Recurrence Plots in Analysis of Computer Systems

Amber Roche

Technical Report CU-CS 1067-10

University of Colorado
Department of Computer Science
Boulder CO 80309-0430 USA

July 2010

Recurrence Plots in Analysis of Computer Systems

Amber Roche

PART 1: Introduction and Explanation

Abstract

To analyze computer systems or compare them to one another, measurements are taken as the program runs. This is called a performance trace, and it can involve large amounts of time series data. Currently, statistical tools are used for analysis but this approach loses information by ignoring the ordered nature of the data.

We propose improving computer performance metrics by viewing the computer as a dynamical system and studying the time varying behavior. We proposed accomplishing this by applying recurrence plots (RPs) to the data [6]. This approach provides a graphical characterization of the system that can (in some cases) filter though noise, identify non-stationary patterns, and make periodic behavior immediately apparent [7].

I tested the hypotheses that RPs will allow us to easily compare systems as well as to identify specific points in the data set that warrant further investigation. We found that RPs are not an appropriate tool for this purpose. We believe this is partially due to the fact that the scale patterns in time series data from hardware traces is much smaller than the entire time series. This difference of 1-2 orders of magnitude makes the patterns difficult to see. In addition, RPs appear too sensitive to noise to be useful for computer applications.

Background

The sensitivity to small perturbations is a defining characteristic of chaotic systems and is well known in computer systems (ie. irreproducibility, bugs, etc). It is exhibited within the architectural implementation (hardware) as well as code (software) and makes analysis difficult and error prone. In previous work, Bradley, Diwan, and Mytkowicz have shown that a nonlinear dynamics model of computer systems captures the effects of both factors [8]. Current work is focused on understanding the effects.

This involves running and analyzing performance traces on hardware as well as simulators. This generates large amounts of time series data with significant noise. The conjecture we will explore here is that RPs will help with the data analysis by providing an additional tool for interpreting and comparing these complex systems [6].

Methods

A recurrence plot (RP) is a tool which allows recurrence patterns in time series data to be visualized in two dimensions [9]. This is done by plotting points where the trajectory at time i is close to the trajectory at time j. This can produce a colored graph corresponding to a range of differences (unthresholded), or a black and white graph (thresholded) where only the points outside a specified threshold are plotted. Mathematically, a thresholded RP corresponds

to $R_{i,j} = 1$ if:

$$|x_i - x_j| < \tau$$

else $R_{i,j} = 0$, where R is the matrix corresponding to the recurrence plot, x_t corresponds to displacement at time t, and τ is some threshold value.

An RP always contains the line $y = x$ about which it is symmetric (reflecting the fact that the system's state is equal to itself at every point in time). Lines parallel to this diagonal signify recurrences. This is useful in identifying periodicities, limit cycles, or chaotic behavior. Recurrence plots allow one to look beyond noise in the system as well as to identify non-stationary patterns and other interesting points [3, 1].

The data to be studied here consists of example data generated for the purpose of understanding the tools as well as instructions per cycle (IPC) data taken from a simulator. They both consist of some measurement as a function of time. We also analyzed cache misses (a memory usage metrics) but these results are not included in the report. Data was gathered by other members of the team (specifically Todd and Stephen) from simulators as well as real hardware systems. Measurements were taken every hundred thousand cycles and later normalized to "per cycle" count to produce IPC. When calibrating the RP software, I down sampled the data in order to achieve a manageable size.

The goal of this work was to evaluate RPs as a tool for identifying patterns and points of interest in hardware traces. The data from such traces is long and difficult to interpret. We explored whether RPs aided in the analysis of these data.

PART 2: Understand and Calibrate Tools

In order to evaluate the effectiveness of recurrence plots in identifying points of interest in large time series data, I first looked at data with known characteristics. This served the purpose of both developing my understanding of the tools as well as determining the strengths and weaknesses of RP analysis. I chose to look at the effects of two characteristics that are likely in our experimental data: noise and drift.

Noise Experiment:

Noise is present in virtually every experimental data set. In our hardware traces (from real systems), many steps have been taken to limit the amount of noise, but it is impossible to completely eliminate it. For example, we can ensure that there are no superfluous programs running when we are taking our measurements but we cannot turn off the operating system. Therefore, our data contains known noise from the operating system interrupting and using cycles of the CPU to perform tasks unrelated to our test programs. It is highly likely that there exists noise from a variety of unknown factors.

In an effort to quantify effects of noise on the analysis, I generated example data with varying amounts of noise. The data was generated from the following function:

$$x(t) = sin(t) + \frac{1}{2}sin(2t) + c * r$$

Noise was introduced in the last term where r is a random number from a gaussian distribution, and c is some scaling constant. Various levels of noise were introduced and are quantified below as a percentage of the range of the original time series.

The following analysis shows data with no noise (red) compared against data with 2% noise (green) and completely random data (100% noise, blue). Other levels of noise (1%, 5%, 10%) were also analyzed but are not shown due to space considerations.

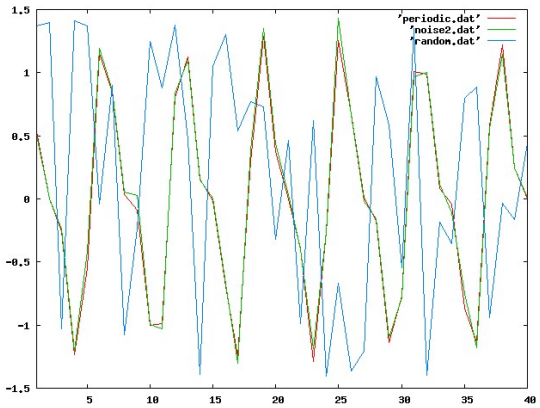


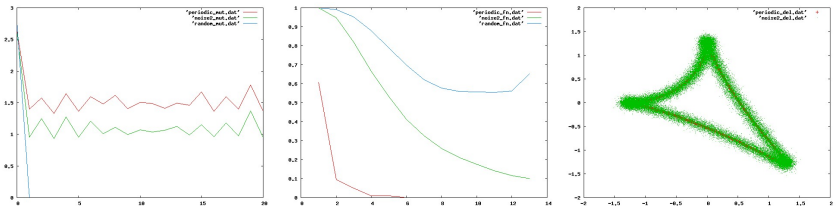
Figure 1: Time series of periodic data with noise

Figure 1 shows a snapshot of the time series for visualization purposes (the entire series contain 3,000 points)[10]. As you can see, the time series with 2% noise closely matches the series without noise while the random series does not (which is expected).

This data was analyzed using TISEAN, a package for analysis of time series data [4]; select results are shown below.

The first graph of Figure 2 shows mutual information functions for each level of noise. This is a way of measuring the degree of dependency between two variables. In the above case, it measures the dependency of two points in the time series, given they are x distance apart (where x is measured along the x-axis). We can see that the periodic signal retains a higher degree of mutual information as compared to the signal with 2% noise. The random signal retains no mutual information after the first step (which, again, is expected.)

Figure 2b shows TISEAN's false nearest neighbors function (FNN) is used to estimate the dimension of compressed data. We expect our traces to be many-dimensional but we are only measuring one dimension. FNN gives us a way of determining which points appear close in our trace data but are not actually in



(a) Mutual Information (b) False Nearest (c) Delay

Figure 2: TISEAN analysis

the system. We can see that the periodic signal is significantly lower than the signals with noise. We would ideally like the false nearest measure under 0.1.

Figure 2c is a visual representation of the reconstructed dynamics according to the delay coordinate embedding theorem [5]. This accurately represents the 2% noise signal as a "fuzzy" approximation of the periodic signal (clean red line, partially visible underneath.) The random signal was left off for clarity (it has no discernible pattern and covers the entire plot.)

Drift Experiment:

We would also like to be able to identify drift, a form of non-stationarity, in our data. Drift describes a phenomena where a recurring pattern is obscured by a constant force in some direction. For example, a swimmer's periodic motion can be obscured by the constant current in a river. I analyzed the ability of our analysis to detect drift by looking at data generated from the following function:

$$x(t) = sin(t) + \frac{1}{2}sin(2t) + c * t$$

where c is some small constant. Following is the beginning of a periodic time series (red) graphed with $c = 0.1$ (green). You can see that their pattern is similar but the latter is drifting upwards.

Non-stationarity is difficult to detect with common analysis tools such as a statistical mean, which ignores time varying behavior. But, it should be easy to identify with recurrence plots. For example, the above pattern that reoccurs but is affected by drift upwards (positive) reflects this with diagonal lines (recurrences) close to the center diagonal but fewer further out, as seen below.

PART 3: Simulator data (IPC)

The next step in evaluating recurrence plots as a tool for pattern detection in hardware traces is to analyze data from simulators. They are more complex and difficult to analyze than the above toy systems but are much simpler than real systems (i.e., hardware traces from real computers.)

Simulators are a common way to analyze computer systems because they allow researchers to look at various parameters such as cache size and configuration, processor speed, and much more without buying specific machine. This

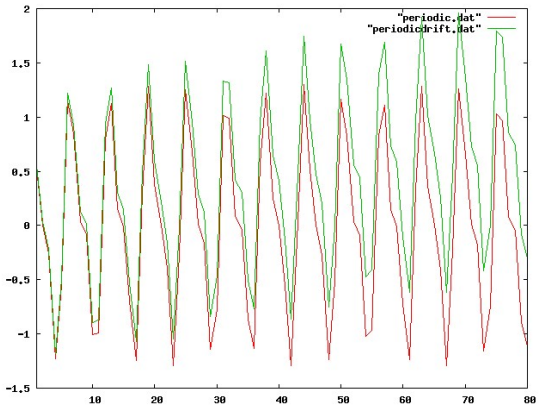


Figure 3: Time series of periodic data with drift

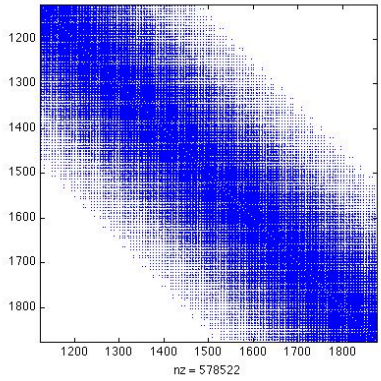


Figure 4: RP of periodic data with drift

method is also used in industry, allowing the manufacturer to do some analysis of a processor even before it is built. Simulators can take a large number of measurements as the virtual machine runs a program, and the entire environment can be controlled and measured.

The simplification of simulators is both a benefit and detriment. It is very useful to isolate variables in an experiment and, therefore, simulators can be very helpful in determining the effect of each parameter. But, often, this can be very misleading because real systems do not operate in isolated, controlled environments. In fact, it is often the *interaction* of different parts (cache size, processor speed, operating system, specific program implementation, etc) that dominates the dynamics. This can result in simulator traces that are vastly

different than their corresponding hardware traces.

While we recognize the limited connection between simulator results and hardware results, we chose to test RPs as a tool on various simulator runs. For the duration of the experiment, the simulator was configured as follows with approximately 2.4 Ghz processor and 4 MB L2 cache.

I began by running the following two programs on the simulator:

- repeated row major matrix initialization
- repeated column major matrix initialization.

The programs are very simple and are given below:

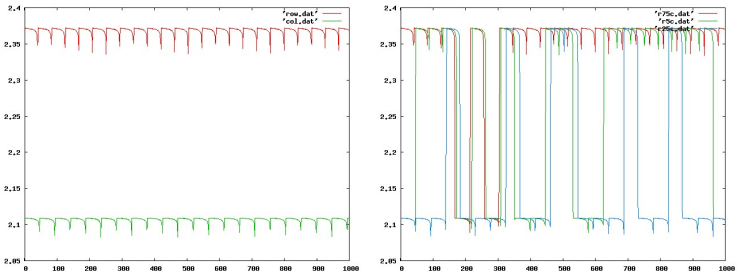
```
void rowmajor () {
  int i, j, n;
  for (n = 0; n < N; n++)
    for (i = 0; i < 1024; i++)
      for (j = i; j < 1024; j++)
        data[i][j] = 0;
}


void colmajor () {
  int i, j, n;
  for (n = 0; n < N; n++)
    for (i = 0; i < 1024; i++)
      for (j = i; j < 1024; j++)
        data[j][i] = 0;
}
```

where $N = 1000$ and the matrix, $data$, is of size 1024×1024. The simulator interrupted the program every 100,000 cycles to retrieve data. One must be careful when determining how often this interrupt happens because if it is too frequent, there will be a large amount of noise introduced. But, if it is not frequent enough, there will be insufficient data to draw conclusions. We chose 100,000 because, from past experience, we believe this minimized both issues [8].

Because of the way matrix data is stored in memory and the way the processor moves data in and out of the caches, the row major initialization is a fast, efficient use of memory and processors, while the column major initialization is slow and inefficient.

Next, we ran three more programs that performed a mixture of the above two programs. They do various percentages of row and column matrix initialization; equal parts of each, 25% row and 75% column, and visa versa. This was done by picking a random number from a uniform distribution after each pass through the matrix to determine which function call, $rowmajor()$ or $colmajor()$, would execute next.

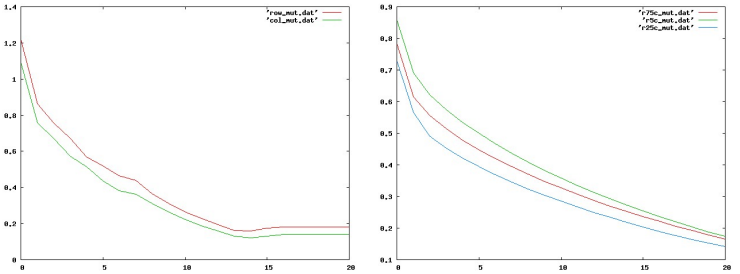
The following are snapshots of the time series of the above five programs. As you can see, row major matrix initialization has consistently higher IPC and, therefore, is a more efficient use of the computers resources (both memory and processors).



(a) Row, Column Data Separately (b) Mixed Row and Column Data

Figure 5: Time series of IPC data

Figure 6 shows TISEAN's mutual information function. We can see that both the row and column major programs drop off quickly while the mixed programs drop off more slowly. This results because of the random component of the mixed programs, making them more "fuzzy."

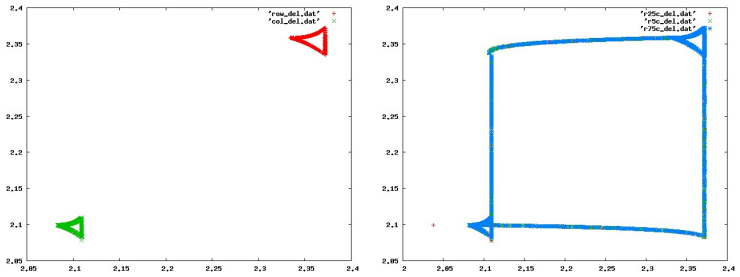


(a) Row, Column Data Separately (b) Mixed Row and Column Data

Figure 6: Mutual Information Function of IPC data

TISEAN's delay function (Figure 7) is one of the best visualizations of the characteristics of the five programs. Figure 7a shows the row and column major programs as both very distinct and very contained. Figure 7b shows all three mixed programs (one on top of the other), which contain both row and column as well as the phase change between the two.

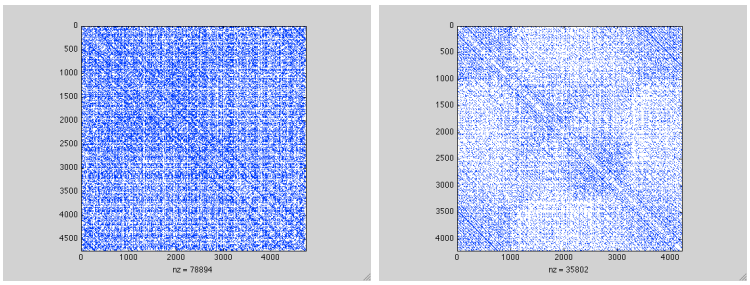
This trait is more obscured in the recurrence plots of these time series (Figures 8 and 9). One can see some degree of periodicity in the row major RP the the others are obscured with noise. It is likely the case that the periodic frequencies of the simulator traces are too low (compared to the length of the



(a) Row, Column Data Separately (b) Mixed Row and Column Data

Figure 7: Delay Function of IPC data

trace) to be easily apparent. Also, in the mixed programs, we hypothesize that periodicity is obscured in the RP's because the dynamics are being dominated by the random element (switching between $rowmajor()$ and $colmajor()$).



(a) Column Major (b) Row Major

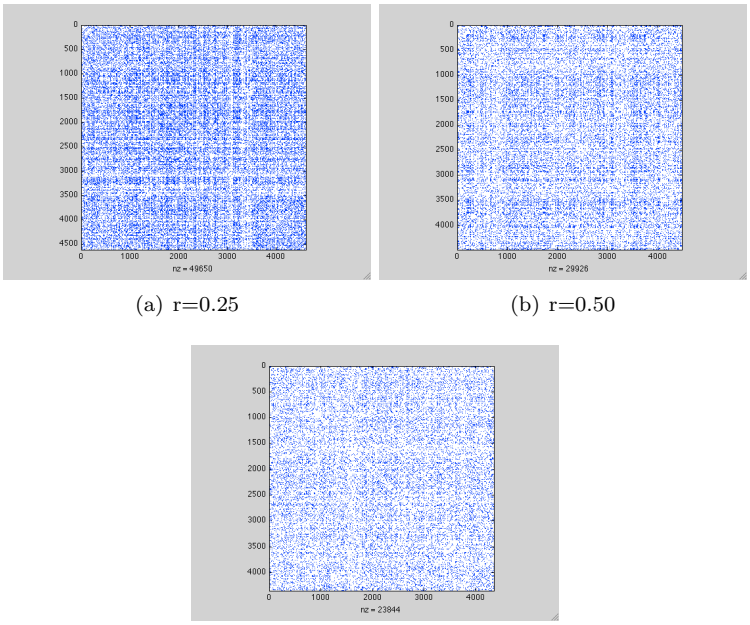
Figure 8: RP's of IPC data, separately

In order to further analyze the above RPs, we use Recurrence Quantification Analysis (RQA). This is a collection of techniques for mathematically quantifying RPs [2]. It includes various measures, including Percent Determinism. This measures the frequency with which a given state reoccurs and is given by:

$$PercentDeterminism = \frac{\sum_{l=lmin}^{N} l \times P(l)}{\sum_{i,j=1}^{N} R(i,j)}$$

where $P(l)$ represents the frequency distribution of l, the lengths of the diagonal lines and $R(i,j)$ is the measure of how close the trajectories are at time i and time j. The following Percent Determinism measures for the above traces give us an idea of the predictability of each system.

Table 1 contains the RQA measures for this data. The % Determinism measures do not seem to reflect the degree of periodicity in the RPs. We believe



(a) r=0.25 (b) r=0.50

(c) r=0.75

Figure 9: RP's of IPC data, mixed

Table 1: Percent Determinism Measures

Program Type	% Determinism	DET/RR
column major	25.6	9
25% row major	26.1	15
50% row major	23.6	19
75% row major	20.7	17
row major	20.8	59

that, for the mixed programs, the dynamics are being dominated by the random element introduced when choosing between *rowmajor*() and *colmajor*(). The second measure (DET/RR) seems to reflect the fact that *rowmajor*() is more efficient. We believe this is because it does each iteration faster (more efficiently), but still has the same overall number of iterations (recurrences).

PART 4: Conclusions

I worked on this project with professors Elizabeth Bradley and Amer Diwan as well as graduate students Todd Mytkowicz (Systems PhD), Zach Alexander (Applied Math PhD), and Stephen Heck (Computer Science Masters). One of the main research interest of the group is an ongoing project titled "Validating Architectural Simulators Using Non-Linear Dynamics Techniques." The key

goals of this project are to determine how accurately simulators model corresponding computer hardware and to identify parts of the simulator responsible for any difference between it and the real hardware system.

We believe that RPs are not an appropriate tool for the analysis of these data. We found that RPs, when applied to the simplified data in this report, produce ambiguous results and were not helpful for analysis. First, RPs were unsuccessful in identifying patterns in the highly periodic simulator data. We believe this is due to issues with the scale of the patterns (periods) as compared to the scale of the data. Also, real hardware data has substantially more noise than any of the time series we looked at in this report. Any useful analysis tool would need to work in these conditions.

References

[1] E. Bradley and R. Mantilla. Recurrence plots and unstable periodic orbits. *Chaos*, 12:596–600, 2002.

[2] Wikipedia contributors. Recurrence quantification analysis, May 2009. `http://www.en.wikipedia.org`.

[3] J.-P. Eckmann, S. Kamphorst, and D. Ruelle. Recurrence plots of dynamical systems. *Europhysics Letters*, 4:973–977, 1987.

[4] R. Hegger, H. Kantz, and T. Schreiber. Practical implementation of nonlinear time series methods: The tisean package. *Chaos*, 9:413, 1999.

[5] J. Iwanski and E. Bradley. Recurrence plots of experimental data: To embed or not to embed? *Chaos*, 8:861–871, 1998.

[6] N. Marwan, M. C. Romano, M.Thiel, and J. Kurths. Recurrence plots for the analysis of complex systems. *Physics Reports*, 435(5-6):237–329, 2007.

[7] N. Marwan, M. C. Romano, and M. Thiel. Recurrence plots and cross recurrence plots, 2009. `http://www.recurrence-plot.tk`.

[8] T. Mytkowicz, A. Diwan, and E. Bradley. Computer systems are dymanical systems. *Chaos*, in press.

[9] S. Strogatz. *Nonlinear Dynamics and Chaos*. Addison-Wesley, Reading, MA, 1994.

[10] T. Williams and C. Kelley. Gnuplot, March 2009. `http://www.gnuplot.info`.