# Return Value Prediction meets Information Theory

Jeremy Singer[1] Gavin Brown[2]

*School of Computer Science*
*University of Manchester*
*Manchester, UK*

**Abstract**

Accurate return value prediction is a key tool for enabling effective speculative method-level parallelism, which will be a standard feature in the next generation of chip-multiprocessor architectures. In this paper we give some information theoretic measures that indicate intrinsic predictability of method return values. This is in stark contrast to the current ad-hoc heuristic measures imposed by specific prediction techniques. Our hope is that the application of information theoretic principles to the field of return value prediction should result in new kinds of predictors, and better deployment of existing prediction techniques. The two main contributions of this work are: (i) to show that there is some correlation between information theoretic measures and return value predictor performance; (ii) to highlight some major issues that need to be resolved before information theory can be adopted practically by the return value prediction community.

*Key words:* return value prediction, information theory, speculative method-level parallelism, entropy.

## 1 Return Value Prediction

Value prediction involves the estimation of the next value in a sequence, given knowledge of the sequence so far. This is a classic time series problem, for which standard prediction techniques are well-known. However this paper focuses on value sequences generated by microprocessors executing code in real time. If the techniques we describe are to be implemented directly in processor architecture, then they must be fast (to operate in real time) and cheap (to be implemented in reasonable transistor budgets). These constraints restrict our consideration to the simplest value prediction techniques. In fact several

[1] Email: jsinger@cs.man.ac.uk
[2] Email: gbrown@cs.man.ac.uk

different value prediction techniques have been proposed, that are amenable to hardware implementation. These techniques have been justified by various empirical studies to determine that value sequences computed by processors are predictable. Unfortunately the only measure of *predictability* is the prediction accuracy of a particular value prediction technique! This paper addresses the issue of predictability at a more fundamental level. We are interested in whether value sequences are *intrinsically predictable*, independent of which techniques may be employed to take advantage of this predictability. The underlying, fundamental predictability may be measured by the application of information theoretic metrics such as entropy and redundancy. To the best of our knowledge, there is no other predictability study that takes this approach.

This paper focuses on a specific kind of value sequence generated at runtime. Assuming an object-oriented model of execution, we are interested in the sequence of values generated by each method. A method $m$ will have a return value sequence of length $n$ if $m$ is called $n$ times during program execution.

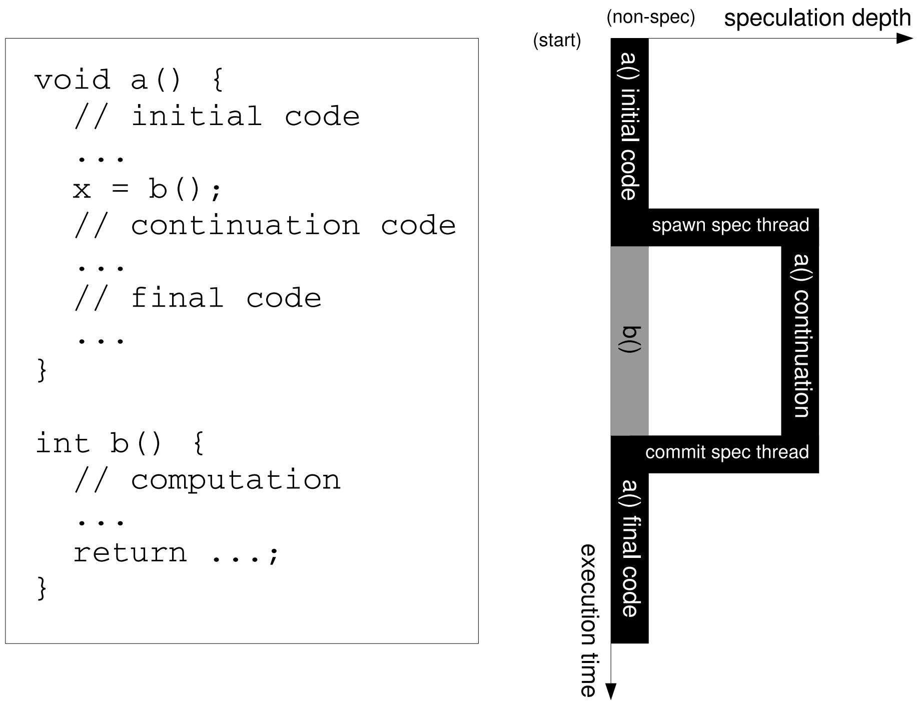
## *1.1 Motivation*

Return value prediction is a key part of *speculative method-level parallelism* (SMLP), as described by Chen and Olukotun [4]. Given the ability to predict method return values accurately, it is possible to speculate on the outcome of method calls. At a call point, the caller usually waits for the callee to complete and return a value, whereas in the SMLP paradigm the caller predicts the callee's value, and executes speculatively as if the callee had already completed. Of course, the callee has to execute too, to validate the prediction. If the speculation succeeds, then effectively both the callee and part of the caller execute in parallel. Figure 1 shows a Java program fragment and how it may be executed via SMLP.

This is an effective way of extracting parallelism from sequential object-oriented programs. Warg and Stenstrom report a speedup of 3.5 over sequential execution on an ideal machine with perfect memory and return value prediction [24]. However when speculation fails, expensive mechanisms are required to rollback and re-execute in a non-speculative manner. These details are beyond the scope of this paper. However their expense dictates that if SMLP is to give any performance advantage, then speculations have to succeed fairly often. The crucial point is that successful speculation is only possible with accurate return value predictions.

## *1.2 Techniques*

Value prediction techniques were originally developed in order to increase instruction-level parallelism. Originally, values were predicted for `LOAD` instructions that retrieve data from memory [17]. This idea was soon extended to predict values for all instructions that update an entry in the processor



```
void a() {
  // initial code
  ...
  x = b();
  // continuation code
  ...
  // final code
  ...
}

int b() {
  // computation
  ...
  return ...;
}
```

Fig. 1. Speculative method-level parallelism in action

register file [16]. These techniques took advantage of the newly discovered phenomenom of *value locality*, which refers to the high likelihood of a previously seen value being computed repeatedly by a particular instruction.

With current interest in thread-level parallelism, value prediction is now applied to sequences of method return values. This is the enabling technology for speculative method-level parallelism (SMLP). The two standard limits studies for SMLP [18,24] both consider value prediction to be helpful, if not essential, for effective SMLP.

The simplest prediction technique is *last value prediction* (LVP) [17]. In this scheme, it is assumed that the next value computed will be the same as the previous value. This naive approach is surprisingly effective. It works because sequences of repeated return values such as $(1, 1, 1, \ldots)$ are fairly common.

An extension of the LVP technique is *stride value prediction* (SVP) [10]. In this scheme, it is assumed that the next value $v_n$ can be computed from the previous two values $v_{n-1}$ and $v_{n-2}$.

$$v_n = v_{n-1} + stride$$

where

$$stride = v_{n-1} - v_{n-2}$$

SVP works for arithmetic progression sequences such as $(1, 2, 3, \ldots)$. Note that when *stride* is 0, SVP behaves as LVP.

Sazeides and Smith [22,21] refer to LVP and SVP as *computational* predictors. They introduce a new class of predictors, known as *context-based* predictors. The finite context method (FCM) predictor uses the history of recent values to predict the next value. This scheme relies on repeated patterns of values occurring in a return value sequence. This is justified by the common occurrence of control flow loops in programs. An FCM predictor is generally implemented as a lookup table, indexed by context. The context will be composed of the most recently seen values. The table entry will contain the values that have followed this context on previous occasions. The value that has the highest frequency is predicted, since it is the most likely value to come next, based on the sequence so far. The *order* of an FCM predictor denotes the size of the context associated with each entry in the lookup table. So an $n$th-order FCM predictor uses $n$ consecutive return values for its context. The *capacity* of an FCM predictor denotes how many entries may be stored in the lookup table.

Hybrid predictor models [23] incorporate two or more different prediction schemes. They are able to achieve greater accuracy in general, as is to be expected from the principles of ensemble learning [3,25].

This paper focuses on LVP and first-order FCM predictors.

### *1.3 Predictability*

Until now, the performance of value predictors has been used to measure the predictability of value sequences. For instance, Sazeides and Smith commence their predictability study [22]: “The predictability of a sequence of values is a function of both the sequence and the predictor used to predict the sequence.” Our view is that predictability should be *independent* of any particular predictor. Gabbay and Mendelson [11] define value predictability as “the potential that resides in a program to successfully predict the outcome values generated during its execution.” While we agree with this statement, we do not agree with the subsequent statement that predictability depends “on the capabilities of the value predictor.”

Essentially, we argue that *predictability is an inherent property of a value sequence*, entirely independent of value prediction schemes. If a sequence is predictable, then it should be predictable by some predictor, but we make no assertions about which particular predictor should be used. Information theory provides us with fundamental measures of predictability, that can be applied to value sequences.

One important measurement for value predictors is *confidence*. This estimates how likely value predictions are to be accurate. Inaccurate predictions are likely to cause incorrect speculations. Recall that it is expensive to recover from the effects of incorrect speculations, so in such cases it is better not to speculate at all.

Generally confidence is based on predictor performance (so all methods are continuously monitored by the return value predictor, regardless of whether or not they are currently candidates for thread-level speculation). We argue that confidence should be based on a more intrinsic measure of predictability, derived from information theoretic principles.

### *1.4 Contributions*

This paper makes four key contributions.

(i) It applies principles of information theory (Section 2) to value prediction.

(ii) It identifies information theoretic metrics of *first-order redundancy* and *normalized mutual information* as possible measures of return value sequence predictability (Section 2).

(iii) It presents a study of Java method return value predictability (Section 3.3). This is superior to similar studies [24,14] that rely on specific value prediction techniques.

(iv) It highlights current challenges (Section 3) and potential future research directions in this new area (Section 4). It notes implications for upcoming virtual machines and chip-multiprocessor architectures.

## 2 Information Theory

Information theory provides a rich mathematical framework for analysis of data sources. Originally developed by IBM in the context of secure communications and cryptography, it has been applied in fields as diverse as machine learning, medical image processing, and financial market prediction. We explore how these ideas might be applied to return value prediction for thread-level speculation. Note that there is little existing work that applies information theoretic entropy to value streams. An early study examines the entropy of hardware-generated streams of memory addresses across the address bus [13]. Recent work by Clark et al [5] measures entropy of variable values in a simple imperative language. They derive rules to model how different program statements transform entropies. Their research area is static analysis for information flow security.

### *2.1 Entropy, Redundancy, Mutual Information*

The fundamental measure in this framework is *entropy*, which explicitly quantifies the information content in a given source of data: the more ‘randomness’ or unpredictability in the data source, the higher the entropy value. As an example consider a device producing symbols according to a random variable $X$, defined over a finite alphabet of possible symbols $S$. If we assume each successive symbol $s_i \in S$ is independent of the previous ones, the *unconditional* *entropy* is defined as,

$$H(X) = -\sum_{i=1}^{|S|} p(i)\log(p(i)) \tag{1}$$

where $p(i)$ is the probability of the $i$th symbol being produced. Note that all logarithms are taken to base 2. In practical terms, $p(i)$ can be calculated with frequency counts, i.e.:

$$p(i) = \frac{\text{number of occurrences of symbol } s_i}{\text{total number of symbols seen}} \tag{2}$$

In this work, we consider the device as a method within a computer program, returning values when it is called[3]. Each successive call may change the machine state, so we consider the random variable defining the method behaviour to have changed also: the variable now is $X$, while on the next call it is $Y$, and we wish to compute the uncertainty in $Y$ given that we know $X$. In this case it is clear that successive method return values are *not* independent of one another, and what we have in fact could be considered as a *time series* of values. An unconditional entropy measurement will not take this into account, therefore we use *conditional entropy.*

$$H(Y|X) = -\sum_{i=1}^{|S|} p(i) \sum_{j=1}^{|S|} p(j|i)\log(p(j|i)) \tag{3}$$

This is the *First Order Conditional Entropy.* The required probabilities can again be computed from frequency counts:

$$p(j|i) = \frac{\text{number of times } s_j \text{ follows } s_i}{\text{number of occurrences of } s_i} \tag{4}$$

First Order Conditional Entropy (FOCE) has a minimum value of zero and a maximum value of $\log(|S|)$. It measures the uncertainty we have in the next return value, given that we know the current value. If values are produced uniformly at random over the set of possible symbols $|S|$, then eq.(3) will converge in the limit to $\log(|S|)$.

An important issue arises here when applying this to return value prediction. Imagine the case when a method only ever produces values from the set $\{-1, 0, +1\}$, for example returning the status of a file handle. Now imagine a method producing values from the set $\{0, 1, 2, 3\}$, for example calculating $modulo(n, 4)$. The random variables defining these methods have different numbers of outcomes—even if they are *entirely* random, the maximum FOCE value of the first method is $\log(3)$, and the second $\log(4)$. Does this mean the first one is more 'predictable'? A fix of sorts can be obtained by normalizing the FOCE, this is the *redundancy*,

$$R(Y|X) = 1 - \frac{H(Y|X)}{\log(|S|)} \tag{5}$$

[3] At present we do not consider possible arguments to these methods. This is a context-insensitive analysis [20].

Redundancy 0 means the variable is entirely random, and 1 means entirely deterministic. The conditional entropy may however not tell us the full story. A small value for $H(Y|X)$ (i.e. a small uncertainty in $Y$ given $X$) could be because $X$ tells us a lot about $Y$, or because $H(Y)$ is small to begin with. This problem of course remains with the redundancy since it is just a normalised version of the conditional entropy.

The *Mutual Information* between $X$ and $Y$ removes this problem. The Mutual Information is,

$$I(X;Y) = H(Y) - H(Y|X) \tag{6}$$

This is easily computed from the entropy measurements we have already described above. This measurement is symmetric, i.e. $I(X;Y) = I(Y;X)$, and tells us the reduction in our uncertainty of $Y$, when the value of $X$ is revealed to us. Unlike Pearson's R correlation coefficient, which only detects *linear* correlations between random variables, the mutual information can detect arbitrary *nonlinear* relationships between $X$ and $Y$.

$I(X;Y)$ can be normalized to a value between 0 and 1 by dividing by $H(Y)$. All mutual information values in Section 3 are normalized in this way. A high value of mutual information indicates that there is information in the symbol sequence to be discovered. A low value of mutual information indicates that there is no information, and therefore little opportunity for prediction. These are 'textbook' definitions of mutual information. When we began to investigate the possibilities for it (and indeed all of these measurements) in return value prediction, we encountered several non-trivial issues that we will now describe.

## 3 Issues to be Resolved

We have explored the possibilities of applying information theory to return value prediction. We note the following issues which we believe are critical for progress in this area.

### *3.1 For methods with a large range of return values, i.e. $|S|$ is large, how can we quickly estimate the entropy when we have such a sparse sample from the probability distribution?*

Consider 1000 calls of a method that may return $2^{32}$ possible values! We have an extremely unreliable estimate of the probabilities, and therefore also of the entropy measurements—how do we deal with this? When can we trust our measurements? This issue is particularly pertinent when we attempt to do online sampling of return values for dynamic speculative optimizations.

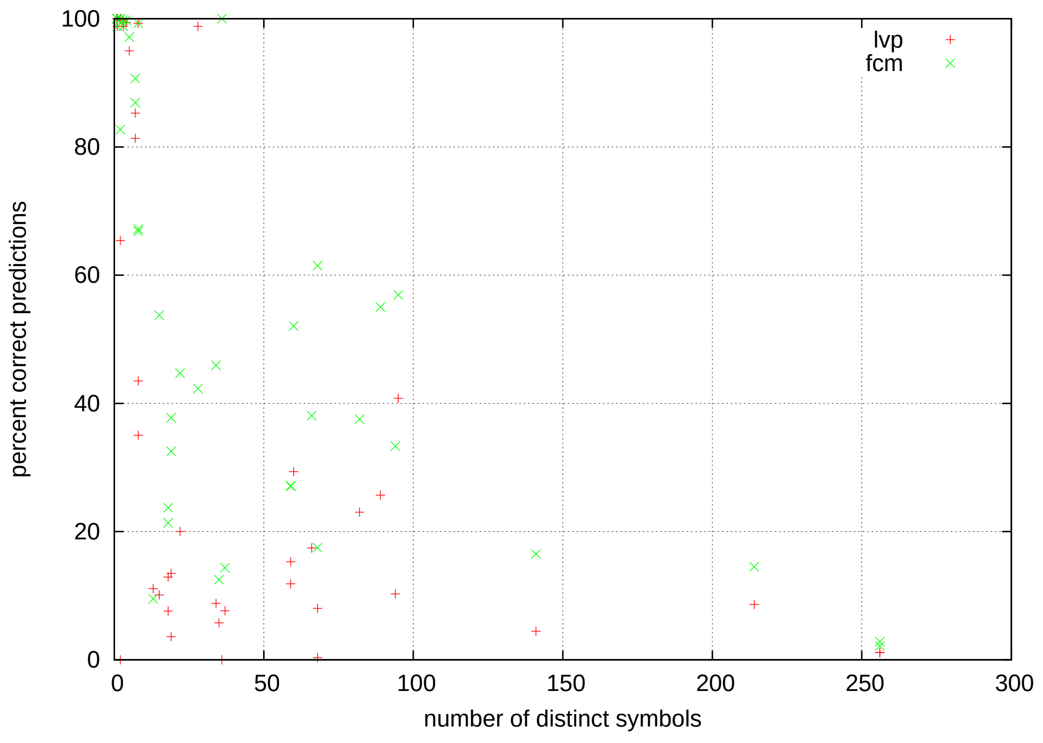


Fig. 2. Alphabet size versus predictability

## *3.2 How can we compare entropy values between methods that have different numbers of possible return values?*

In general a larger alphabet makes a return value sequence harder to predict, as Figure 2 clearly shows. This graph plots the relationship between alphabet size and predictor accuracy for two common predictors. There are two points for each method, one for its LVP accuracy and the other for its FCM accuracy. The data is collected from all methods in the SPEC JVM98 benchmark suite that return 32-bit integer values and are called at least 100 times. Appendix A gives full experimental details.

The issue here is that when we calculate mutual information on one method, we may have seen an alphabet of three characters, but how can we compare this mutual information value to another value computed over 1000 characters? It is difficult to compare information channels of different arities.

One possible approach is to reduce the total number of states that must be computed. (The number of states should increase with the alphabet size.) This reduction can be achieved by *state lumping* [8], which is very similar to the idea of *abstract interpretation* [6] from static analysis of programs. Basically, this approach reduces the state space by grouping together sets of related states into compound abstract states. Such techniques can reduce the size of value prediction tables for context-based predictors in a more principled way than existing hash-based algorithms [21].

A Java type expresses an upper bound on the number of symbols that may be returned by a method. However it is often the case that primitive types are not expressive enough to specify the precise range of symbols returned by a method. All our calculations assume that each method's alphabet size is equal to the number of distinct symbols observed as return values from that method. Methods with large alphabets are ideal candidates for state lumping. One pragmatic reason is that it makes the analysis cheaper. One theoretical reason is that it makes comparison with other methods fairer. We attempted to create a variant of FCM that groups together low frequency symbols, identified by frequency analysis over a training set of return values for each method. We grouped together infrequent symbols into an 'unknown' state. When the FCM predictor predicted the next symbol to be 'unknown' we refused to make any prediction. The technique did not work well. We found that we refused to make predictions far too often. Also our entropy and redundancy calculations were inaccurate since we treated 'unknown' as an ordinary symbol. So a sequence of the form ('unknown', 'unknown', . . . , 'unknown') has redundancy 1, which should indicate predictability. Really, we would need to treat the 'unknown' state similar to the 'Not-a-Number' concept from floating-point arithmetic and somehow work this into our information theory equations. It would be interesting to see how mutual information varies with state lumping, as the number of allowed states varies.

## *3.3 Measures like redundancy and mutual information tell us when there is information to discover, but they give no help on* how *to discover it!*

Figure 3 shows the correlation between redundancy and predictor accuracy. There are two points for each method, one for its LVP accuracy and the other for its FCM accuracy. The data is collected from the same set of benchmark methods as in Section 3.2.

Figure 4 is a similar plot. It shows the correlation between normalized mutual information and predictor accuracy. There seems to be less correlation for normalized mutual information than for redundancy. However we discovered that the anomalous states can be removed by filtering out methods that have an alphabet size of fewer than 15 symbols. There is obviously a complex relationship between entropy and alphabet size and predictor accuracy, which we are still investigating.

One possible clue to the relationship between entropy and accuracy is the *Fano inequality*, a fundamental bound in information theory [7]. Given a data source with FOCE $H(Y|X)$ and alphabet size $|Y|$, then the minimum misprediction rate $p$ is bounded as:

$$p \geq \frac{H(Y|X) - h(p)}{\log(|Y|) - 1} \tag{7}$$

where $h(p)$ is the binary entropy function:

$$h(p) = -(p \log p + (1-p)\log(1-p)) \tag{8}$$

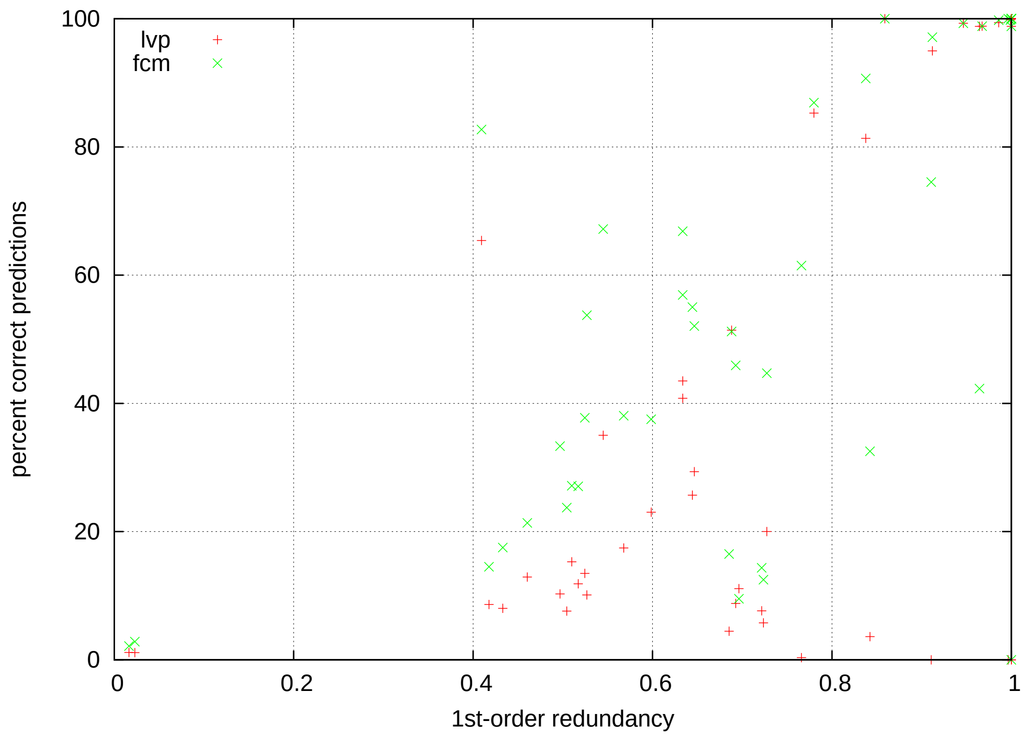


Fig. 3. Redundancy versus predictability

Figure 5 shows the Fano inequality plotted for boolean method return values (where $|Y| = 2$). The points are boolean method return value traces from SPEC JVM98 benchmarks. Each method is called at least 100 times, and its values are predicted using the simple LVP technique. The maximum accuracy bound is $(1 - p)$ where $p$ is the minimum misprediction rate. Clearly all the points satisfy the bound. More theoretical development is needed before we can apply the Fano inequality to larger alphabet sizes (particularly integer return values) and more complex predictors (which make use of other history features in addition to the last value).

It is clear to see that a low value of redundancy (or mutual information) generally indicates a low predictor accuracy for *all* predictors. This is an extremely reliable indication of unpredictability. Conversely, a high value of redundancy (or mutual information) indicates that *some* predictor may have high accuracy, but we have no idea which predictor is correct (FCM, LVP, SVP, other, or hybrid of these). The information theoretic metrics give no clues as to which prediction technique will work best. Thus, redundancy (or mutual information) gives an intrinsic predictability measure but this is not necessarily a tight upper bound on the accuracy of a particular prediction technique!

Figure 6 confirms our intuition. This graph shows that methods that return purely random sequences of values have a redundancy (or mutual information) of 0, indicating complete unpredictability. One method has an alphabet of

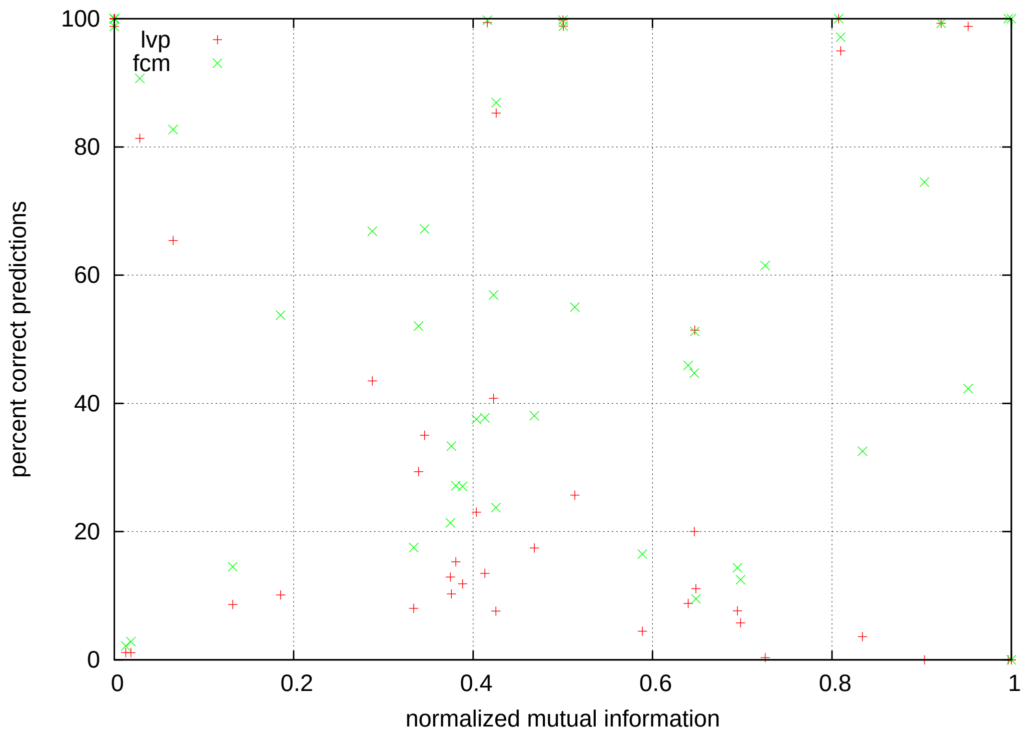


Fig. 4. Mutual information versus predictability

$\{0, 1\}$ and the other has an alphabet of $\{0, 1, \ldots, 99\}$. Note that although both methods are equally unpredictable in terms of information theory, actual predictors are more likely to guess the right answer with a smaller alphabet size than a larger one!

These random methods are contrasted with a real method from the SPEC JVM98 benchmark set, that shows some predictability and some accuracy. Also for reference, we study a stride sequence $(1, 2, \ldots, 99999)$. According to the information theoretic metrics this sequence should be predictable, but neither of our two featured predictors (FCM and LVP) can handle strides, so both have low accuracy scores.

### *3.4 How can we take account of strides?*

We note that any information theoretic measurement will give the same values for a contiguous sequence of increasing values such as $(1, 2, 3, \ldots)$ as for a non-repeating sequence of unrelated values such as $(1, 42, -5, \ldots)$, yet SVP will clearly achieve 100% accuracy on the first sequence. The reason that SVP can succeed here is that it takes into account the fact that the values in the first sequence are *ordinal*, related to each other. Shannon's Entropy assumes *categorical* random variables, i.e. the fact that a 2 follows a 1, is no different from a 42 following a 1—the return values are treated as discrete events from a multinomial probability distribution. One obvious route might

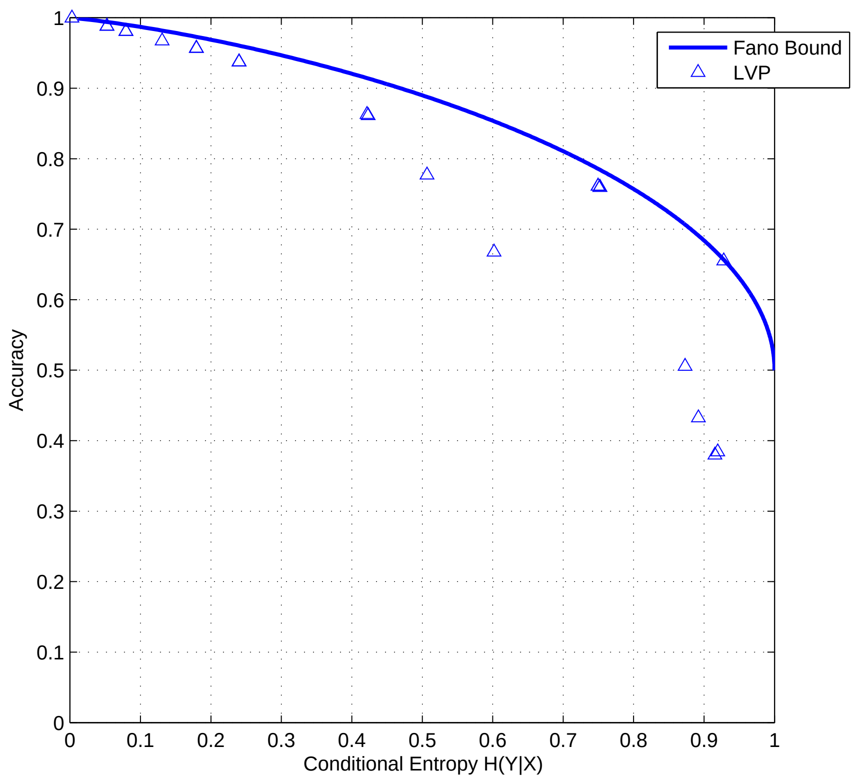


Fig. 5. Fano inequality bounds LVP predictability for boolean methods

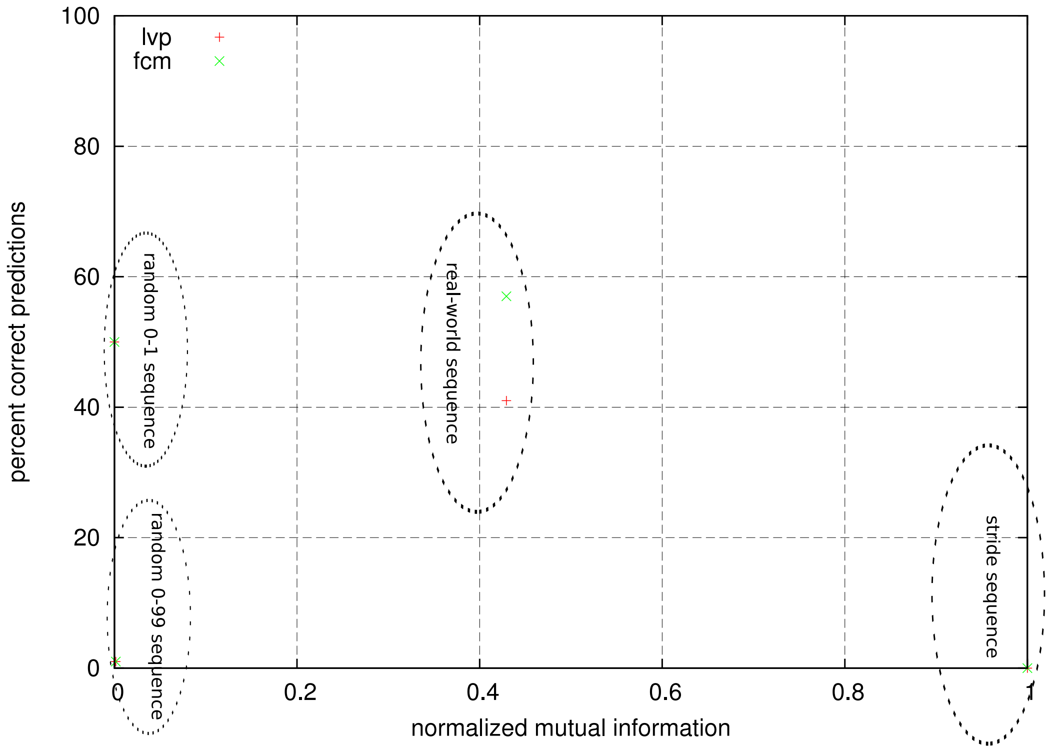


Fig. 6. Mutual information versus predictability for selected methods

be to compute information theoretic properties of the *strides*; in the first sequence above we note that the strides would show up as entirely predictable, having a redundancy of 1. The question of course arises, when do we trust our measurements on the strides as opposed to our measurements on the values themselves?

## *3.5 Could information theoretic measurements be used as* switching *criteria to decide which member of a hybrid predictor will be correct?*

Hybrid predictors are arguably one of the most successful techniques applied in this field [23]. Generally hybrids incorporate a computational predictor such as SVP and a context-based predictor such as FCM. The idea is not constrained to return value prediction however. The field of *ensemble learning* [3,25] is concerned with building machine learning techniques that can efficiently *combine* different predictors, possibly *switching* between them, so as to exploit their strengths and weaknesses in different situations. It is entirely plausible that this field will provide the necessary background for a major boost in the power of return value prediction for thread-level speculation.

## *3.6 How can we implement these measures and procedures efficiently, such that their overhead will not negate their utility when incorporated into a chip?*

To the best of our knowledge, there are no existing hardware implementations of return value prediction units. Lipasti and Shen [15] estimate that 'super-speculative prediction structures, including those for branch prediction' would cost 'roughly 36 million transistors' out of a total of 180 million transistors for a processor core. This is a significant cut of the transistor budget.

These proposed prediction units only perform simple computational operations, including comparisons, increments and hashes. An alternative implementation, based on information theory, would require much more complicated calculations involving logarithms, multiplications and divisions. Ideally, such computation would make use of existing arithmetic and floating-point units, rather than reimplementing specialized circuitry.

However, we envisage that value prediction will only occur for return values at the end of methods, rather than for all instructions. Since the frequency of method execution is much lower than the frequency of instruction execution, it should be possible to implement return value prediction entirely in *software*. For VM-based systems, the prediction mechanism can be incorporated directly into the VM itself. There is an existing precedent for this: SableSpMT [19] performs software return value prediction within the Sable JVM. For this study, we have used Jikes RVM [1,2], which is a powerful adaptive JVM. Jikes RVM already has features for runtime method-level profiling. It would be extremely straightforward to extend this profiling to perform dynamic return value prediction.

In such a system, no extra hardware support is necessary! An all-software solution is extremely flexible and extensible. This is a clear advantage for VM-based architectures. Note that hardware support is still required to deploy thread-level speculation in order to take advantage of predictability, but such hardware issues are beyond the scope of this paper.

## 4 Future Work

Much work remains to be done. Once we have satisfactorily resolved the issues highlighted in Section 3, we hope to implement a system in which these information theoretic calculations can occur at runtime, rather than as a post-processing step. It is well known that programs exhibit phased behaviour at runtime [9,12]. We conjecture that this is true of method predictability. We hope to spot changes in predictability behaviour dynamically, and make adaptive prediction decisions in response to our online measurements.

## References

[1] Alpern, B., C. R. Attanasio, J. J. Barton, M. G. Burke, P. Cheng, J.-D. Choi, A. Cocchi, S. J. Fink, D. Grove, M. Hind, S. F. Hummel, D. Lieber, V. Litvinov, M. F. Mergen, T. Ngo, J. R. Russell, V. Sarkar, M. J. Serrano, J. C. Shepherd, S. E. Smith, V. C. Sreedhar, H. Srinivasan and J. Whaley, *The Jalapeño virtual machine*, IBM System Journal **39** (2000), pp. 211–238.

[2] Alpern, B., S. Augart, S. M. Blackburn, M. Butrico, A. Cocchi, P. Cheng, J. Dolby, S. Fink, D. Grove, M. Hind, K. S. McKinley, M. Mergen, J. E. B. Moss, T. Ngo, V. Sarkar and M. Trapp, *The Jikes research virtual machine project: Building an open source research community*, IBM Systems Journal **44** (2005), pp. 1–19.

[3] Brown, G., J. Wyatt, R. Harris and X. Yao, *Diversity creation methods: A survey and categorisation*, Journal of Information Fusion **6** (2005), pp. 5–20.

[4] Chen, M. and K. Olukotun, *Exploiting method-level parallelism in single-threaded Java programs*, in: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 1998, pp. 176–184.

[5] Clark, D., S. Hunt and P. Malacaria, *Quantified interference for a while language*, Electronic Notes in Theoretical Computer Science **112** (2005), pp. 149–166.

[6] Cousot, P. and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, in: *Proceedings of the 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, 1977, pp. 238–252.

[7] Cover, T. M. and J. A. Thomas, “Elements of Information Theory,” Wiley, 1991.

[8] Derisavi, S., H. Hermanns and W. Sanders, *Optimal state-space lumping in Markov chains*, Information Processing Letters **87** (2003), pp. 309–315.

[9] Duesterwald, E., C. Cascaval and S. Dwarkadas, *Characterizing and predicting program behavior and its variability*, in: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2003, pp. 220–231.

[10] Gabbay, F. and A. Mendelson, *Speculative execution based on value prediction*, Technical Report EE Department TR 1080, Technion: Israel Institute of Technology (1996).

[11] Gabbay, F. and A. Mendelson, *Using value prediction to increase the power of speculative execution hardware*, ACM Transactions on Computer Systems **16** (1998), pp. 234–270.

[12] Georges, A., D. Buytaert, L. Eeckhout and K. D. Bosschere, *Method-level phase behavior in Java workloads*, in: *Proceedings of the 19th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications*, 2004, pp. 270–287.

[13] Hammerstrom, D. W. and E. S. Davidson, *Information content of CPU memory referencing behavior*, in: *Proceedings of the 4th Annual Symposium on Computer Architecture*, 1977, pp. 184–192.

[14] Hu, S., R. Bhargava and L. K. John, *The role of return value prediction in exploiting speculative method-level parallelism*, Journal of Instruction-Level Parallelism **5** (2003).

[15] Lipasti, M. and J. Shen, *Superspeculative microarchitecture for beyond AD 2000*, IEEE Computer **30** (1997), pp. 59–66.

[16] Lipasti, M. H. and J. P. Shen, *Exceeding the dataflow limit via value prediction*, in: *Proceedings of the 29th Annual ACM/IEEE International Symposium on Microarchitecture*, 1996, pp. 226–237.

[17] Lipasti, M. H., C. B. Wilkerson and J. P. Shen, *Value locality and load value prediction*, in: *Proceedings of the Seventh International Conference on Architectural Support for Programming Languages and Operating Systems*, 1996, pp. 138–147.

[18] Oplinger, J. T., D. L. Heine and M. S. Lam, *In search of speculative thread-level parallelism*, in: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 1999, pp. 303–313.

[19] Pickett, C. J. and C. Verbrugge, *Return value prediction in a Java virtual machine*, in: *Proceedings of the Second Value-Prediction and Value-Based Optimization Workshop*, 2004, pp. 40–47.

[20] Ruf, E., *Context-insensitive alias analysis reconsidered*, in: *Proceedings of the ACM SIGPLAN 1995 Conference on Programming Language Design and Implementation*, 1995, pp. 13–22.

[21] Sazeides, Y. and J. Smith, *Implementations of context based value predictors*, Technical Report TR-ECE-97-8, Department of Electrical Computer Engineering, University of Wisconsin-Madison (1997).

[22] Sazeides, Y. and J. E. Smith, *The predictability of data values*, in: *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 248–258.

[23] Wang, K. and M. Franklin, *Highly accurate data value prediction using hybrid predictors*, in: *Proceedings of the 30th Annual ACM/IEEE International Symposium on Microarchitecture*, 1997, pp. 281–290.

[24] Warg, F. and P. Stenström, *Limits on speculative module-level parallelism in imperative and object-oriented programs on CMP platforms*, in: *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2001, pp. 221–230.

[25] Woods, K., W. P. Kegelmeyer and K. Bowyer, *Combination of multiple classifiers using local accuracy estimates*, IEEE Transactions on Pattern Analysis and Machine Intelligence **19** (1997), pp. 405–410.

## A Experimental Setup

The experiments were conducted using the Jikes RVM adaptive runtime compilation system for Java programs [1,2]. We modified the code generation system so that when a 32-bit integer return JVM bytecode instruction (`ireturn`) is compiled, the code sequence also dumps out the return value and method identifier to a trace file. Note that the `ireturn` instruction is used to return values from Java primitive types `boolean`, `byte`, `char`, `short` and `int`. It is necessary to use high-level debugging information in the Java class file to determine the actual type of return value for each method. Note also that each method has its own unique integer identifier.

We took the following steps to ensure that the trace data is as realistic as possible.

(i) All experiments are performed using the *optimizing* version of the Jikes RVM compiler. This performs standard JVM optimizations including method inlining. So all methods should be reasonably sized and non-trivial.

(ii) All experiments run the SPEC JVM98 benchmark suite for trace file data generation. This is an industry-standard Java benchmark suite. It consists of eight compute-intensive standalone Java applications.

(iii) The benchmark programs take maximum sized (`s100`) input data sets, representing real-world workloads. It is to be noted that previous studies into Java method return value predictability [24,14] used much smaller (`s1`) input data sets for the same benchmarks.

Once the trace files have been generated, simple script programs postprocess the return value streams on a per-method basis, to simulate the behaviour of different value predictors, or to compute information theoretic quantities such as entropy and redundancy.