Determinism, Complexity, and Predictability in Computer Performance

Joshua Garland [*1], Ryan James [†1], and Elizabeth Bradley [‡1,2]

[1]Department of Computer Science, University of Colorado at Boulder, Colorado, USA
[2]Santa Fe Institute, New Mexico, USA

January 27, 2014

Abstract

Computers are deterministic dynamical systems [**?**]. Among other things, that implies that one should be able to use deterministic forecast rules to predict aspects of their behavior. That statement is sometimes—but not always—true. The memory and processor loads of some simple programs are easy to predict, for example, but those of more-complex programs like `gcc` are not. The goal of this paper is to determine why that is the case. We conjecture that, in practice, complexity can effectively overwhelm the predictive power of deterministic forecast models. To explore that, we build models of a number of performance traces from different programs running on different Intel-based computers. We then calculate the *permutation entropy*—a temporal entropy metric that uses ordinal analysis—of those traces and correlate those values against the prediction success.

1 Introduction

Computers are among the most complex engineered artifacts in current use. Modern microprocessor chips contain multiple processing units and multi-layer memories, for instance, and they use complicated hardware/software strategies to move data and threads of computation across those resources. These features—along with all the others that go into the design of these chips—make the patterns of their processor loads and memory accesses highly complex and hard to predict. Accurate forecasts of these quantities, if one could construct them, could be used to improve computer design. If one could predict that a particular computational thread would be bogged down for the next 0.6 seconds waiting for data from main memory, for instance, one could save power by putting that thread on hold for that time period (e.g., by migrating it to a processing unit whose clock speed is scaled back). Computer performance traces are, however, very complex. Even a simple "microkernel," like a three-line loop that repeatedly initializes a matrix in column-major order, can

[*]joshua.garland@colorado.edu
[†]ryan.james@colorado.edu
[‡]lizb@colorado.edu

produce *chaotic* performance traces [**?**], as shown in Figure **??**, and chaos places fundamental limits on predictability.

Figure 1: A small snippet of the L2 cache miss rate of `col_major`, a three-line C program that repeatedly initializes a matrix in column-major order, running on an Intel Core Duo®-based machine. Even this simple program exhibits chaotic performance dynamics.

The computer systems community has applied a variety of prediction strategies to traces like this, most of which employ regression. An appealing alternative builds on the recently established fact that computers can be effectively modeled as deterministic nonlinear dynamical systems [**?**]. This result implies the existence of a deterministic forecast rule for those dynamics. In particular, one can use *delay-coordinate embedding* to reconstruct the underlying dynamics of computer performance, then use the resulting model to forecast the future values of computer performance metrics such as memory or processor loads [**?**]. In the case of simple microkernels like the one that produced the trace in Figure **??**, this deterministic modeling and forecast strategy works very well. In more-complicated programs, however, such as speech recognition software or compilers, this forecast strategy—as well as the traditional methods—break down quickly.

This paper is a first step in understanding when, why, and how deterministic forecast strategies fail when they are applied to deterministic systems. We focus here on the specific example of computer performance. We conjecture that the complexity of traces from these systems—which results from the inherent dimension, nonlinearity, and nonstationarity of the dynamics, as well as from measurement issues like noise, aggregation, and finite data length—can make those deterministic signals *effectively* unpredictable. We argue that *permutation entropy* [**?**], a method for measuring the entropy of a real-valued-finite-length time series through ordinal analysis, is an effective way to explore that conjecture. We study four examples—two simple microkernels and two complex programs from the SPEC benchmark suite—running on different Intel-based machines. For each program, we calculate the permutation entropy of the processor load (instructions per cycle) and memory-use efficiency (cache-miss rates), then compare that to the prediction accuracy attainable for that trace using a simple deterministic model.

It is worth taking a moment to consider the theoretical possibility of this task. We are not attempting to predict the state of the CPU at an arbitrary point in the future — this, at least with perfect accuracy, would be tantamount to solving the halting problem. What we are attempting is to predict aspects or functions of the running of the CPU: instructions executed per second, cache misses per 100,000 instructions, and similar statistics. Prediction of these quantities at some finite time in the future, even with perfect accuracy, does not violate the Rice-Shapiro theorem.

2 Modeling Computer Performance

Delay-coordinate embedding allows one to reconstruct a system's full state-space dynamics from a *single* scalar time-series measurement—provided that some conditions hold regarding that data. Specifically, if the underlying dynamics and the measurement function—the mapping from the unknown state vector $\vec{X}$ to the scalar value x that one is measuring—are both smooth and generic, Takens [**?**] formally proves that the delay-coordinate map

$$F(\tau, m)(x) = ([x(t)\ x(t+\tau)\ \ldots\ x(t+m\tau)])$$

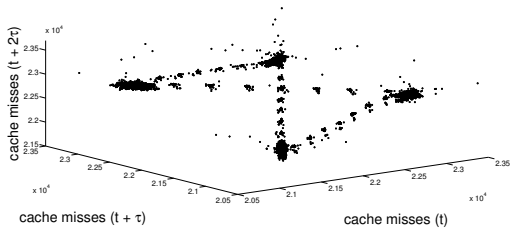


Figure 2: A 3D projection of a delay-coordinate embedding of the trace from Figure ?? with a delay (τ) of 100,000 instructions.

from a d-dimensional smooth compact manifold M to Re^{2d+1}, where t is time, is a diffeomorphism on M—in other words, that the reconstructed dynamics and the true (hidden) dynamics have the same topology.

This is an extremely powerful result: among other things, it means that one can build a formal model of the full system dynamics without measuring (or even knowing) every one of its state variables. This is the foundation of the modeling approach that is used in this paper. The first step in the process is to estimate values for the two free parameters in the delay-coordinate map: the delay τ and the dimension m. We follow standard procedures for this, choosing the first minimum in the average mutual information as an estimate of τ [?] and using the false-near(est) neighbor method of [?], with a threshold of 10%, to estimate m. A plot of the data from Figure ??, embedded following this procedure, is shown in Figure ??. The coordinates of each point on this plot are differently delayed elements of the `col_major` L2 cache miss rate time series $y(t)$: that is, $y(t)$ on the first axis, $y(t+\tau)$ on the second, $y(t+2\tau)$ on the third, and so on. Structure in these kinds of plots—clearly visible in Figure ??—is an indication of determinism[1]. That structure can also be used to build a forecast model.

Given a nonlinear model of a deterministic dynamical system in the form of a delay-coordinate embedding like Figure ??, one can build deterministic forecast algorithms by capturing and exploiting the geometry of the embedding. Many techniques have been developed by the dynamical systems community for this purpose (e.g., [?, ?]). Perhaps the most straightforward is the "Lorenz method of analogues" (LMA), which is essentially nearest-neighbor prediction in the embedded state space [?]. Even this simple algorithm—which builds predictions by finding the nearest neighbor in the embedded space of the given point, then taking that neighbor's path as the forecast—works quite well on the trace in Figure ??, as shown in Figure ??. On the other hand, if we use the same approach to forecast the processor load[2] of the `482.sphinx3` program from the SPEC cpu2006 benchmark suite, running on an Intel i7®-based machine, the prediction is far less accurate; see Figure ??.

Table ?? presents detailed results about the prediction accuracy of this algorithm on four different examples: the `col_major` and `482.sphinx3` programs in Figures ?? and ??, as well as another simple microkernel that initializes the same matrix as `col_major`, but in row-major order, and another complex program (`403.gcc`) from the SPEC cpu2006 benchmark suite. Both microkernels

[1] A deeper analysis of Figure ??—as alluded to on the previous page—supports that diagnosis, confirming the presence of a chaotic attractor in these cache-miss dynamics, with largest Lyapunov exponent $\lambda_1 = 8000 \pm 200$ instructions, embedded in a 12-dimensional reconstruction space [?].

[2] Instructions per cycle, or IPC

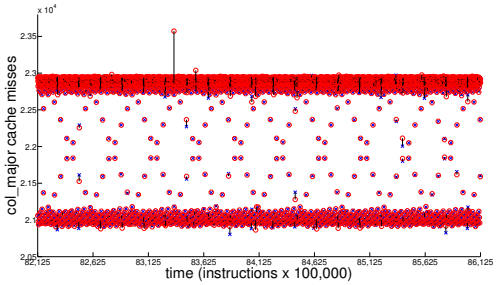


Figure 3: A forecast of the last 4,000 points of the signal in Figure **??** using an LMA-based strategy on the embedding in Figure **??**. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

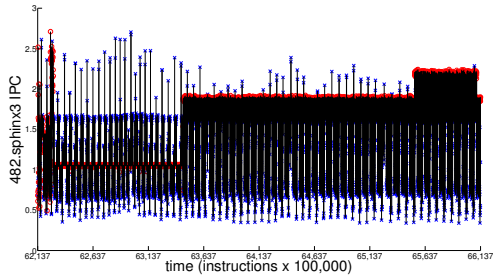


Figure 4: An LMA-based forecast of the last 4,000 points of a processor-load performance trace from the `482.sphinx3` benchmark. Red circles and blue ×s are the true and predicted values, respectively; vertical bars show where these values differ.

were run on the Intel Core Duo® machine; both SPEC benchmarks were run on the Intel i7® machine. We calculated a figure of merit for each prediction as follows. We held back the last k elements[3] of the N points in each measured time series, built the forecast model by embedding the first $N-k$ points, used that embedding and the LMA method to predict the next k points, then computed the Root Mean Squared Error (RMSE) between the true and predicted signals:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{k}(c_i - \hat{p_i})^2}{k}}$$

To compare the success of predictions across signals with different units, we normalized RMSE as follows:

$$nRMSE = \frac{RMSE}{X_{max,obs} - X_{min,obs}}$$

Table 1: Normalized root mean squared error (nRMSE) of 4000-point predictions of memory & processor performance from different programs.

	cache miss rate	instrs per cycle
`row_major`	0.0324	0.0778
`col_major`	0.0080	0.0161
`403.gcc`	0.1416	0.2033
`482.sphinx3`	0.2032	0.3670

The results in Table **??** show a clear distinction between the two microkernels, whose future behavior can be predicted effectively using this simple deterministic modeling strategy, and the more-complex SPEC benchmarks, for which this prediction strategy does not work nearly as well. This begs the question: If these traces all come from deterministic systems—computers—then why are they not equally predictable? Our conjecture is that the sheer complexity of the dynamics of the SPEC benchmarks running on the Intel i7® machine make them effectively impossible to predict.

3 Measuring Complexity

For the purposes of this paper, one can view entropy as a measure of complexity and predictability in a time series. A high-entropy time series is almost completely unpredictable—and conversely. This can be made more rigorous: Pesin's relation [**?**] states that in chaotic dynamical systems, the Shannon entropy rate is equal to the sum of the positive Lyapunov exponents, λ_i. The Lyapunov exponents directly quantify the rate at which nearby states of the system will diverge with time: $|\Delta x(t)| \approx e^{\lambda t} |\Delta x(0)|$. The faster the divergence, the more difficult prediction becomes.

Utilizing entropy as a measure of temporal complexity is by no means a new idea [**?**, **?**]. Its effective usage requires categorical data: $x_t \in \mathcal{S}$ for some finite or countably infinite *alphabet* $\mathcal{S}$, whereas data taken from real-world systems is effectively real-valued. To get around this, one must discretize the data— typically achieved by binning. Unfortunately, this is rarely a good solution to

[3]Several different prediction horizons were analyzed in our experiment; the results reported in this paper are for k=4000

the problem, as the binning of the values introduces an additional dynamic on top of the intrinsic dynamics whose entropy is desired. The field of symbolic dynamics studies how to discretize a time series in such a way that the intrinsic behavior is not perverted, but these methods are fragile in the face of noise and require further understanding of the underlying system, which defeats the purpose of measuring the entropy in the first place.

Bandt and Pompe introduced the *permutation entropy* (PE) as a "natural complexity measure for time series" [**?**]. Permutation entropy employs a method of discretizing real-valued time series that follows the intrinsic behavior of the system under examination. Rather than looking at the statistics of sequences of values, as is done when computing the Shannon entropy, permutation entropy looks at the statistics of the *orderings* of sequences of values using ordinal analysis. Ordinal analysis of a time series is the process of mapping successive time-ordered elements of a time series to their value-ordered permutation of the same size. By way of example, if $(x_1, x_2, x_3) = (9, 1, 7)$ then its *ordinal pattern*, $\phi(x_1, x_2, x_3)$, is 231 since $x_2 \leq x_3 \leq x_1$. This method has many features; among other things, it is generally robust to observational noise and requires no knowledge of the underlying mechanisms.

Definition (Permutation Entropy)**.** *Given a time series* $\{x_t\}_{t=1,\ldots,T}$*. Define* $\mathcal{S}_n$ *as all* $n!$ *permutations* π *of order* n*. For each* $\pi \in \mathcal{S}_n$ *we determine the relative frequency of that permutation occurring in* $\{x_t\}_{t=1,\ldots,T}$*:*

$$p(\pi) = \frac{|\{t | t \leq T - n, \phi(x_{t+1}, \ldots, x_{t+n}) = \pi\}|}{T - n + 1}$$

Where $|\cdot|$ *is set cardinality. The* permutation entropy *of order* $n \geq 2$ *is defined as*

$$H(n) = -\sum_{\pi \in \mathcal{S}_n} p(\pi) \log_2 p(\pi)$$

Notice that $0 \leq H(n) \leq \log_2(n!)$ [**?**]. With this in mind, it is common in the literature to normalize permutation entropy as follows: $\frac{H(n)}{\log_2(n!)}$. With this convention, "low" entropy is close to 0 and "high" entropy is close to 1. Finally, it should be noted that the permutation entropy has been shown to be identical to the Shannon entropy for many large classes of systems [**?**].

Here we will be utilizing a variation of the permutation entropy, the *weighted permutation entropy* (WPE) [**?**]. The weighted permutation entropy attempts to correct for observational noise which is larger than some trends in the data, but smaller than the larger scale features — for example, a signal that switches between two fixed points with noise about those fixed points. The weighted permutation entropy would be dominated by the switching rather than by the stochastic fluctuation. To accomplish this, the *weight* of a permutation is taken into account:

$$w(x_{t+1:t+n}) = \frac{1}{n} \sum_{x_i \in \{x_{t+1:t+n}\}} (x_i - \bar{x}_{t+1:t+n})^2$$

where $x_{t+1:t+n}$ is a sequence of values $x_{t+1}, \ldots, x_{t+n}$, and $\bar{x}_{t+1:t+n}$ is the arithmetic mean of those values.

The weighted probability of a permutation is then:

$$p_w(\pi) = \frac{\sum_{t \leq T-n} w(x_{t+1:t+n}) \cdot \delta(\phi(x_{t:t+n}), \pi)}{\sum_{t \leq T-n} w(x_{t+1:t+n})}$$

where $\delta(x, y)$ is 1 if $x = y$ and 0 otherwise. Effectively, this weighted probability enhances permutations involved in "large" features and demotes permutations which are small in amplitude relative to the features of the time series. The weighted permutation entropy is then:

$$H_w(n) = -\sum_{\pi \in \mathcal{S}_n} p_w(\pi) \log_2 p_w(\pi),$$

which can also be normalized by dividing by $\log_2(n!)$, and will be in all the results of this paper.

In practice, calculating permutation entropy and weighted permutation entropy involves choosing a good value for the word length n. The primary consideration is that the value be large enough that forbidden ordinals are discovered, yet small enough that reasonable statistics over the ordinals are gathered: e.g.:

$$n = \operatorname*{argmax}_{\ell} \{T \gtrapprox 100\ell!\},$$

assuming an average of 100 counts per ordinal is sufficient. In the literature, $3 \leq n \leq 6$ is a standard choice — generally without any formal justification. In theory, the permutation entropy should reach an asymptote with increasing n, but that requires an arbitrarily long time series. In practice, what one should do is calculate the *persistent* permutation entropy by increasing n until the result converges, but data length issues can intrude before that convergence is reached.

The weighted permutation entropy for the `SVD` program is given in Fig. [**?**]. To generate this image a window of 5,000 values slid over the time series. Within each of those windows, the statistics over words of length 4 are computed and the WPE is calculated. The gray bands denote regions where the 5,000 value window overlapped visually-distinct regimes. It can be seen that the behaviors of the weighted permutation entropy vary between regimes.

The relationship between prediction accuracy and the weighted permutation entropy (WPE) is much as we conjectured: performance traces with high WPE — those whose temporal complexity is high, in the sense that the entropy generation per value is approximately equal to the total entropy per value thus little information is being propagated forward in time — are indeed harder to predict using the simple deterministic forecast model described in the previous section. See Fig. [**?**] for the distribution. The effects of changing the word length n are also interesting: using a longer word length generally lowers the WPE — a natural consequence of finite-length data — but the falloff is less rapid in some traces than in others, suggesting that those values are closer to the theoretical asymptote that exists for perfect data. The persistent WPE values of 0.5–0.6 for the `col_major` trace are consistent with dynamical chaos, further corroborating the results of [**?**].

In figure **??** we directly compare the performance of LMA to the value of the weighted permutation entropy for all runs of each program under consideration. Largely, the data supports our hypothesized relationship between prediction and WPE, namely that they are proportional. `SVD` regime 1 bucks this trend a bit, having a larger WPE than might be expected from how well its predictions are. We believe that this simply due to a limitation of the WPE, namely that in the absence of any large features, as all the other regimes have, the WPE falls back to the standard PE which the noisy behavior drives toward 1.0.

Figure **??** demonstrates that the LMA prediction algorithm is better suited to the dynamics exhibited in the computer traces considered here, except for the 1$^{\text{st}}$ regime of the `SVD` program. There ARIMA performs slightly better.

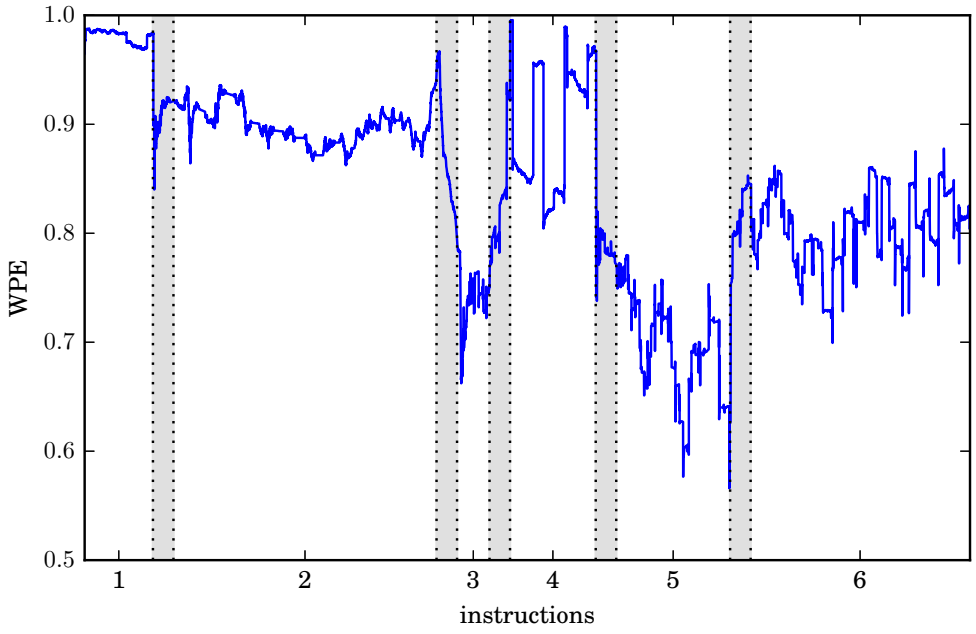


Figure 5: The weighted permutation entropy of one run of SVD. The gray bands are regions where the window overlaps regimes. The window size used is $5,000 \times 100,000$ instructions and the word length is 4.

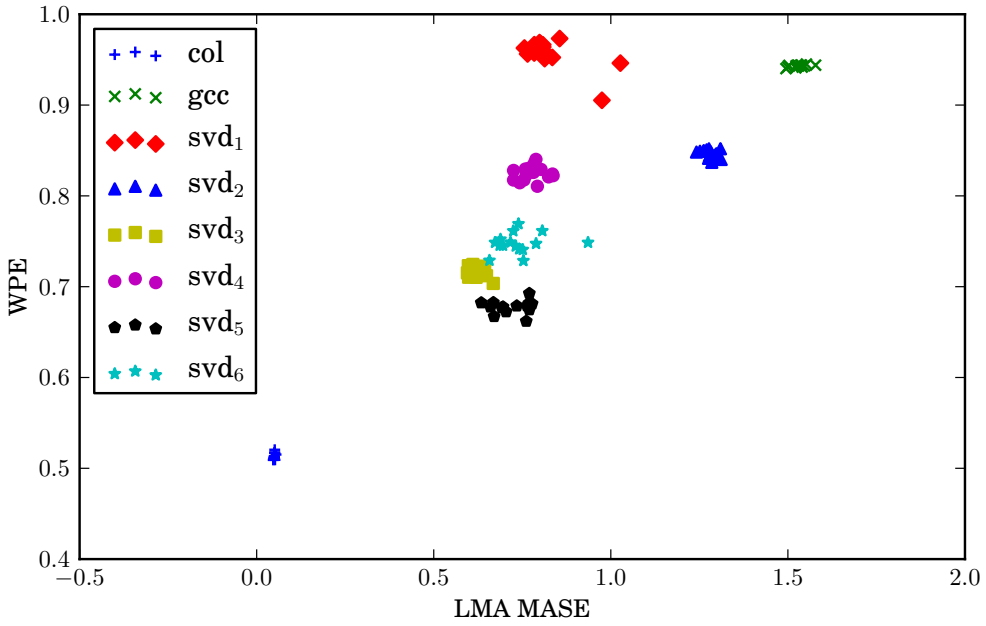


Figure 6: The MASE of LMA vs weighted permutation entropy. For each of these, the word length used is 5.

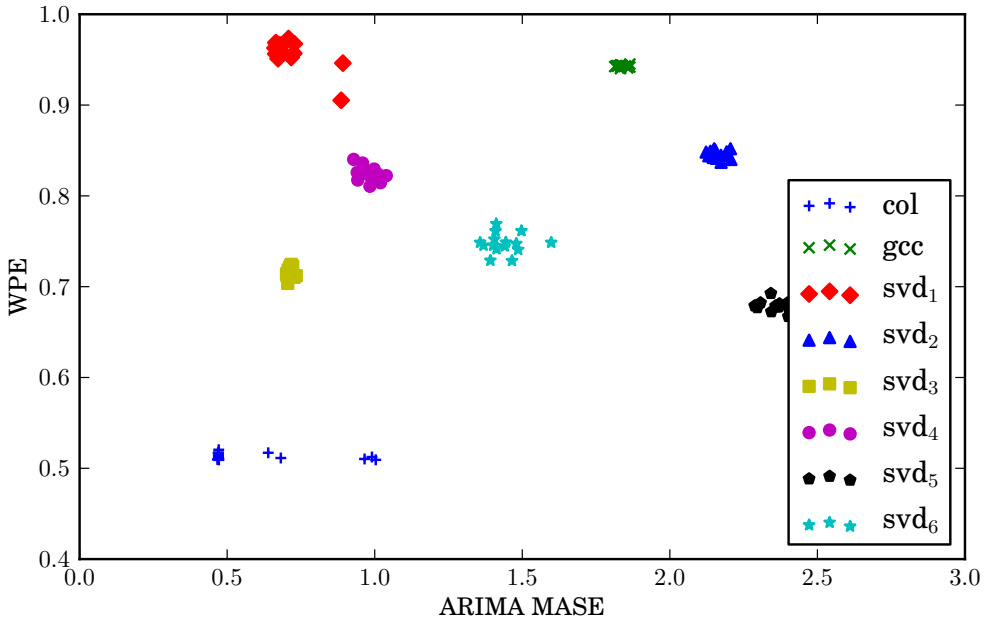


Figure 7: The MASE of ARIMA vs weighted permutation entropy. For each of these, the word length used is 5.

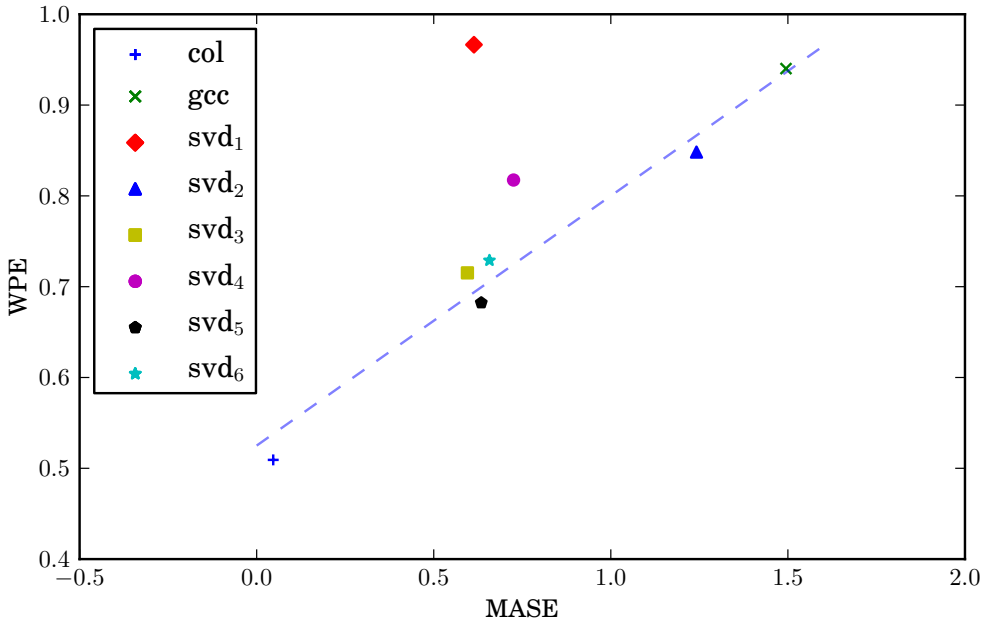


Figure 8: The best MASE among all runs and prediction methods vs weighted permutation entropy. For each of these, the word length used is 5.

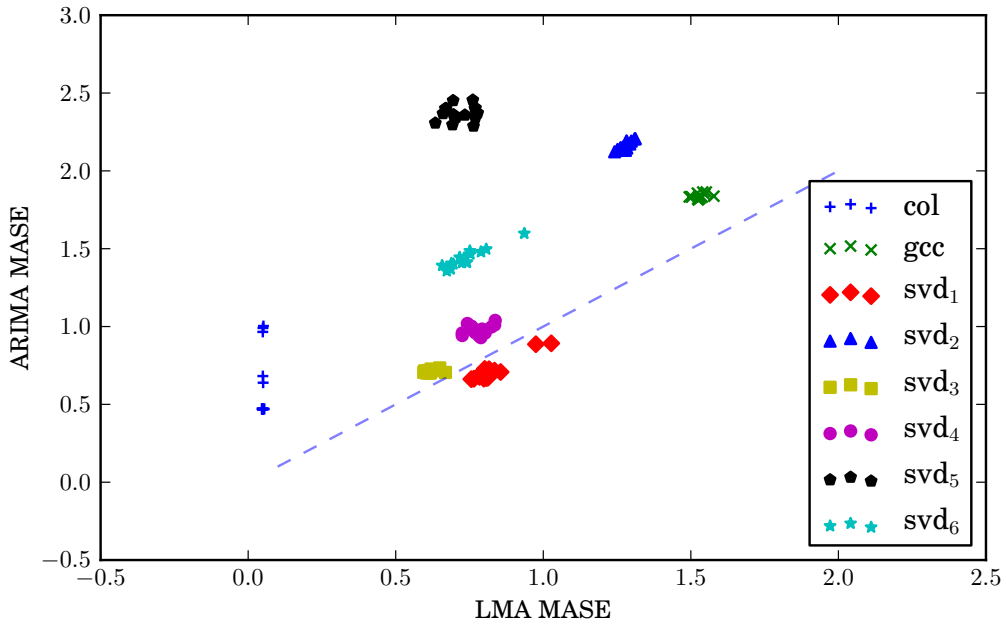


Figure 9: The MASE values for LMA against ARIMA. The dashed line is the identity.

4 Conclusions & Future Work

The results presented here suggest that permutation entropy—a ordinal calculation of forward information transfer in a time series—is an effective metric for predictability of computer performance traces. Experimentally, traces with a persistent PE $\gtrapprox 0.97$ have a natural level of complexity that may overshadow the inherent determinism in the system dynamics, whereas traces with PE $\lessapprox 0.7$ seem to be highly predictable (viz., at least an order of magnitude improvement in nRMSPE).

If information is the limit, then gathering and using more information is an obvious next step. There is an equally obvious tension here between data length and prediction speed: a forecast that requires half a second to compute is not useful for the purposes of real-time control of a computer system with a MHz clock rate. Another alternative is to sample several system variables simultaneously and build multivariate delay-coordinate embeddings. Existing approaches to that are computationally prohibitive [**?**]. We are working on alternative methods that sidestep that complexity.

5 New Figures and Tables

Possible new error measure

$$q_t = \frac{e_t}{\frac{1}{n-1}\sum_{i=2}^{n}|Y_i - Y_{i-1}|}$$

$$MASE = mean(|q_t|)$$

Helpful to remember that $F_i = Y_{i-1}$ for random walk prediction

"When $MASE < 1$, the proposed method gives, on average, smaller errors than the one-step errors from the naïve method. If multi-step forecasts are being computed, it is possible to scale by the in- sample MAE[[Mean Absolute Error(mean($|e_t|$)]] computed from multi-step naïve forecasts."

"The errors have been scaled by the one-step in- sample forecast errors from the naïve method, and then averaged across all series. So a value of 2 indicates that the out-of-sample forecast errors are, on average, about twice as large as the in-sample one-step forecast errors from the naïve method. Because the scaling is based on one-step forecasts, the scaled errors for multi-step forecasts are typically larger than one. "

We use this error measure with n at the start of the predicted chunk.

Acknowledgment

This work was partially supported by NSF grant #CMMI-1245947 and ARO grant #W911NF-12-1-0288.

Table 2: Embedding Parameters for reference if needed.

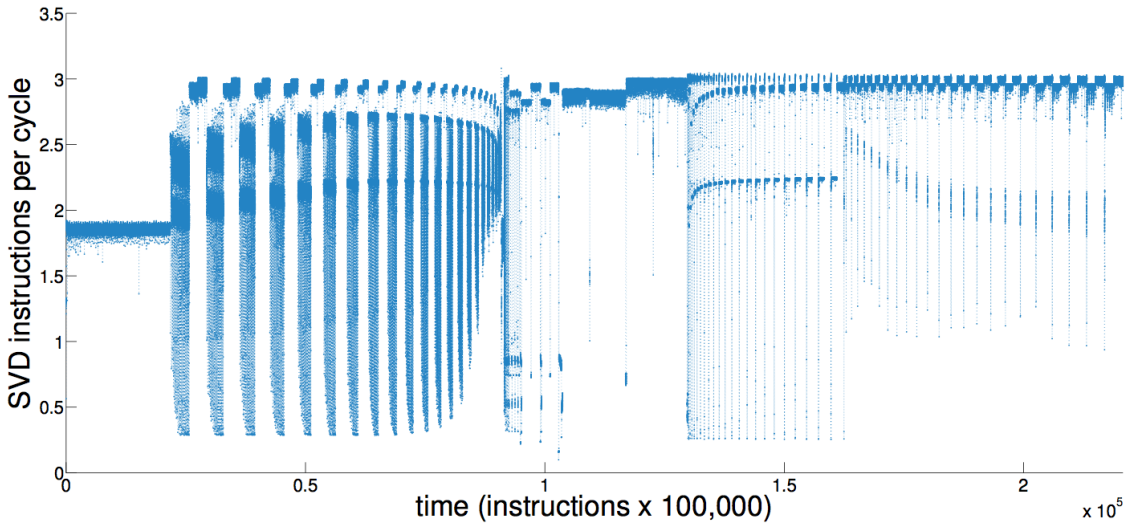
	τ	m
gcc	10	13
col_major	2	12
SVD_Full	10	12
SVD_IPC_Regime1	5	14
SVD_IPC_Regime2	10	12
SVD_IPC_Regime3	2	9
SVD_IPC_Regime4	3	11
SVD_IPC_Regime5	23	10
SVD_IPC_Regime6	30	12

Table 3: Average nRMSE over 15 runs for each signal and average wpe at word length 5 and 6 for each signal.

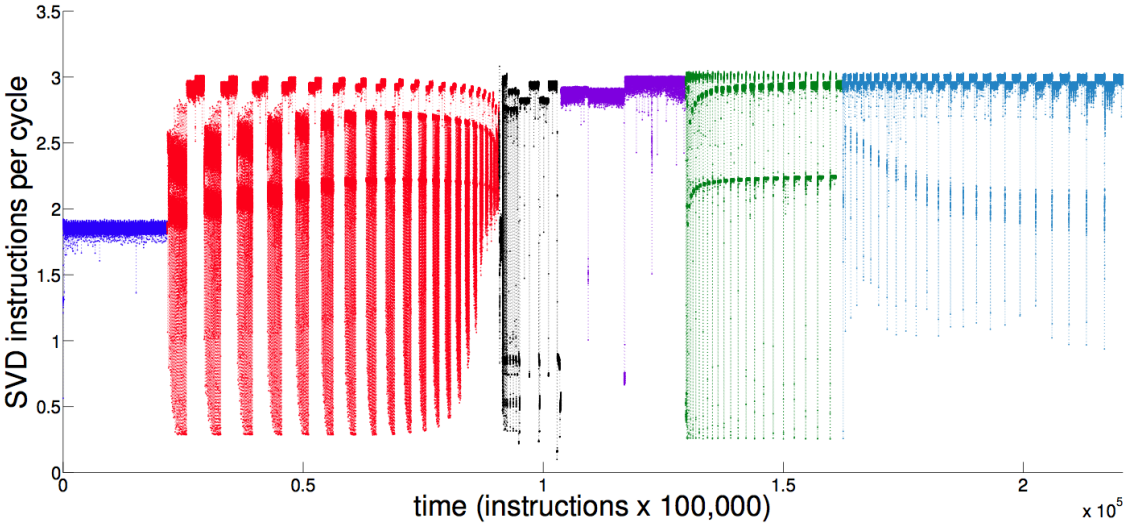
	nRMSE LMA	nRMSE naïve	$l = 5$	$l = 6$
gcc	0.1407 ± 0.0063	0.1487 ± 0.0066	0.9510 ± 0.0011	0.9430 ± 0.0013
col_major	0.0252 ± 0.0061	0.1975 ± 0.0436	0.5636 ± 0.0031	0.5131 ± 0.0034
SVD_IPC_Regime1	0.1680 ± 0.0317	0.3517 ± 0.5223	0.9761 ± 0.0084	0.9572 ± 0.0156
SVD_IPC_Regime2	0.1716 ± 0.0043	0.1762 ± 0.0012	0.8760 ± 0.0052	0.8464 ± 0.0044
SVD_IPC_Regime3	0.0507 ± 0.0011	0.5413 ± 0.0005	0.7768 ± 0.0073	0.7157 ± 0.0056
SVD_IPC_Regime4	0.1288 ± 0.0471	0.2308 ± 0.0867	0.9073 ± 0.0080	0.8246 ± 0.0077
SVD_IPC_Regime5	0.0235 ± 0.0022	0.1306 ± 0.0003	0.7333 ± 0.0076	0.6776 ± 0.0068
SVD_IPC_Regime6	0.0196 ± 0.0022	0.0508 ± 0.0003	0.8101 ± 0.0135	0.7475 ± 0.0106

Table 4: Average MASE for 1-step predictions at a 10% prediction horizon over 15 runs for each signal and average wpe at word length 5 and 6 for each signal.

	MASE LMA	MASE ARIMA	MASE naïve	$l = 5$	$l = 6$
gcc	1.5296 ± 0.0214	1.8366 ± 0.0157	1.7970 ± 0.0095	0.9510 ± 0.0011	0.9430 ± 0.0013
col_major	0.0500 ± 0.0018	0.5989 ± 0.2114	0.5707 ± 0.0017	0.5636 ± 0.0031	0.5131 ± 0.0034
SVD_IPC_Regime1	0.8273 ± 0.0755	0.7141 ± 0.0745	2.6763 ± 4.3282	0.9761 ± 0.0084	0.9572 ± 0.0156
SVD_IPC_Regime2	1.2789 ± 0.0196	2.1626 ± 0.0265	3.0543 ± 0.0404	0.8760 ± 0.0052	0.8464 ± 0.0044
SVD_IPC_Regime3	0.6192 ± 0.0209	0.7129 ± 0.0096	31.3857 ± 0.2820	0.7768 ± 0.0073	0.7157 ± 0.0056
SVD_IPC_Regime4	0.7789 ± 0.0358	0.9787 ± 0.0321	2.6613 ± 0.0739	0.9073 ± 0.0080	0.8246 ± 0.0077
SVD_IPC_Regime5	0.7177 ± 0.0483	2.3700 ± 0.0505	20.8703 ± 0.1915	0.7333 ± 0.0076	0.6776 ± 0.0068
SVD_IPC_Regime6	0.7393 ± 0.0682	1.4379 ± 0.0609	2.1967 ± 0.0830	0.8101 ± 0.0135	0.7475 ± 0.0106

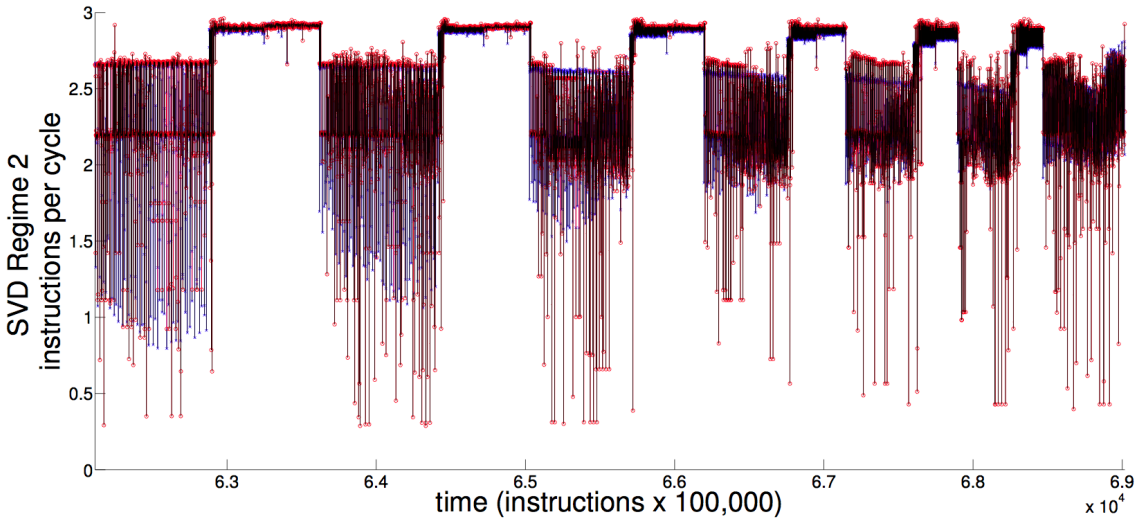


(a) SVD IPC without coloring

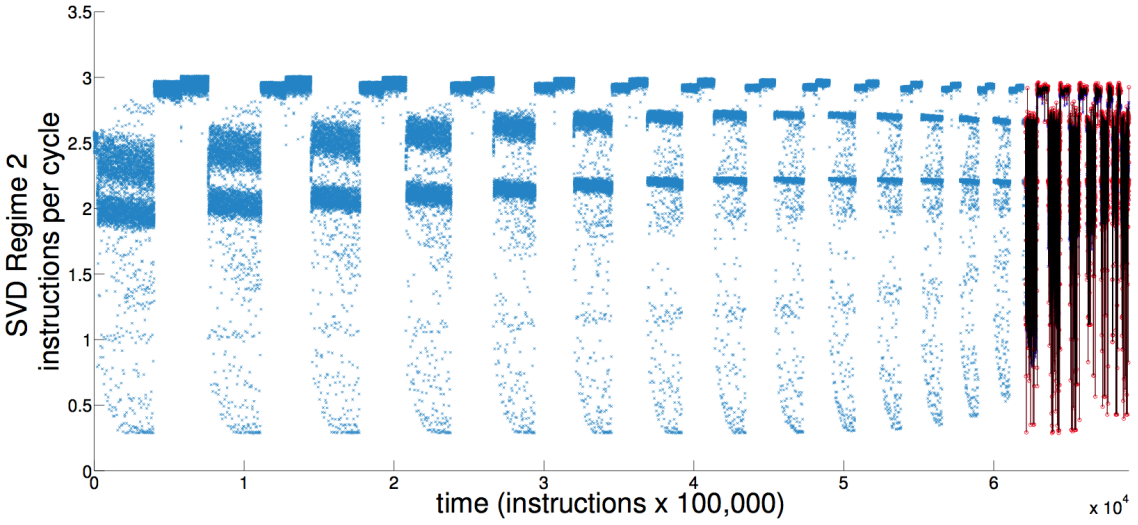


(b) SVD IPC with coloring

Figure 10: SVD Full Time Series

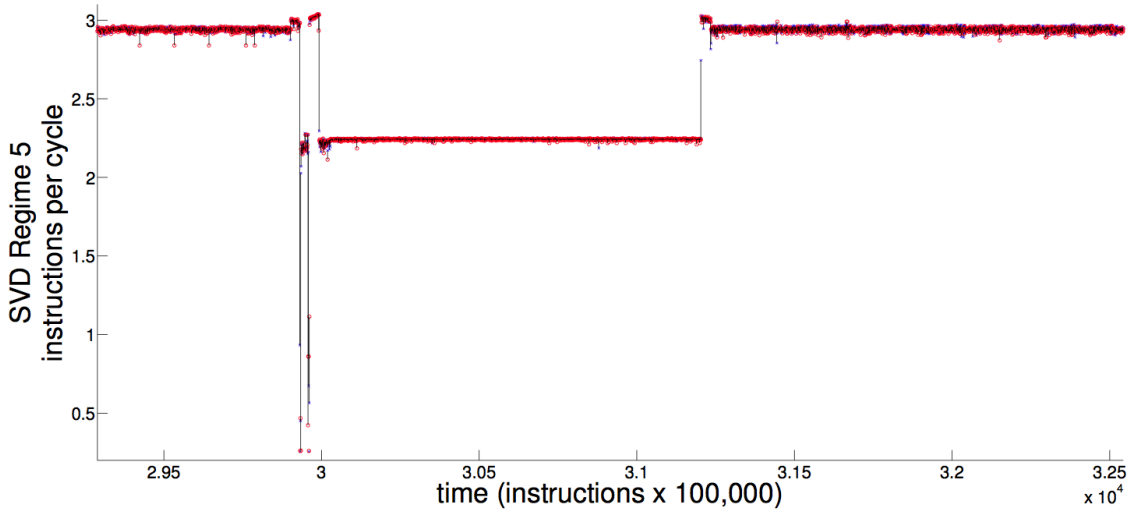


(a) SVD Regime 2 LMA Prediction

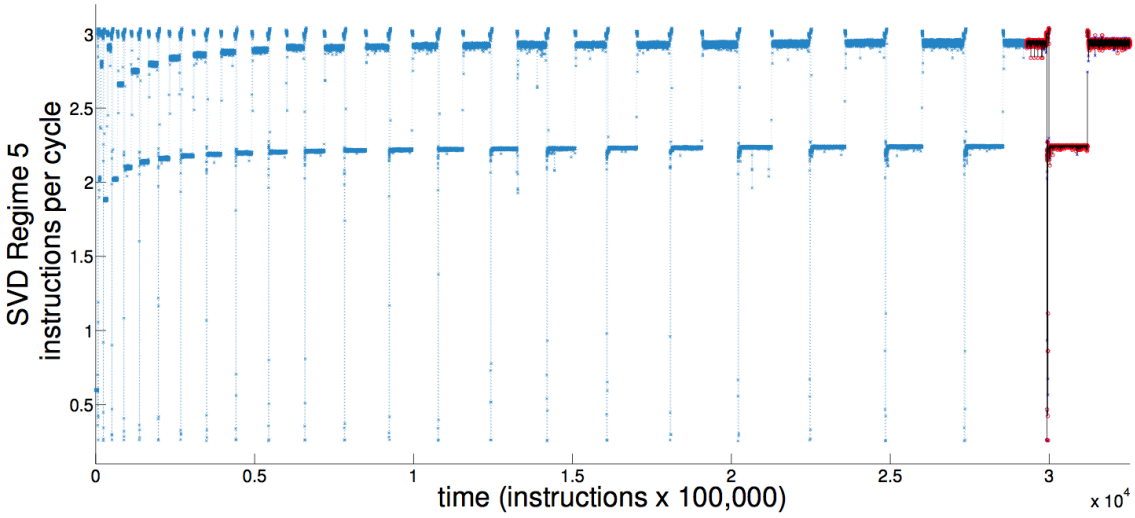


(b) SVD Regime 2 LMA Prediction with Full Time Series

Figure 11: SVD Regime 2 LMA Prediction Figures

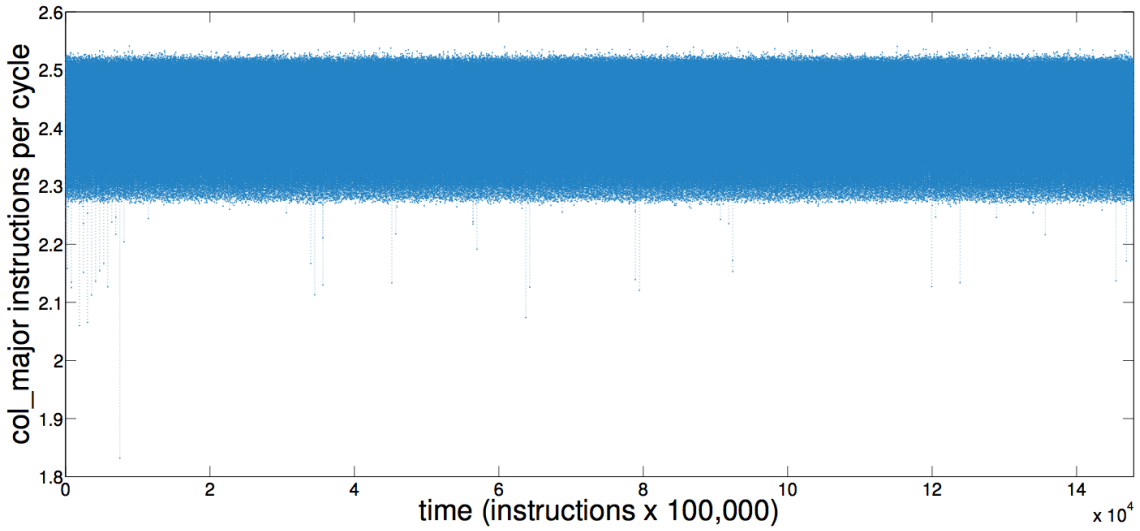


(a) SVD Regime 5 LMA Prediction

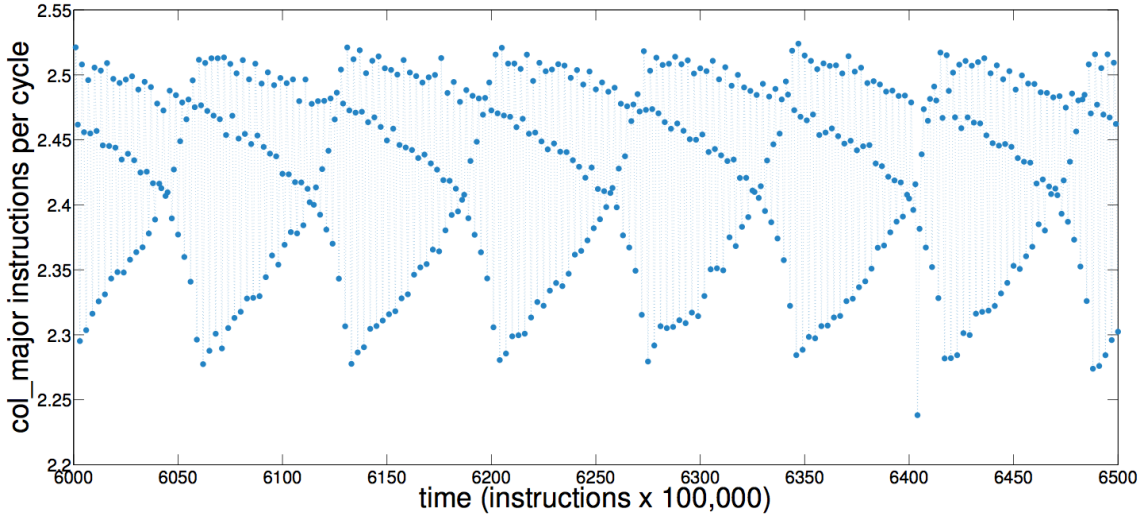


(b) SVD Regime 5 LMA Prediction with Full Time Series

Figure 12: SVD Regime 5 LMA Prediction Figures

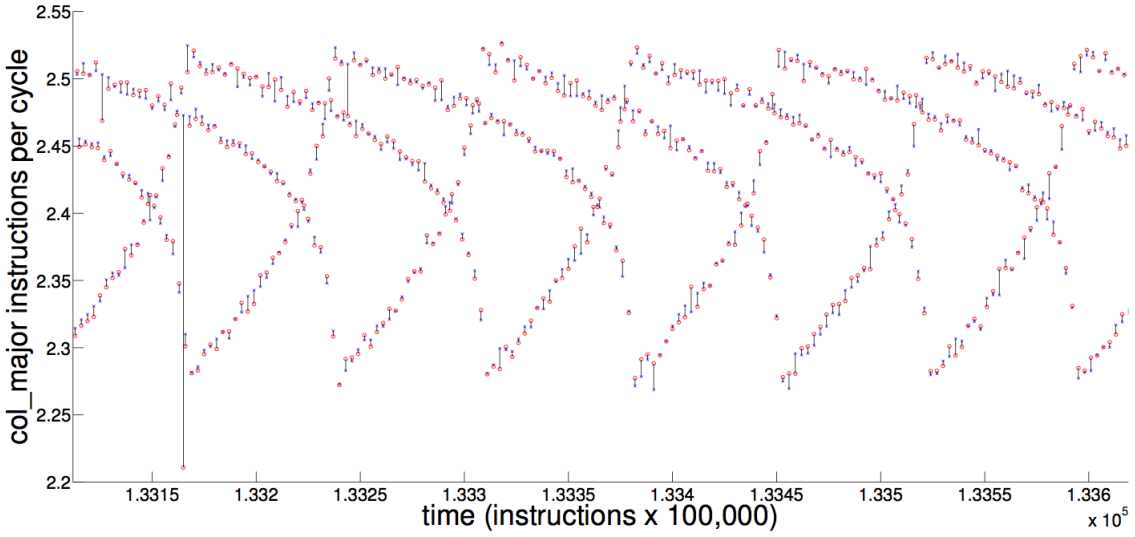


(a) col IPC Full Time Series

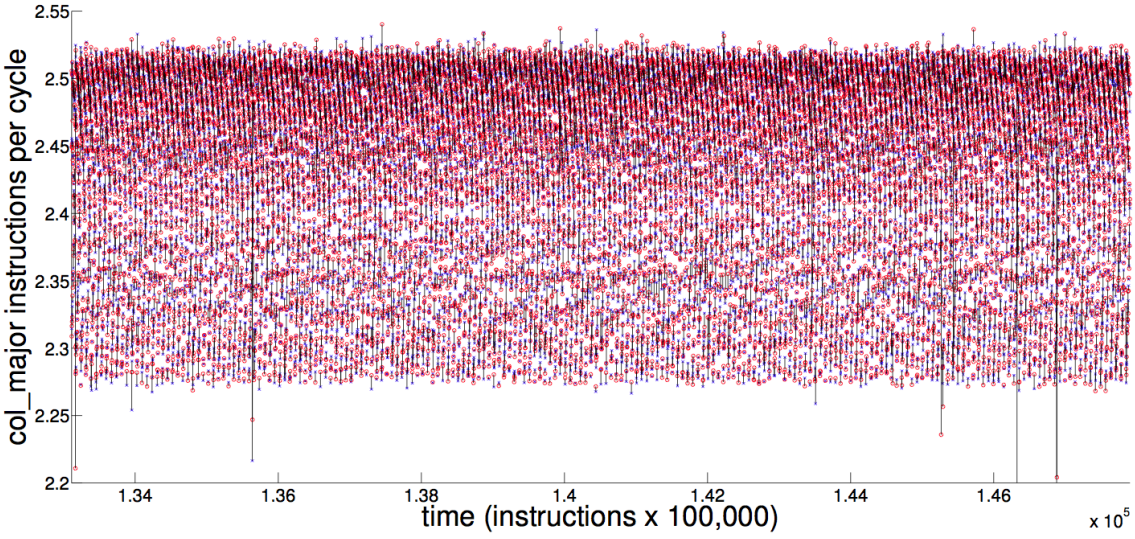


(b) col IPC Short Time Series for structure illustration

Figure 13: col Time Series



(a) col IPC Full Time Series



(b) col IPC Short Time Series for structure illustration

Figure 14: col_major Predictions

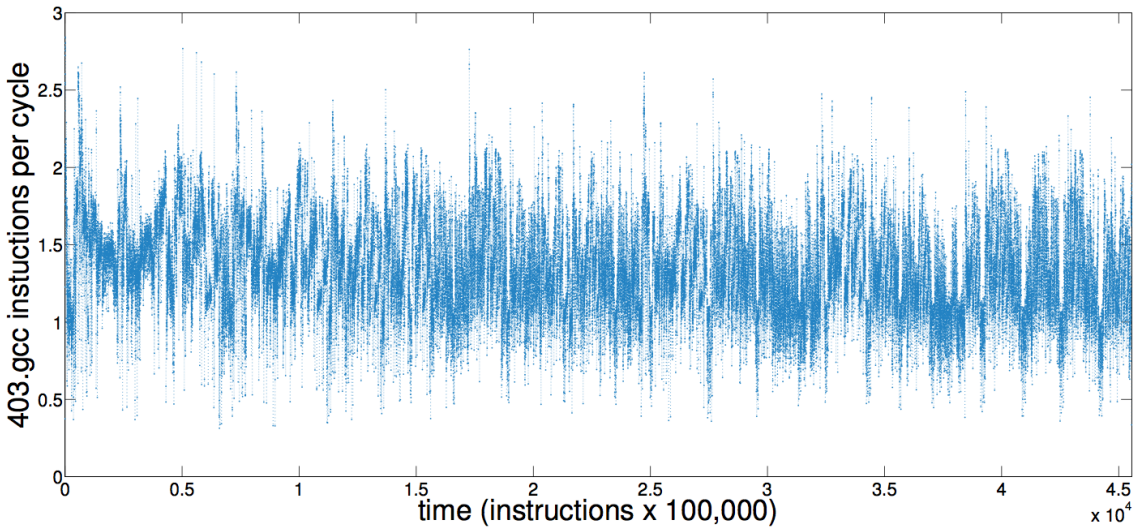


Figure 15: 403.gcc full time series

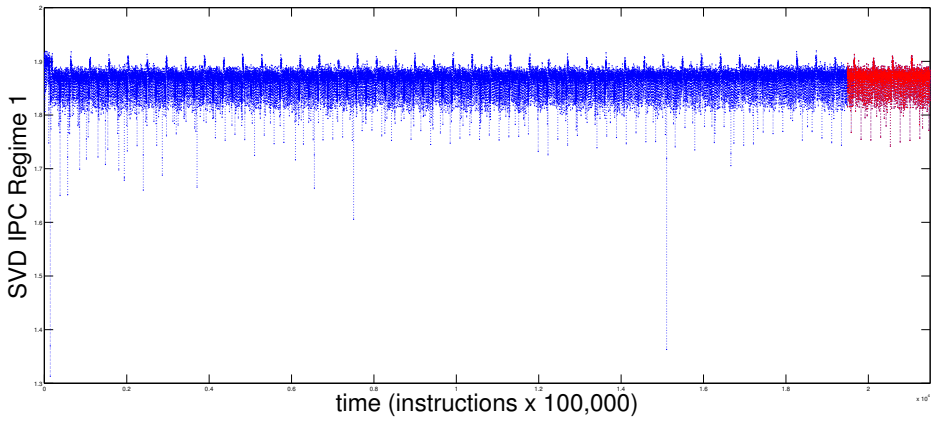


Figure 16: SVD IPC Regime 1 Time Series

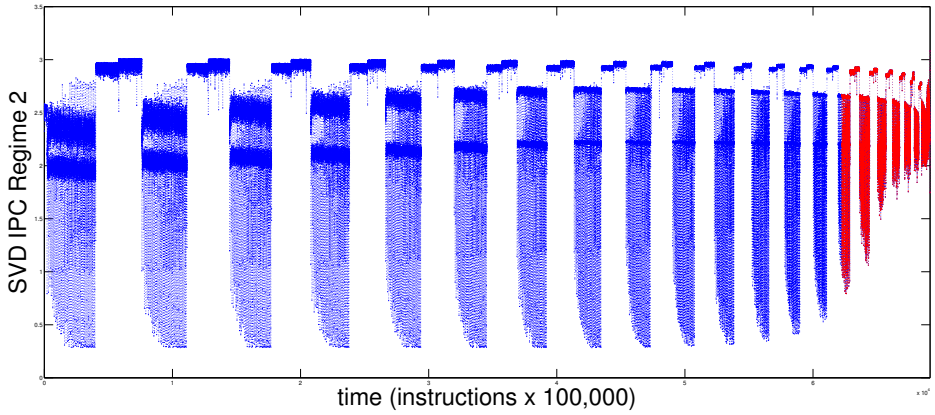


Figure 17: SVD IPC Regime 2 Time Series

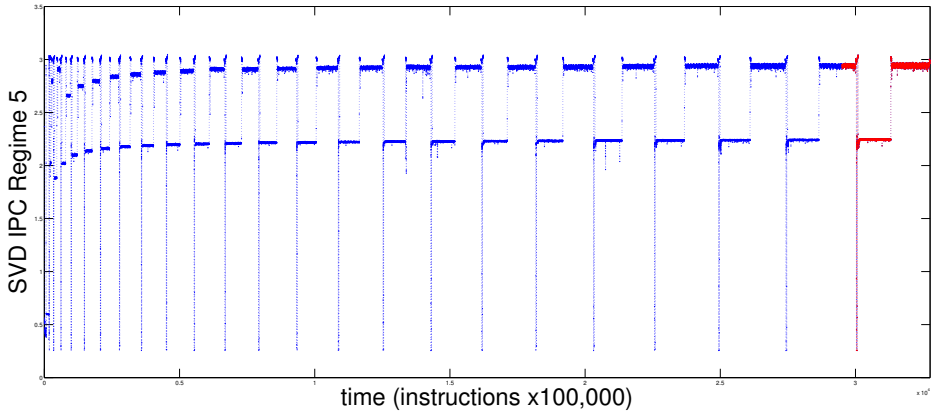


Figure 18: SVD IPC Regime 5 Time Series

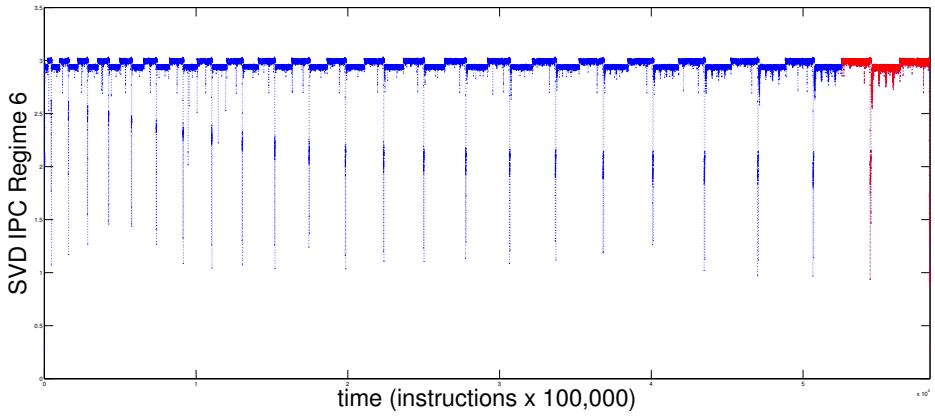


Figure 19: SVD IPC Regime 6 Time Series