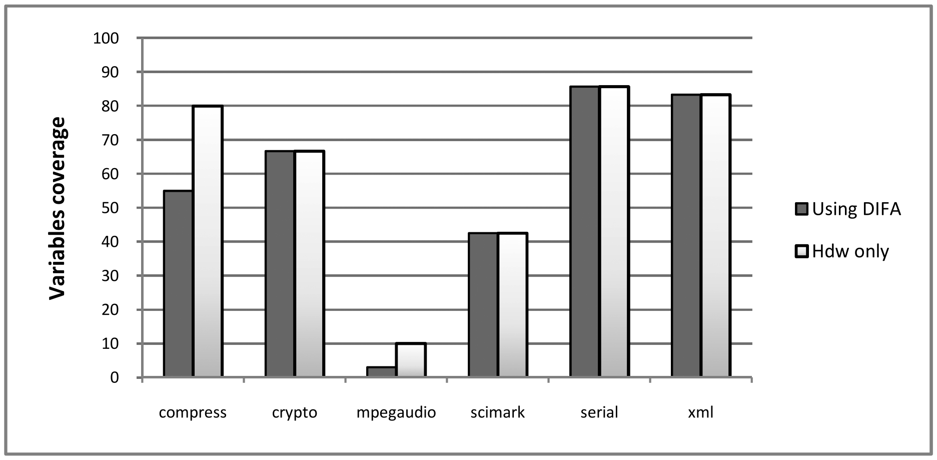

Fig. 8. Percentages of the number of variables predicted with and without the usage of DIFA.

or FCM. It has the advantage that it can predict more variables than each of the two predictors if used alone, and can achieve better prediction accuracy than each of them if used to predict all the variables it predicts. For *crypto*, *scimark*, *serial* and *xml*, the same prediction accuracies and variables coverages are accomplished with and without the usage of DIFA. This is due to the fact, that variables that are not selected via DIFA all have 0% prediction accuracies and are not considered for prediction since we are using the confidence bits. We note that the number of variables of these benchmarks is much less than that of *compress* and *mpegaudio*. For *compress*, using DIFA, stride predictor achieves 100%. None of the variables selected via DIFA is predicted using the FCM predictor since all of them follow a stride pattern. Stride and FCM predictors achieve 68% and 42% respectively without DIFA for *compress*. For *mpegaudio*, stride and FCM predictors achieve 97.2% and 100% respectively with DIFA, and 28.6% and 26% respectively without DIFA. For *compress* and *mpegaudio*, Figure 8 shows that higher variables coverages are accomplished using hardware only than with the usage of DIFA. This implies that more variables are predicted using hardware only, which explains the lower prediction accuracies achieved for these benchmarks when hardware only is used as can be seen from Figure 7. In fact, DIFA selects highly predictable variables. In case using hardware only, additional variables are predicted, this will normally result in a lower prediction accuracy.

Figure 7 shows that the update stride predictor either achieves the same prediction accuracy or outperforms the hybrid predictor that makes use of DIFA. It outperforms the hybrid predictor in situations where the initial value changes from an iteration to another, and hence the FCM predictor no longer achieves 100% accuracy. This is the case for *xml* as can be seen from Figure 7.

7.6.2 Assembly Level Empirical Results. We report in this section empirical results obtained from applying DIFA-directed value prediction to the C version of the Siemens benchmark suite run on x86 platform. We leverage the *Pin* dynamic binary instrumentation tool [Luk et al. 2005] to collect the results. An unlimited table size is assumed for the results reported in Figures 9, 10, 11, and 12.

From Figure 9, it can be seen that the last value predictor (LV) achieves 100% prediction accuracy and 87.92% prediction coverage on average for the selected

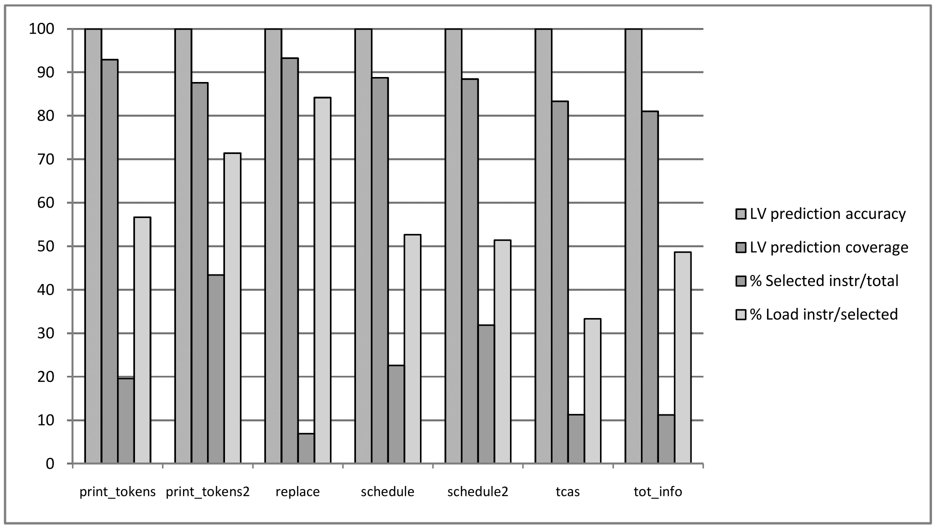

Fig. 9. Constant outcome instructions measurements.

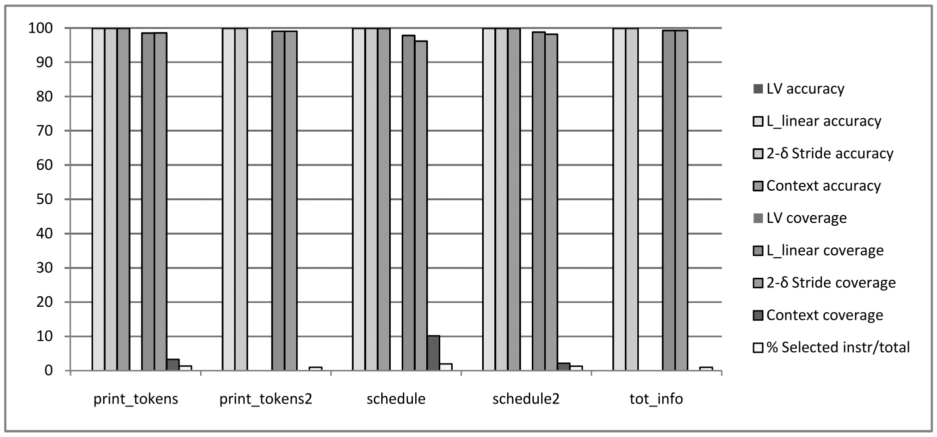

Fig. 10. Instructions with linear relation from the instruction to itself measurements.

constant outcomes instructions[8]. These instructions account for 21% on average of the totally dynamically executed value producing instructions. Load instructions constitute 56.92% on average of these selected instructions.

From Figure 10, it can be seen that at least one of the local value predictors achieves 100% prediction accuracy for instructions of the *self linear relation* type. A high prediction coverage, 98.67% on average, is accomplished. Instructions of this category present small portion, 1.11% on average, of the totally dynamically executed value producing instructions. The LV predictor achieves 0%. This is due to the fact that instructions that follow last value pattern are considered constant outcomes instructions and selected for the measurements reported in Figure 9.

From Figure 11, it can be seen that the global linear predictor achieves 99.58% prediction accuracy and 93.42% prediction coverage on average for instructions of the

[8]We define constant outcome instruction as the instruction that always produces the same value.

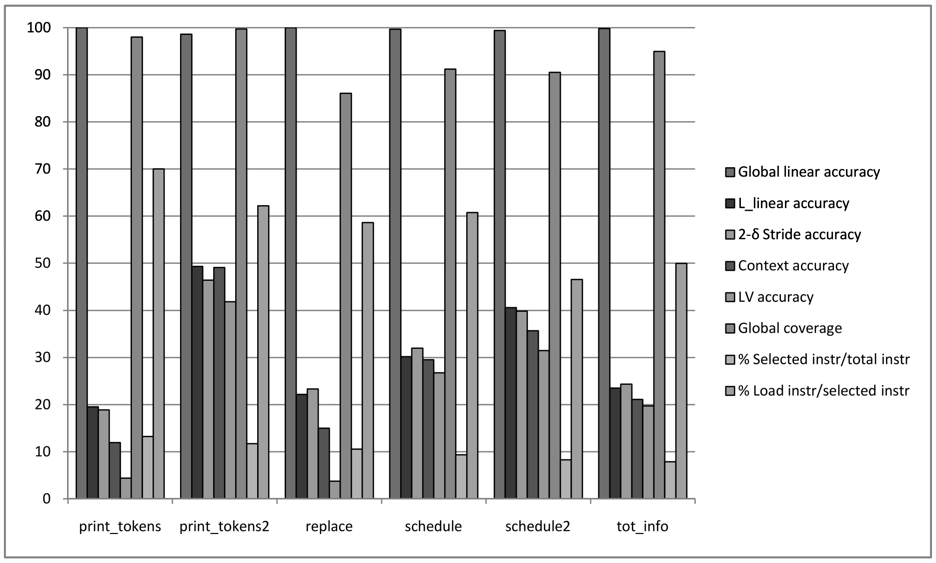

Fig. 11. Global linear predictor measurements.

distinct linear relation type. It significantly outperforms all local predictors. Instructions of this category present 10.18% on average of the totally dynamically executed value producing instructions. Load instructions constitute 58.03% on average of these instructions. We note that 74.32% on average of the instructions of the distinct linear relation type are separated by a dynamic distance[9] greater than 10. We can select instructions that are separated by a dynamic distance large enough to overcome the global predictor latency.

We implement a combined DIFA-directed value predictor that uses three distinct predictors: last value, local and global linear predictors. Each predictor is assigned a set of instructions to predict. The sets of instructions assigned to the predictors are mutually exclusive, i.e. they do not have any common element. The last value predictor predicts constant outcome instructions. The local linear predictor predicts instructions of the self linear relation type. The global linear predictor predicts instructions of the distinct linear relation type.

From Figure 12, it can be seen that the combined DIFA-directed value predictor achieves better average prediction accuracy (99.89% vs. 92.26% for STR, and 88.55% for DFCM), and prediction coverage (89.23% vs. 56.9% for STR, and 41.45% for DFCM) than the 2-delta stride and DFCM predictors.

8. POTENTIAL PERFORMANCE OF DIFA-DIRECTED PREDICTORS

To observe the effects of the proposed DIFA-directed value prediction techniques on the processor performance, we use the *PTLsim* simulator [Yourst 2007]. We implement value prediction inside *PTLsim* and simulate all of the Siemens benchmarks.

The usage of value prediction results in *predicted* and *speculated* values. Predicted value is the outcome of a value predictor. Speculated value is the outcome of an instruction that has one or more predicted or speculated operands [Sazeides 2002]. We only execute branch and memory instructions with valid operands.

[9]We define dynamic distance between two distinct instructions as the number of instructions dynamically executed between their executions.

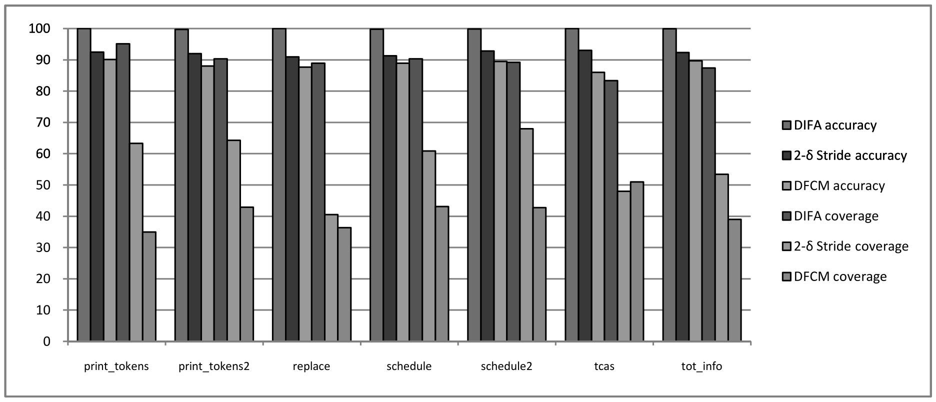

Fig. 12. Combined DIFA-directed value predictor measurements.

We use the *PTLsim redispatch* recovery mechanism to handle value misprediction. The redispatch mechanism finds the set of uops that depend on a mis-speculated uop and send them back to the ready-to-dispatch state. This mechanism is known as *reexecute* recovery. We predict at the dispatch stage and update the predictors when the valid values become available.

Loads that are highly predictable using local last value or stride predictors, are also predictable using the DIFA-directed last value or local linear predictors. In addition, as can be seen from Figure 11, DIFA predictor is capable of predicting with very high accuracy, using the global linear predictor, loads that are not predictable using any of the existing local prediction techniques.

DIFA predictor does not require a predictor selector that might present a source of error. It also does not suffer from negative interference that might exist between hybrid predictor components [Burtscher and Zorn 2002]. Also, since each set of selected instructions is predicted using a specific predictor, smaller prediction tables can be used than the tables that are required when a predictor is used to predict all encountered instructions or all instructions of a specific type such as load instructions. Computation and large prediction table are two sources of predictor complexity [Sam and Burtscher 2005]. Using smaller prediction table, making use of the simple last value predictor, and avoiding the usage of hybrid predictors help reducing the complexity of the DIFA predictor. Reducing table sizes and using highly accurate predictors help reducing energy consumption [Sam and Burtscher 2005]. Also, since the set of instructions assigned to the three predictors used are disjoint, there is no shared performance contribution[10] between any pair of the three used predictors. This means, none of them is doing redundant computations. All of the benchmarks were run for 1 billion committed instructions with the configuration parameters for our simulations given in Table IV.

From Figure 13, it can be seen that the combined DIFA-directed value predictor shows significant performance potential. It outperforms both 2-delta stride and DFCM predictors. This can be attributed to its capability of exploiting computational locality in both local and global value histories, identifying highly predictable instructions, and assigning them to the appropriate predictors that operate disjointly.

[10]Shared performance contribution is provided by either two or all of the three predictors used. Using all of the involved predictors will not increase this contribution [Burtscher and Zorn 2002].

Table IV. Parameters for Our Architectural Configuration

Machine width	4-wide fetch, 4-wide issue, 4-wide commit
Instruction L1 Cache	32KB 4-way, 2 cycles latency
Data L1 Cache	16KB 4-way, 2 cycles latency
L2 Cache	256KB 16-way, 6 cycles latency
L3 Cache	4MB 32-way, 16 cycles latency
Mem Latency	140 cycles
Branch Predictor	Combined bimodal and gshare, 64K meta, 64K bimodal, 64K gshare, 4K BTB
2-delta stride	16 K entries
DFCM	first level 16 K entries, second level 64 K entries
Combined DIFA	2K entries for local and global linear predictors, 4K entries for LV predictor

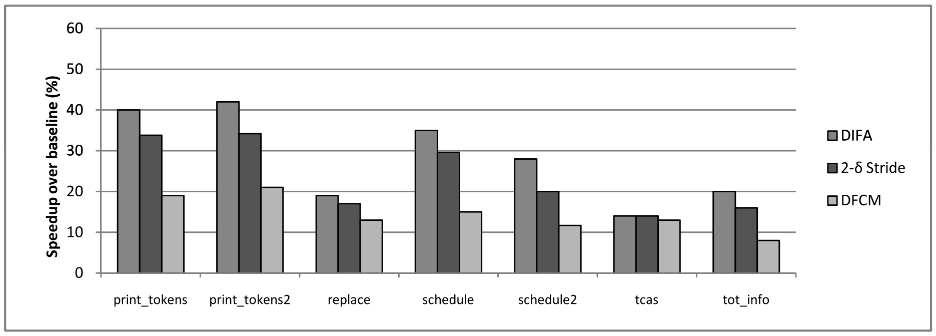

Fig. 13. Value prediction speedups.

9. FUTURE WORK

Following are some related research questions and topics we will investigate in future work.

(1) We are interested in exploring the possibilities of using DIFA in branch prediction [Smith 1981], control reconvergence [Ghandour 2009] and memory disambiguation [Ellis 1986]. We are going to look into identifying the set of branches on which a given branch depends strongly. We are interested in determining the optimum history size, either global or local, that needs to be used to accomplish maximum prediction accuracy for a given branch. The intention is to use an adaptive branch predictor size that can be defined per instruction. Identification of correlated branches [Thomas et al. 2003] and dynamic per-branch history length adjustment [Kwak and Jhon 2007] were previously addressed using hardware.

(2) Can we exploit paths that have zero flow strength to increase the effective ILP? Such paths are dependences where the source variables do not matter in the computation of the destination variables, thus making them irrelevant dependences to the execution. What type of hardware mechanisms could be implemented to exploit zero strength paths?

10. CONCLUSION

We conducted a study to determine the usability and benefits of correlating data value predictors. We used both analytical and empirical approaches in our study and employed dynamic information flow analysis. We derived upper bound on value prediction accuracy using information theory. We presented and proved theorems that relate variable value prediction and pattern preservation to the type of the relation between consecutive values of the same variable or between source and target variables related with strong dependence chain. Our empirical study confirmed our analytical proofs. We concluded from our empirical study that (1) there is no direct relation between the maximum strength of flow into a variable and its predictability, (2) the prediction accuracy of a given variable is associated with the way this variable is manipulated in the program, (3) there exists a relation between the predictability of a target variable and the predictability of other variables which are sources of strong flow paths to the target variable. We presented and simulated the update stride predictor. We introduced and simulated the global and local linear value predictors. We proved and experimentally showed that, using a global linear predictor, we can predict with 100% accuracy the values of a target variable knowing the values of another variable which is the source of a strong flow into the target variable. We explained when a given value predictor performs well and outperforms other predictors. We provided a selection criteria that can be used to identify highly predictable variables. This can significantly reduce value misprediction. We compared a hardware value prediction implementation that uses DIFA to one that does not, and found that DIFA directed prediction outperforms hardware only prediction on all our benchmarks, sometimes by a significant amount. We showed, using *PTLsim*, the potential of DIFA-directed value prediction in enhancing processor performance.

REFERENCES

AHUJA, P. S., SKADRON, K., MARTONOSI, M., AND CLARK, D. W. 1998. Multi-path execution: Opportunities and limits. In *Proceedings of the 12th International Conference on Supercomputing.* 101–108.

AKKARY, H. AND DRISCOLL, M. A. 1998. A dynamic multithreading processor. In *Proceedings of the Annual ACM/IEEE International Symposium on Microarchitecture*.

AKKARY, H., JOTHI, K., RETNAMMA, R., NEKKALAPU, S., HALL, D., AND SHAHIDZADEH, S. 2008. On the potential of latency tolerant execution in speculative multithreading. In *Proceedings of the International Forum on Next-Generation Multicore/Manycore Technologies*.

AKKARY, H., SRINIVASAN, S. T., AND LAI, K. 2003. Recycling waste: Exploiting wrong-path execution to improve branch prediction. In *Proceedings of the International Conference on Supercomputing*. 12–21.

AL-ZAWAWI, A. S., REDDY, V. K., ROTENBERG, E., AND AKKARY, H. 2007. Transparent Control Independence (TCI). In *Proceedings of the 34th Annual International Symposium on Computer Architecture*.

ALLISON, P. D. 1998. *Multiple Regression: A Primer*. Sage Publications Inc.

AUSTIN, T. M. AND SOHI, G. S. 1992. Dynamic dependency analysis of ordinary programs. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*. 342–351.

BISHOP, M. 2002. *Computer Security: Art and Science*. Addison-Wesley Professional.

BURTSCHER, M. AND ZORN, B. G. 1999. Exploring last n value prediction. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*. 66–76.

BURTSCHER, M. AND ZORN, B. 2002. Hybrid load-value predictors. *IEEE Trans. Comput. 51*, 759–774.

BUTLER, M., YEH, T., PATT, Y., ALSUP, M., SCALES, H., AND SHEBANOW, M. 1991. Single instruction stream parallelism is greater than two. In *Proceedings of the 18th Annual International Symposium on Computer Architecture*. 276–286.

CALDER, B., REINMAN, G., AND TULLSEN, D. M. 1999. Selective value prediction. In *Proceedings of the 26th Annual International Symposium on Computer Architecture*.

CHRYSOS, G. Z. AND EMER, J. S. 1998. Memory dependence prediction using store sets. In *Proceedings of the 25th Annual International Symposium on Computer Architecture*.

COVER, T. M. AND THOMAS, J. A. 1991. *Elements of Information Theory*. Wiley, New York.

DENNING, D. E. 1982. *Cryptography and Data Security*. Addison-Wesley.

DO, H., ELBAUM, S., AND ROTHERMEL, G. 2005. Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. *Empir. Softw. Engin. 10*, 4, 405–435.

EICKEMEYER, R. J. AND VASSILIADIS, S. 1993. A load instruction unit for pipelined processors. *IBM J. Resear. Devel. 37*, 4, 547–564.

ELLIS, J. R. 1986. *Bulldog: A Compiler for VLIW Architectures*. MIT Press.

FANO, R. M. 1961. *Transmission of Information: A Statistical Theory of Communications*. MIT Press.

FRANKLIN, M. 1993. The multiscalar architecture. Ph.D. thesis, University of Wisconsin.

GABBAY, F. AND MENDELSON, A. 1997. Can program profiling support value prediction? In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*. 270–280.

GANDHI, A., AKKARY, H., AND SRINIVASAN, S. 2004. Reducing branch misprediction penalty via selective branch recovery. In *Proceedings of the 10th International Symposium on High-Performance Computer Architecture*.

GANUSOV, I. AND BURTSCHER, M. 2006. Future execution: A prefetching mechanism that uses multiple cores to speed up single threads. *ACM Trans. Archit. Code Optim*. 3, 424–449.

GHANDOUR, W. J. 2009. Dynamic control independence predictor for speculative multithreading processors. *Contemp. Engin. Sci. 2*, 517–534.

GHANDOUR, W. J., AKKARY, H., AND MASRI, W. 2010. The potential of using dynamic information flow analysis in data value prediction. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'10)*. 431–442

GOEMAN, B., VANDIERENDONCK, H., AND BOSSCHERE, K. D. 2001. Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. In *Proceedings of the 7th International Symposium on High-Performance Computer Architecture.* 207–216.

HU, S., BHARGAVA, R., AND JOHN, L. K. 2003. The role of return value prediction in exploiting speculative method-level parallelism. *J. Instruct.-Lev. Paral. 5*, 1–21.

HWU, W. W. 1995. Compiling for ilp processors. *Proc. IEEE 83*, 12.

INTEL. 2006. Intel Itanium architecture software developers manual. Document number: 245319-005.

KACHIGAN, S. 1986. *Statistical Analysis: An Interdisciplinary Introduction to Univariate and Multivariate Methods*. Radius Press.

KIRMAN, N., KIRMAN, M., CHAUDHURI, M., AND MARTINEZ, J. F. 2005. Checkpointed early load retirement. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*. 16–27.

KRUS, D. J. 2006. *Visual Statistics with Multimedia*. Chapter 14, visualstatistics.net.

KWAK, J. W. AND JHON, C. S. 2007. Dynamic per-branch history length adjustment to improve branch prediction accuracy. *Microprocess. Microsyst. 31*, 1, 63–76.

LAM, M. S. AND WILSON, R. P. 1992. Limits of control flow on parallelism. In *Proceedings of the 19th Annual International Symposium on Computer Architecture*. 46–57.

LIPASTI, M., WILKERSON, C., AND SHEN, J. 1996. Value locality and load value prediction. In *Proceedings of the 7th Conference on Architectural Support for Programming Languages and Operating Systems*. 138–147.

LIPASTI, M. H. AND SHEN, J. P. 1996. Exceeding the dataflow limit via value prediction. In *Proceedings of the 29th Annual ACM/IEEE International Symposium and Workshop on Microarchitecture*. 226–237.

LUK, C., COHN, R., MUTH, R., PATIL, H., KLAUSER, A., LOWNEY, G., WALLACE, S., REDDI, V. J., AND HAZELWOOD, K. 2005. Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'05)*.

MARCUELLO, P., TUBELLA, J., AND GONZELEZ, A. 1999. Value prediction for speculative multithreaded architectures. In *Proceedings of the 32nd International Symposium on Microarchitecture*.

MARCUELLO, P., GONZALEZ, A., AND TUBELLA, J. 2004. Thread partitioning and value prediction for exploiting speculative thread-level parallelism. *IEEE Trans. Comput. 53*, 2, 114–125.

MASRI, W. AND PODGURSKI, A. 2006. An empirical study of the strength of information flows in programs. In *Proceedings of the International Workshop on Dynamic Analysis*. 73–80.

MASRI, W. AND PODGURSKI, A. 2009a. Algorithms and tool support for dynamic information flow analysis. *Inf. Softw. Technol. 51,* 2, 385–404.

MASRI, W. AND PODGURSKI, A. 2009b. Measuring the strength of information flows in programs. *ACM Trans. Softw. Engin. Methodol. 19*, 2.

MASRI, W., PODGURSKI, A., AND LEON, D. 2004. Detecting and debugging insecure information flows. In *Proceedings of the 15th IEEE International Symposium on Software Reliability Engineering*.

MASRI, W., PODGURSKI, A., AND LEON, D. 2007. An empirical study of test case filtering techniques based on exercising information flows. *IEEE Trans. Softw. Engin. 33*, 454–477.

MOHAN, W. AND FRANKLIN, M. 1992. Improving data value prediction accuracy using path correlation. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems*. 76–84.

MONTOYE, R. K., HOKENEK, E., AND RUNYON, S. L. 1990. Design of the ibm risc system/6000 floating-point execution unit. *IBM J. Resear. Devel. 34*, 1, 59–70.

MOSHOVOS, A. I. 1998. Memory dependence prediction. Ph.D. thesis, Computer Sciences Department, University of Wisconsin, Madison.

MOSHOVOS, A. AND SOHI, G. S. 2002. Reducing memory latency via read-after-read memory dependence prediction. *IEEE Trans. Comput. 51*, 3, 313–326.

MOSHOVOS, A., BREACH, S. E., VIJAYKUMAR, T. N., AND SOHI, G. 1997. Dynamic speculation and synchronization of memory dependences. In *Proceedings of the 24th Annual ACM/IEEE Conference on Computer Architecture*.

NAKRA, T., GUPTA, R., AND SOFFA, M. L. 1999. Global context-based value prediction. In *Proceedings of the 5th International Symposium on High-Performance Computer Architecture*.

OPLINGER, J., HEINE, D., AND LAM, M. 1999. In search of speculative thread-level parallelism. In *Proceedings of the 8th International Conference on Parallel Architectures Compilation Techniques*.

PAPWORTH, D. B. 1996. Tuning the pentium pro microarchitecture. *IEEE Micro 16*, 8–15.

PICKETT, C. J. F. AND VERBRUGGE, C. 2004a. Compiler analyses for improved return value prediction. Tech. rep. SABLETR-2004-6, Sable Research Group, School of Computer Science, McGill University.

PICKETT, C. J. F. AND VERBRUGGE, C. 2004b. Return value prediction in a java virtual machine. In *Proceedings of the 2nd Value-Prediction and Value-Based Optimization Workshop*.

RYCHLIK, B., FAISTL, J., KRUG, B., AND SHEN, J. P. 1998. Efficacy and performance impact of value prediction. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'98)*. 148–154.

SAM, N. B. AND BURTSCHER, M. 2005. Complex load-value predictors: Why we need not bother. In *Proceedings of the 4th Annual Workshop on Duplicating, Deconstructing, and Debunking*. 16–24.

SAZEIDES, Y. 2002. Modeling value prediction. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*.

SAZEIDES, Y. AND SMITH, J. 1997a. Implementations of context-based value predictors. Tech. rep., ECE-TR-97-8, University of Wisconsin, Madison.

SAZEIDES, Y. AND SMITH, J. E. 1997b. The predictability of data values. In *Proceedings of the 30th International Symposium on Microarchitecture*.

SAZEIDES, Y., VASSILIADIS, S., AND SMITH, J. 1996. The performance potential of data dependence speculation and collapsing. In *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*.

SIEGEL, S. 1956. *Nonparametric Statistics for the Behavioral Sciences*. McGraw-Hill.

SINGER, J. AND BROWN, G. 2006. Return value prediction meets information theory. In *Proceedings of the 4th International Workshop on Quantitative Aspects of Programming Languages (QAPL'06)*. 137–151.

SMITH, A., NAGARAJAN, R., SANKARALINGAM, K., MCDONALD, R., BURGER, D., KECKLER, S. W., AND MCKINLEY, K. S. 2006. Dataflow predication. In *Proceedings of the 39th Annual ACM/IEEE International Symposium on Microarchitecture*. 89–102.

SMITH, J. E. 1981. A study of branch prediction strategies. In *Proceedings of the Annual International Symposium on Computer Architecture*. 135–148.

SMITH, J. E. AND SOHI, G. S. 1995. The microarchitecture of superscalar processors. *Proc. IEEE 83*, 12, 1609–1624.

SODANI, A. AND SOHI, G. 1997. Dynamic instruction reuse. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*.

SODANI, A. AND SOHI, G. 1998. Understanding the differences between value prediction and instruction reuse. In *Proceedings of the 31st Annual ACM/IEEE International Symposium on Microarchitecture*. 205–215.

STEFFAN, J., COLOHAN, C., ZHAI, A., AND MOWRY, T. 1999. Improving value communication for thread-level speculation. In *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*.

THOMAS, R. AND FRANKLIN, M. 2001. Using dataflow based context for accurate value prediction. In *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT'01)*. 107–117.

THOMAS, R., FRANKLIN, M., WILKERSON, C., AND STARK, J. 2003. Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*. 314–323.

TUCK, N. AND TULLSEN, D. M. 2005. Multithreaded value prediction. In *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*.

TULLSEN, D. AND SENG, J. 1999. Storageless value prediction using prior register values. In *Proceedings of the 26th International Symposium on Computer Architecture.*

VANDIERENDONCK, H. AND BOSSCHERE, K. D. 2010. Implicit hints: Embedding hint bits in programs without ISA changes. In *Proceedings of the IEEE International Conference on Computer Design*. 364–369.

WANG, K. AND FRANKLIN, M. 1997. Highly accurate data value prediction using hybrid predictors. In *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*. 281–290.

WARTER, N., LAVERY, D., AND HWU, W. 1993. The benefit of predicated execution for software pipelining. In *Proceedings of the 26th Hawaii International Conference on System Sciences*. 497–506.

YOURST, M. T. 2007. Ptlsim: A cycle accurate full system x86-64 microarchitectural simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*. 23–34.

ZHAO, Q. AND LILJA, D. J. 2004. Static classification of value predictability using compiler hints. *IEEE Trans. Comput. 53*, 929–944.

ZHOU, H., FLANAGAN, J., AND CONTE, T. M. 2003. Detecting global stride locality in value streams. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA'03)*. 324–335.

Received October 2010; revised April 2011, July 2011; accepted September 2011