The Potential of Using Dynamic Information Flow Analysis in Data Value Prediction

Walid J. Ghandour, Haitham Akkary and Wes Masri
Electrical and Computer Engineering Department
American University of Beirut
Beirut, Lebanon 1107 2020
{wjg00, ha95, wm13} @aub.edu.lb

ABSTRACT

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 and future outputs of an instruction are predicted based on outputs from previous executions of the same instruction.

In this paper, we explore the possibility of introducing highly accurate general **correlating** value predictor using dynamic information flow analysis. We use information theory to mathematically prove the validity and benefits of correlating value predictors. We also introduce the concept of **linear** value predictors, a new technique that predicts a new value from another one using a linear relation. We then conduct empirical analysis using programs from 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. Furthermore, we provide a scheme to select highly predictable variables, and explain when a specific value predictor can perform well and even outperform other predictors.

Using a dynamic information flow analysis tool called *DynFlow*, we show that the values of a set of selected variables can be predicted with very high accuracy, up to 100%, from previous history of the same variables or other variables that have strong information flow into the predicted variable.

Categories and Subject Descriptors

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

General Terms

Design, Performance, Theory

Keywords

computer architecture, correlation, dynamic information flow analysis, information flow strength, information theory, instruction level parallelism, program dependence analysis, value prediction.

1. INTRODUCTION

The availability of large number of transistors on chip has enabled the computer industry to move to many-core architectures [51],[33]; however, not all programs may have enough parallelism to make use of the available parallel cores. As the number of cores increases, communication latency increases and communication overhead may dominate execution time [31],[50]. Value prediction techniques [41],[39],[64] speculate the outcome of an instruction and allow data dependent instructions to issue and execute in parallel. In addition to its potential usage in overcoming computation latency, value prediction could also be used with software or hardware parallelization techniques to eliminate communication latency, within the same core or between different cores in a many-core architecture, when values are successfully predicted.

Most of the existing general value prediction techniques are *local* (i.e., instruction-based). A new value of a variable is predicted using its previous history. The majority of previous work in the area of value prediction has focused on local prediction schemes to increase prediction accuracy and reduce required hardware overhead. A specific case of correlating value prediction that focuses on correlation between procedure parameters and return value is addressed in [52],[15],[14]. A scheme that correlates the prediction of the value of an instruction with that of the immediately preceding instruction is provided in [57].

In this paper, we introduce for the first time general correlating value predictors that predict the value of a given variable using the values of one or more other variables. We attempt to increase value prediction accuracy by exploiting correlation that might exist between arbitrary pairs of variables. We also present a scheme that can be used to identify highly predictable variables. Using information theory, we derive mathematical proofs and conditions for the predictability of data values. To demonstrate the validity of our theoretical analysis, we use empirical studies based on *dynamic information flow analysis* (DIFA), a program analysis technique that monitors information flow among variables in an executing program [58]. The chain of dynamic dependences through which information can flow from a variable to another is called information channel and the amount of information that flows through a channel is quantified by what is called information flow *strength*.

1.1 Paper Objective

We attempt to answer the following questions in this paper:

1. What advantages can a correlation based predictor give over existing instruction-based predictors?
2. Can we make use of dynamic information flow analysis to provide a measure of predictability confidence?
3. Can we make some observations on the predictability of a variable given the maximum strength of all paths leading to it?
4. Can we make some observations on the predictability of a target variable knowing the predictability of source variables that have strong information flow into it?
5. Can we use the schemes that capture the strength of information flow to select appropriate value predictors?
6. Which of the schemes that measure dynamic information flow strength are more relevant to value prediction?

1.2 Paper Contributions

This paper makes the following contributions:

1. Uses dynamic information flow analysis, information theory, and correlation-based techniques in value prediction.
2. Studies the advantage of correlating value predictors over instruction-based value predictors.
3. Derives upper bounds of prediction accuracy rates for correlating and instruction-based value predictors.
4. Proves mathematically that when a linear relation exists between two different variables, stride[1] and Finite Context Method (FCM)[2] patterns are preserved, and the linear relation can be used to compute the value of the target variable using that of the source.
5. Shows that when a non-linear relation exists between two different variables, a FCM pattern is preserved and the non-linear relation can be used to compute the value of the target using that of the source.
6. Introduces linear predictors. A linear predictor uses the linear relation between the values of two different variables to predict the value of one of them knowing that of the other, or the linear relation between consecutive values of the same variable to predict its next value from the current one. A stride predictor is a special case of this predictor.
7. Gives an explanation about when a given value predictor can perform well and even outperform other predictors.
8. Provides a scheme that can be used to identify highly predictable variables.

[1]In a stride sequence, an element is obtained from the previous one by adding to it a constant delta called stride.

[2]A Finite Context Method (FCM) predictor "matches recent value history (context) with previous value history and predict values based entirely on previously observed patterns"[64].

1.3 Paper Overview

The rest of this paper 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 discusses the experimental work that we have done. Section 6 addresses applications and implementation of the proposed value prediction schemes. Section 7 proposes interesting research directions we plan to address in future work. Finally, we conclude in Section 8.

2. BACKGROUND AND PREVIOUS 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 (anti-dependence) 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 [7],[43], iii) multipath execution [46], iv) parallel execution of control-independent code portions [23],[10],[25], and v) misprediction penalty reduction [6],[26], [3].

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 dependence and memory dependence. 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. A

Memory-order violation occurs when a load depending on a store reads the memory location before a store writes to it. This can be avoided by preventing loads from executing until all prior stores have executed. This solution decreases performance by creating false dependences and unnecessarily delaying loads from executing. Memory dependence prediction is a solution that [4],[5],[11],[16] i) predicts load instructions that will cause memory-order violation if allowed to execute, and ii) delays their executions 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 [48], ii) executing independent instructions in parallel such as in ILP compilers [63] and dynamically scheduled processors [27], and iii) speculating on the results of data-generating instructions. Limited parallelism can be achieved using the second technique as shown in [56],[38],[35]. The third technique, *data value prediction*, is the topic of this paper. 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 that predicts the result of an instruction based on its past behavior. By predicting instruction results early and passing them to dependent instructions, DVP reduces the length of critical paths through a program [64]. 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 [39].

Prediction schemes can be either computational or context based [64]. A computational predictor yields a predicted next value by performing an operation on previous values such as *stride value* (STR) predictor [65] and *last value* (LV) predictor[3] [41]. 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 predict a new value. Newly computed stride is stored in s_2. Stride s_1 is only updated with s_2 when s_2 occurs twice in a row [47]. 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 [54]. *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 potential to be more space efficient and faster to warm up than FCM [12]. The *register value* predictor in [19] always assumes that the target register of a load instruction contains the value to be loaded. The *last four value* predictor [37] stores the four most recently seen values. It is composed of four independent last value predictors with a meta-predictor that selects one of them. A pattern based two-level prediction scheme that makes use of the last four distinct values is introduced in [34]. Two different hybrid predictors, (two-level, stride) and (stride, last value), are proposed in [34]. A hybrid (stride, FCM) predictor is presented in [13].

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

We call the value prediction schemes described above local or instruction-based value predictors since they store in their history tables information about the values seen by individual instruction operands. Other value predictors that consider non-local execution history to make value predictions have been proposed as well. A study of the effect of control flow correlation schemes on stride predictors is provided in [62]. A *path identity correlation* based stride predictor is proposed in [62]. *Path-based last value* and *per-path stride* prediction techniques are presented in [57]. A scheme that correlates the prediction of the value of an instruction with that of the immediately preceding instruction is provided in [57].

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.

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) [8]. The reader is referred to [9] for a detailed comparison between DIR and value prediction.

The previous techniques were developed in the framework of superscalar architectures. Several efforts such as in [10],[44],[45],[28],[32] and [42] have addressed the usage of value prediction in the framework of thread-level speculation and simultaneous multithreading.

Speculative method-level parallelism (SMLP) is a speculative multithreading (SpMT) variant in which speculative threads are spawned at method call boundaries. Return value prediction is addressed in [52],[15],[14],[29],[32]. It is shown in [52] that SMLP can significantly benefit from accurate return value prediction.

The work presented in [29] applies the principles of information theory to return value prediction. It focuses on measuring the uncertainty in the next return value knowing the current value. It uses Fano's inequality [17],[22] to define a lower bound on misprediction rate.

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[4], leading to what is called *strength-based dependence analysis*, is presented in [61], [59].

Strength-based dependence analysis could have a pivotal role in enhancing data value prediction. With strength-based dependence analysis, strong chains of dynamic dependences could be exploited. *The prediction of the future value of a variable v could be based on the values of the variables that v strongly depends on. DynFlow* is a tool that supports strength-based dependence analysis for Java programs [58], [60]. It calculates flow strength at the bytecode level and maps it to the Java level. *DynFlow* uses three techniques for measuring information flow strength. The first employs classical information theory [17], and makes use of the entropy-based information flow characterization described in [20] and [36]. 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 [53]. The third technique uses the *eta* coefficient [55],[18], which provides the same value as *standard r* for linearly related variables and a greater value for non-linearly related variables. More details about information theory, *standard r* and *eta* coefficient are provided below in sections 2.4, 2.5.1 and 2.5.2, respectively.

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) [17]. Next, we define two information theoretic terms that are relevant to our work, namely entropy and mutual information.

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 [20],[36]:

$$H(X) = -\sum_i P(X = x_i) \log_2 P(X = x_i).$$

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:
$H(X|Y) = -\sum_j P(Y = y_j) \sum_i P(X = x_i|Y = y_j) \log_2 P(X = x_i|Y = y_j)$.

[4]Informally, the strength of a dynamic dependence chain quantifies the amount of information that propagates from the source to the target of the chain, equivalently, the correlation between them. A formal definition can be found in section 3 in [61].

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 *non-linear* 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 guess the value of X. Fano's inequality relates the probability of error in guessing the random variable X to its 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 $X \to Y \to \widehat{X}$, with $P_e = P_r(X \neq \widehat{X})$, we have

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

The inequality can be weakened to

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

or

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

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

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

2.5 Correlation-based Techniques

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

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 [53]:

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

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 Eta Coefficient

The correlation ratio or eta coefficient provides the same value as standard r for linearly related variables and a greater value for non-linearly related variables. The difference between eta and r presents a measure of the non-linearity of the relation between two variables. The eta coefficient is defined as [55],[18]:

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

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.

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 inequality [17],[22] to prove that we can potentially achieve a higher prediction rate using a correlating predictor rather than a local predictor.

Simulation Approach.

We take an implementation-independent approach in studying the usage of DIFA in data value prediction, and the usage and benefits of correlating value predictor. We remove any hardware constraint to obtain a basic and general understanding of the problem. Additional research is required to achieve a realistic implementation. We implement and simulate two types of predictors: stride and finite context method (FCM). We also introduce and simulate the linear predictor. We use unbounded table size, hence, we do not have to worry about aliasing. We update the table immediately after a prediction is made. In reality, it might take few cycles to predict a value and update the prediction tables.

We use DynFlow [58], [60] to determine information flow between variables and calculate their strength. We conduct our study using the Siemens benchmark suite [1] and SPECjvm2008 (Java Virtual Machine Benchmark) [2]. The Siemens benchmark suite contains seven programs which were originally created by researchers at Siemens Corporate Research. 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 JRE (Java Runtime Environment). 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 *Schedule* program from Siemens benchmark suite, and the subset of the SPECjvm2008 benchmarks listed and described in Table 1. We did not use some of the SPECjvm2008 benchmarks as we were unable to run them within the Dynflow tool. The *Schedule* program parses files that contains integer and float numbers and count the number of digits in each number. We study the predictability of all numeric variables in the programs we have used.

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 1: Subset of *SPECjvm2008* benchmarks studied in this work.

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, and (iii) a combination of the two previous cases.

Consider a variable for which we may be interested in predicting its value after each execution instance. Each successive execution may result in changing the machine state corresponding to the variable. 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. The conditioning reduces entropy theorem [17] states that $H(X|Y) \leq H(X)$, with equality if and only if X and Y are independent. 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 $X \rightarrow Y \rightarrow \widehat{X}$, with $P_e = P_r(X \neq \widehat{X})$, we have [22]:

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

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 the above 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 hit rate. 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, correlat-*

ing or hybrid, and 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: $P_a = 1 - P_e$, where P_e is the probability of error previously defined in section 2.4.3. We define the theoretical hit rate to be equal to $P_a * 100$. It represents the upper bound on the predictability hit rate.

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 consider these two cases below.

1. The relation is between consecutive values of the same variable: we introduce a new predictor that we call *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 non-stride pattern, the following equations can be used to calculate the values of a and b:

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

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

   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:

$$x_1, x_2, x_3 ... x_1, x_2, x_3 ...$$

$$ax_1 + b, ax_2 + b, ax_3 + b ... ax_1 + b, ax_2 + b, ax_3 + b ...$$

$$y_1, y_2, y_3 ... y_1, y_2, y_3 ...$$

   (c) **Theorem 3:** When the source variable does not follow a specific 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:

$$y_1 = ax_1 + b$$

$$y_2 = ax_2 + b$$

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

$$b = y_1 - ax_1 = y_2 - ax_2$$

We conclude that when two different variables are related with a linear equation, both source and target follow the same pattern. The linear predictor can be used to predict the value of the target knowing that of the source when the variables themselves do not follow neither a linear nor a FCM pattern.

4.2.2 Non Maximum r

Now we consider the case where $|r| < 1$. We can use the regression line to predict future value of the target, knowing the source 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 example, 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,...
}
```

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

In this section we consider the case where there exists a non-linear 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 non-linear relations.

Theorem 4: A non-linear relation does not imply stride pattern is preserved.

Proof: We prove the above theorem using the following counter-example. Consider the non-linear 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 non-linear 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 non-linear 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], 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.

If the non-linear relation can be defined, it can be used to compute the value of the target using that of the source or the new value of a variable using its previous one.

Now that we have analytically answered some of the questions posed earlier, we try to empirically answer the remaining ones and confirm our analytical results in the following section.

5. EMPIRICAL STUDIES USING DIFA

Most of existing value prediction work is done at the assembly level where data dependences are between instruction operands. At the Java level, dependences are between variables and objects. Pairs between which dependences exist at the assembly level are generally much more than those at the Java level. This makes studying dependences and value prediction at the assembly level easier and gives more confidence in the observed result.

We conduct our study in this paper at the Java level due to the availability of DynFlow that we use to determine dynamic flow between variables and measure their strengths. In future work, we are planing to conduct this work at the assembly level. We have employed stride, linear and FCM predictors to predict all numeric variables values. We observe that the source and target variables are of the same type except for very few cases where the target is a downcast of the source. The linear predictor defines the equation of the linear relation that exists between two different variables and predicts the target variable value knowing that of the source or between successive values of the same variable

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

Variable Name	Stride Hit Rate	FCM Hit Rate	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

Table 2: *Schedule* program integer variables measurements.

and predicts its next value knowing its current value. We do not implement last value predictor since it is a special case of stride and FCM predictors. We account for the learning time of the predictors.

5.1 Can we make some observations on the predictability of a variable given the maximum strength of all paths leading to it?

For each integer variable from the *Schedule* program, we provide in Table 2 the hit rates 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 schemes, and the strongest flow into the variable measured per scheme. We can tell from Table 2 that there is no direct relation between the maximum strength of flow into a variable and its predictability. Consider variables *job.val, numProcesses–, newProcess.proc.priority and readInt.i* from Table 2. 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 rates for all of these variables using both stride and FCM predictors.

To try to explain the hit rates obtained using the predictors, we closely examine all the flows into each variable and how the variable is updated in the program. We conclude that the hit rate of a target variable is affected by: i) hit rates 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.

5.2 Can we make some observations on the predictability of a target variable knowing the predictability of source variables that have strong flow into it?

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 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 equal 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.

5.2.1 *Linear relation from the variable to itself captured via standard r*

For type (a,i), Figure 1 shows the average hit rates obtained using stride, linear, and FCM predictors. It also shows the average theoretical hit rates. The 100% theoretically achievable hit rates are explained by the fact that the conditional entropy $H(X|Y)$ for variables of this type is 0, which implies that knowing the value of the variable at the current state, we can predict its value at the next state with no uncertainty. For all the benchmarks, at least one of the three predictors used perform 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.

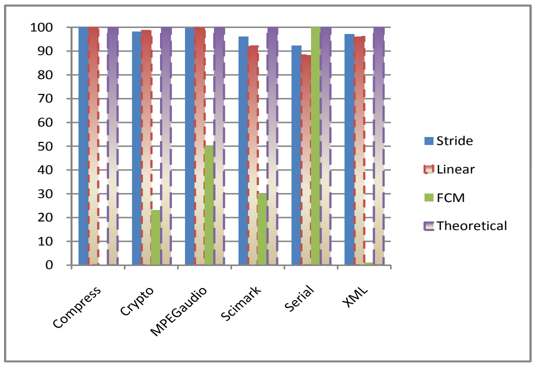

Figure 1: Average hit rates for variables selected via linear relation from the variable to itself captured via r = 1.

5.2.2 *Linear relation from a source variable into a target variable captured via standard r, where source and target are different*

For type (a,ii), it can be seen from Figure 2 that linear relation predictor, described in section 4.2.1, achieves 100% hit rate. The linear predictor uses the linear relation between the values of the source and target variables to predict the target value from that of the source. The predictability rates at source variables are preserved within acceptable deviation at the target variables. This validates our mathematical analysis in section 4.2.1. For *MPEGaudio* and *Scimark* benchmarks, the predictability rates at the source and target variables are exactly the same. For *Compress* benchmark, there is small difference in the predictability rates between source and target variables. For *MPEGaudio*, the hit rates using FCM predictor are equal to 0 for all source and target variables. For *Scimark*, the values of the hit rates are equal to 0 for all source and target variables using the three predictors. We do not find any pair of variables related with a linear relation captured via standard r for the remaining three benchmarks: *Crypto*, *Serial* and *XML*. The theoretical achievable prediction hit rates of the target knowing the source are equal to 100%. This is due to the fact that the conditional entropy $H(X|Y)$ for variables of this type is 0, which implies that knowing the value of the source variable, we can predict the value of the target variable with no uncertainty.

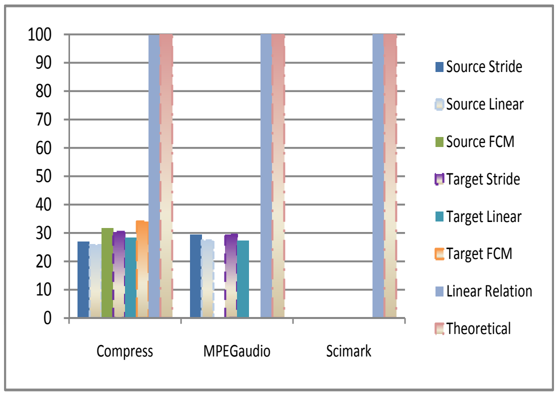

Figure 2: Average hit rates for variables selected via linear relation from source to target captured via r = 1.

Low average predictability rates 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 type (a,i) discussed in section 5.2.1. 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.

In cases where there are multiple paths with strong linear flow into a target variable from more than one source variable, we use the appropriate source, according to the path that is taken, to predict the target.

5.2.3 *Non-linear relation from the variable to itself*

For type (b,i), Figure 3 presents the predictability rates using stride, linear, and FCM predictors. It also shows the average theoretical hit rates. The 100% theoretically achievable hit rates are explained by the fact that the conditional entropy $H(X|Y)$ for variables of this type is 0. For *Compress* benchmark, FCM predictor achieves very high hit rates, close to 100%, and outperforms the other two predictors. Hit rates 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 perform slightly better than FCM 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 stride and linear predictors hit rates are equal to 0. We do not have candidate variables that meet type (b,i) selection criteria, in the remaining two benchmarks: *Serial* and *XML*.

5.2.4 *Non-linear relation from a source variable into a target variable, where source and target are different*

For type (b,ii) above, Figure 4 presents the predictability rates using stride, linear, and FCM predictors. It also shows the average theoretical hit rates. The theoretical achievable hit rate of a target knowing its source is equal to 100%. This is due to the fact that the conditional entropy $H(X|Y)$ for variables of this type is 0, where X corresponds to the target and Y to the source. It is clear from the results of

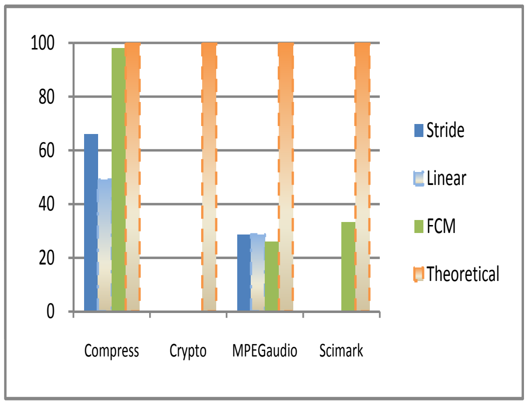

Figure 3: Average hit rates for variables selected via non-linear relation from the variable to itself captured via eta = 1.

benchmarks *Compress*, *Crypto* 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. The target might follow a FCM pattern though the source does not, such as in *Crypto*. This result confirms our analysis in section 4.3. We do not find any pair of variables related with a non-linear relation captured via eta coefficient in *Serial* and *XML* benchmarks.

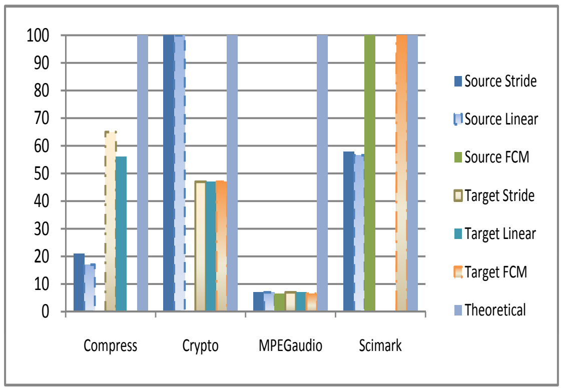

Figure 4: Average hit rates for variables selected via non-linear relation from source to target captured via eta = 1.

We only observe paths in *MPEGaudio* benchmark that satisfy type (c). 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 non-linear relation captured via normalized mutual information, we could have achieved up to 100% prediction accuracy.

5.3 Can we use the schemes that capture the strength of information flow to select appropriate value predictors?

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

When the strength of flow from the variable to itself measured using standard r is equal to 1, either a FCM or linear predictor can be used depending on the pattern of the values of the variable. When the variable follows a linear sequence pattern, a linear predictor can be used. When the variable follows a repeating linear sequence pattern, a FCM predictor achieves a higher hit rate than a linear predictor. As the length of the sequence increases, the hit rate of the linear predictor increases and it becomes more suitable to use it when a FCM predictor is not affordable. When the strength of flow from source variable into target variable measured using standard r is equal to 1, a linear predictor can be used to predict the target variable values from those of the source. The linear predictor reduces to a stride predictor when the value of a in the linear relation is 1.

5.4 Which of the schemes used to measure dynamic information flow strength is more relevant to our work?

DynFlow makes use of normalized mutual information, eta coefficient and standard r to measure the strength of dynamic information flow between variables. We utilize the three schemes in our analytical study. We use conditional entropy in defining an upper bound on prediction accuracy, and the three schemes in mathematically proving which patterns of the source variable values are preserved by the target variable values. In our empirical study, we have used standard r more than the other two schemes, since a linear relation is much easier to define and requires fewer computations than a non-linear relation captured via eta coefficient and normalized mutual information. We have introduced linear predictor that predicts the next value of a variable whose values follow a linear pattern from its current value, and predicts a target variable values from those of a source variable that have maximum strength flow into the target captured via standard r. We use conditional entropy in our experimental study to confirm the upper bounds on prediction accuracy derived in section 4.1. We note that a FCM pattern is detected using both normalized mutual information and eta coefficient, though, normalized mutual information is more reliable when the FCM pattern changes.

6. APPLICATIONS AND IMPLEMENTATION

We have conducted in this paper theoretical and empirical studies about the potential usage of DIFA in data value prediction. We have concluded that:

- We can use DIFA to identify highly predictable variables.
- We can recommend for some variables the type of predictor to use.
- We can identify using DIFA pairs of variables related with a linear relation and use the linear predictor to predict the values of one of them knowing those of the other when both of them do not follow a predictable pattern.

Using our results, we can be very conservative and select only variables into which there exist linear flow captured via standard r, where $|r| = 1$, since such variables are highly predictable. We can further select variables identified via non-linear relations from the variable to itself, though these variables may or may not be predictable using FCM predictor. This selection criteria can significantly reduce the impact of value misprediction.

Our proposed techniques will not require significant changes to existing value prediction techniques. The linear predictor requires the calculation of the values of its parameters, which requires very little overhead in the current era of high integration multi-core processors.

Variable identification can be done in software and propagated to the hardware using special compiler added hints or ISA extensions. It could also be done in hardware. For example, architectures that support DIFA for security applications [40],[24] could make use of our proposed techniques to enhance performance and value prediction.

7. FUTURE WORK

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

1. Can we determine using DIFA which function arguments affect its returned value? Can we predict the returned value by observing these arguments? Is there correlation between some of the variables manipulated within the function and its returned value?

2. Will the observations and results we have obtained from Java programs hold for other programming languages? Will they hold for other applications, such as SPEC CPU2006?

   A compiler translates a program from a high-level language to a low-level language that the computer can run. It produces machine or assembly language programs called object programs. The Java compiler does not translate a Java program into assembly language; instead it translates a Java program into bytecode. *Dynflow* calculates flow strength at the bytecode level and maps it to the Java level. We are interested in checking if the obtained results hold at the assembly level. We are going to develop a dynamic information flow analysis tool that works on x86 binaries and verify if the results we obtain at the Java level hold at the assembly level using SPEC CPU.

3. We are interested in exploring the possibilities of using DIFA in branch prediction and memory disambiguation [21]. We are going to look into identifying the set of branches on which a given branch strongly depends. 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 [49] and dynamic per-branch history length adjustment [30] were previously addressed using hardware.

4. 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 would need to be implemented to exploit such increase in effective ILP from zero strength paths?

5. One intriguing research direction is to use dynamic dependence analysis for speculative parallelization. We will work on developing hardware mechanisms that leverage strength-based dynamic dependence analysis to perform accurate value prediction of task interdependent data values in speculative multithreaded architectures. We will also study the potential of exploiting zero-strength flows in the parallelization and task selection process.

6. We will evaluate architectures that support DIFA automatically in hardware and make use of these architectures for automatically identifying highly predictable variables. Such hardware-only value prediction technique with DIFA could be useful to improve performance of existing applications without needing to recompile.

8. 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 predictability hit rate 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 introduced and simulated the linear value predictor. We proved and experimentally showed that, using a 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 gave an explanation about when a given value predictor can perform well and even outperform other predictors. We provided a selection criteria that can be used to identify highly predictable variables.

We believe that the usage of dynamic information flow analysis in microarchitecture, mainly in data value prediction and branch prediction, has significant potential for performance improvement. We are going to further explore this in our ongoing research.

9. ACKNOWLEDGMENTS

This work has been supproted by grants from Intel Corporation and the American University of Beirut Research Board.

10. REFERENCES

[1] M. J. Harrold and G. Rothermel. Siemens Programs, HR Variants. http://pleuma.cc.gatech.edu/aristotle/Tools/subjects/.

[2] The Standard Performance Evaluation Corporation (SPEC). The SPEC Benchmark Suite. http://www.spec.org.

[3] A. Gandhi, H. Akkary and S. Srinivasan. Reducing branch misprediction penalty via selective branch recovery. *In Proceedings of the 10th International Symposium on High Performance Computer Architecture*, February 2004.

[4] A. I. Moshovos. Memory Dependence Prediction. Technical report, Ph.D. Thesis, Computer Sciences Department, University of Wisconsin, Madison, December 1998.

[5] A. Moshovos, S. E. Breach, T. N. Vijaykumar and G.S. Sohi. Dynamic speculation and synchronization of memory dependences. *In the Proceedings of the 24th Annual ACM/IEEE Conference on Computer Architecture*, June 1997.

[6] A. S. Al-Zawawi, V. K. Reddy, E. Rotenberg and H. Akkary. Transparent Control Independence (TCI). *ISCA-34*, June 2007.

[7] A. Smith, R. Nagarajan, K. Sankaralingam, R. McDonald,D. Burger, S. W. Keckler and K. S. McKinley. Dataflow predication. *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 89–102, 2006.

[8] A. Sodani and G. Sohi. Dynamic instruction reuse. *Proceedings of the 24th International Symposium on Computer Architecture (ISCA)*, June 1997.

[9] A. Sodani and G. Sohi. Understanding the differences between value prediction and instruction reuse. *MICRO-31*, pages 205–215, 1998.

[10] H. Akkary and M. A. Driscoll. A dynamic multithreading processor. *MICRO-31*, November 1998.

[11] A.s Moshovos and G. S. Sohi . Reducing memory latency via read-after-read memory dependence prediction. *IEEE Transactions on Computers*, 51(3):313–326, March 2002.

[12] B. Goeman, H. Vandierendonck and K. De Bosschere. Differential fcm: Increasing value prediction accuracy by improving table usage efficiency. *Proceedings of the Seventh International Symposium on High-Performance Computer Architecture (HPCA'01).*

[13] B. Rychlik, J. Faistl, B. Krug and J. P. Shen. Efficacy and performance impact of value prediction. *1998 International Conference on Parallel Architectures and Compiler Technology*, October 1998.

[14] C. J. F. Pickett and C. Verbrugge. Compiler Analyses for Improved Return Value Prediction. Technical report, Technical Report SABLETR-2004-6, Sable Research Group, School of Computer Science, McGill University, 2004.

[15] C. J. F. Pickett and C. Verbrugge. Return value prediction in a java virtual machine. *In PW2: Proceedings of the 2nd Value-Prediction and Value-Based Optimization Workshop*, pages 40–47, October 2004.

[16] G. Z. Chrysos and J. S. Emer. Memory dependence prediction using store sets. *Proceedings of the 25th Annual International Symposium on Computer Architecture*, June 1998.

[17] T. M. Cover and J. A. Thomas. *Elements of Information Theory.* Wiley, New York, 1991.

[18] D. J. Krus. Visual statistics: Chapter 14, regression on multiple categories, visual-statistics.net. 2006.

[19] D. Tullsen and J. Seng. Storageless value prediction using prior register values. *26th International Symposium on Computer Architecture*, May 1999.

[20] D. E. Denning. *Cryptography and Data Security.* Addison-Wesley, 1982.

[21] J. R. Ellis. *Bulldog: A Compiler for VLIW Architectures.* The MIT Press, 1986.

[22] R. M. Fano. *Transmission of Information: A Statistical Theory of Communications.* The MIT Press, and John Wiley and Sons, New York, 1961.

[23] M. Franklin. *The Multiscalar Architecture.* PhD thesis, University of Wisconsin, November 1993.

[24] G. Venkataramani, I. Doudalis, Y. Solihin and M. Prvulovic. Flexitaint: A programmable accelerator for dynamic taint propagation. *IEEE 14th International Symposium High Performance Computer Architecture*, pages 173–184, February 2008.

[25] H. Akkary, K. Jothi, R. Retnamma, S. Nekkalapu, D. Hall and S. Shahidzadeh. On the potential of latency tolerant execution in speculative multithreading. *IFMT*, November 2008.

[26] H. Akkary, S. T. Srinivasan and K. Lai. Recycling waste: exploiting wrong-path execution to improve branch prediction. In *ICS-17*, pages 12–21, 2003.

[27] J. E. Smith and G. S. Sohi. The microarchitecture of superscalar processors. *Proceedings of the IEEE*, 83(12):1609–1624, August 1995.

[28] J. Oplinger, D. Heine and M.S. Lam. In search of speculative thread-level parallelism. *Proc. Eighth Int'l Conf. Parallel Architectures Compilation Techniques*, 1999.

[29] J. Singer and G. Brown. Return value prediction meets information theory. *In QAPL'06: Proceedings of the 4th International Workshop on Quantitative Aspects of Programming Languages*, 164:137–151, October 2006.

[30] J. W. Kwak and C. S. Jhon. Dynamic per-branch history length adjustment to improve branch prediction accuracy. *Microprocess. Microsyst.*, 31(1):63–76, 2007.

[31] J.D. Owens, W.J. Dally, R. Ho, D.N. Jayasimha, S.W. Keckler and L. Peh. Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108, September-October 2007.

[32] J.G. Steffan, C.B. Colohan, A. Zhai and T.C. Mowry. Improving value communication for thread-level speculation. *Proc. Eighth Int'l Symp. High-Performance Computer Architecture*, 1999.

[33] K. Asanovi, R. Bodik, B. C. Catanzaro, J.J. Gebis, P. Husbands, K. Keutzer, D.A. Patterson, W.L. Plishker, J. Shalf, S.W. Williams and K.A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical report, EECS UC Berkeley, Technical Report No. UCB/EECS-2006-183, 2006.

[34] K. Wang and M. Franklin. Highly accurate data value prediction using hybrid predictors. *Proceedings of the*

30th annual ACM/IEEE international symposium on Microarchitecture, pages 281–290, 1997.

[35] M. S. Lam and R. P. Wilson. Limits of control flow on parallelism. *Proceedings of the 19th Annual International Symposium on Computer Architecture*, pages 46–57, May 1992.

[36] M. Bishop. Computer security: Art and science. 2, December 2002.

[37] M. Burtscher and B. G. Zorn. Exploring last n value prediction. *International Conference on Parallel Architectures and Compilation Techniques*, pages 66–76, October 1999.

[38] M. Butler, T. Yeh, Y. Patt, M. Alsup, H. Scales and M. Shebanow. Single instruction stream parallelism is greater than two. *Proceedings of the 18th Annual International Symposium on Computer Architecture*, pages 276–286, 1991.

[39] M. H. Lipasti and J. P. Shen. Exceeding the dataflow limit via value prediction. *In Proceedings of the 29th Annual ACM/IEEE International Symposium and Workshop on Microarchitecture*, pages 226–237, December 1996.

[40] M.Dalton, H.Kannan and C.Kozyrakis. Raksha: A Flexible Information Flow Architecture for Software Security. *Proceedings of the 34th International Symposium on Computer Architecture*, 2007.

[41] M.H. Lipasti, C.B. Wilkerson and J.P. Shen. Value locality and load value prediction. *Proc. of the 7th. Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 138–147, October 1996.

[42] N. Tuck and D. M. Tullsen. Multithreaded value prediction. *Proceedings of the 11th International Symposium on High-Performance Computer Architecture*, 2005.

[43] N.J. Warter, D.M. Lavery and W.W. Hwu. The benefit of predicated execution for software pipelining. *Proceeding of the Twenty-Sixth Hawaii International Conference on System Sciences*, 1:497–506, January 1993.

[44] P. Marcuello, A. Gonzalez and J. Tubella. Thread partitioning and value prediction for exploiting speculative thread-level parallelism. *IEEE Trans. Computers*, 53(2):114–125, February 2004.

[45] P. Marcuello, J. Tubella, and A. Gonzelez. Value prediction for speculative multithreaded architectures. *Proceedings of the 32nd International Symposium on Microarchitecture*, November 1999.

[46] P. S. Ahuja, K. Skadron, M. Martonosi and D. W. Clark. Multi-path execution: Opportunities and limits. *In Proceedings of the 12th International Conference on Supercomputing*, pages 101–108, July 1998.

[47] R. J. Eickemeyer and S. Vassiliadis. A load instruction unit for pipelined processors. *IBM Journal of Research and Development*, 37(4):547–564, July 1993.

[48] R. K. Montoye, E. Hokenek and S. L. Runyon. Design of the ibm risc system/6000 floating-point execution unit. *IBM Journal of Research and Development*, 34(1):59–70, January 1990.

[49] R. Thomas, M. Franklin, C. Wilkerson and J. Stark. Improving branch prediction by dynamic dataflow-based identification of correlated branches from a large global history. In *ISCA '03: Proceedings of the 30th annual international symposium on Computer architecture*, pages 314–323, 2003.

[50] R.P. Martin, A.M. Vahdat, D.E. Culler and T.E. Anderson. Effects of communication latency, overhead, and bandwidth in a cluster architecture. *SIGARCH Computer Architecture News*, 25(2):85–97, 1997.

[51] S. Borkar. Thousand Core Chips-A Technology Perspective. *44th ACM/IEEE Design Automation Conference*, 4(8):746–749, June 2007.

[52] S. Hu, R. Bhargava and L. K. John. The role of return value prediction in exploiting speculative method-level parallelism. *Journal of Instruction-Level Parallelism*, 5:1–21, November 2003.

[53] S. Kachigan. Statistical analysis: An interdisciplinary introduction to univariate and multivariate methods. 1986.

[54] Y. Sazeides and J. Smith. Implementations of context-based value predictors. Technical report, ECE-TR-97-8, University of Wisconsin, Madison, 1997.

[55] S. Siegel. *Nonparametric Statistics For The Behavioral Sciences.* NY: McGraw-Hill, 1956.

[56] T. M. Austin and G. S. Sohi. Dynamic dependency analysis of ordinary programs. *Proceedings of the 19th annual international symposium on Computer architecture (ISCA '92)*, 20(2):342–351, May 1992.

[57] T. Nakra, R. Gupta and M. L. Soffa. Global context-based value prediction. *Proceedings of the 5th International Symposium on High Performance Computer Architecture*, pages 4–12, January 1999.

[58] W. Masri, A. Podgurski and D. Leon. Detecting and debugging insecure information flows. *15th. IEEE International Symposium on Software Reliability Engineering, ISSRE*, November 2004.

[59] W. Masri and A. Podgurski. An empirical study of the strength of information flows in programs. In *WODA '06: Proceedings of the 2006 international workshop on Dynamic systems analysis*, pages 73–80, May 2006.

[60] W. Masri and A. Podgurski. Algorithms and tool support for dynamic information flow analysis. *Inf. Softw. Technol.*, 51(2):385–404, February 2009.

[61] W. Masri and A. Podgurski. Measuring the strength of information flows in programs. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 19(2), October 2009.

[62] W. Mohan and M. Franklin. Improving data value prediction accuracy using path correlation. *Proceedings of the fifth international conference on Architectural support for programming languages and operating systems*, pages 76–84, 1992.

[63] W. W. Hwu. Compiling for ilp processors. *Proceedings of the IEEE*, 83(12), December 1995.

[64] Y. Sazeides and J. E. Smith. The predictability of data values. *Proceedings of the 30th International Symposium on Microarchitecture*, December 1997.

[65] Y. Sazeides, S. Vassiliadis and J.E. Smith. The performance potential of data dependence speculation and collapsing. *MICRO-29*, December 1996.