Detecting Phases in Cache-miss Traces with Wavelets

Stephen L. Heck

Technical Report CU-CS 1071-10

University of Colorado at Boulder
Department of Computer Science

Spring 2010

Abstract

Beyond being a curiosity in describing programs, phase research has been incorporated into compiler optimization, CPU power usage, memory management and program performance simulation [1, 2]. Recently wavelets have garnered interest as a tool for phase detection and analysis. This paper introduces the wavelet perspective of time-frequency analysis and presents an algorithm that incorporates the Discrete Wavelet Transform for detecting phases in L1 cache-miss performance traces.

1 Introduction

The goals of this paper are:

1. Provide a general introduction for the wavelet approach to time-frequency analysis.
2. Present a simple algorithm for phase detection that merits further investigation.
3. Discuss future directions in applying wavelets to computer performance traces.

Although the theoretical side of computation has a solid foundation and is a good starting point to analyze the performance of an algorithm, the analysis of real programs running on real machines is in many ways a black art. Due to advancements in computer architecture such as instruction level parallelism, cache memory design and multi-core processors, at a certain point the theoretical description of an algorithm does not offer any further insight to the program's lower-level run-time behavior [3].

Another issue in systems work is that the principles of a good experimental setup such as limited perturbation of the item being measured and repeatable starting states are not possible when monitoring computer programs. To measure a computer program, we employ another program on the same machine that monitors the machine's performance. The performance monitor uses the same system resources as the program being monitored and thus actively changes the state of the computer during the experiment. Also, two different instances of the same experiment will not have the same initial system state. Thus, each experiment becomes a unique interaction of the program and the state of the system. Recent research that discusses these issues can be found in [4].

Although these issues are present, a computer program does exhibit common behavior across multiple experiments on the same machine. If you are careful not to perturb the system too much, the behavior of the performance data can be considered an accurate reflection of the program.

With these issues in mind, phase detection and analysis of performance data is a tool that picks-up where the theoretical description leaves off. Beyond being a curiosity in describing programs, phase research has been incorporated into compiler optimization, CPU power usage, memory management and program performance simulation [1, 2]. Thus the analysis of performance time-series of a program, such as the L1 cache-miss behavior, becomes an essential tool to analyzing real programs.

2 Motivation

Wavelet theory is a powerful time-frequency analysis technique that has been applied to a wide range of signal processing problems including image processing and the analysis of seismic signals [5, 6]. Wavelets are natural choice for analyzing time-series data because of their ability to describe a signal whose frequency content changes with time. Because computer programs have different regions of code that consume system resources in different ways and rates, the effect of the program on a system is not constant throughout its execution. A performance trace such as the L1 cache-miss time-series of a program will reflect this fact with cache-miss frequencies that change in time. Thus wavelets are very applicable in examining computer performance time-series. This paper presents an algorithm that incorporates the Discrete Wavelet Transform to detect phases in L1 cache-miss data.

Various non-wavelet methods for phase detection and analysis have been examined in [7, 8, 9, 10, 11]. Recently though, wavelets have garnered the interest of researchers. Huffmire and Sherwood used a 2-dimensional Haar wavelet transform to predict program memory access [1]. Cho and Li used fixed width portions of cache-memory address accesses and analyzed these "phases" with a complexity metric based on a multi-resolution wavelet analysis [12]. Casas, Badia and Labarta used wavelets to detect phases in MPI applications to identify program regions that aren't scaling well [13]. Shen, Zhong and Ding used wavelets to predict *locality phases* based on the notion of how many data access are performed before reusing a memory location [14].

This paper takes a more humble approach in its application of wavelets. First, the time-series performance trace is not as complicated as in the previous papers, but simply L1 cache-miss counts sampled every 100,000 cycles. This is attractive because the instrumentation to capture this trace is very minimal. Second, this paper develops a simple threshold algorithm to detect phases based on the persistence of peaks that appear in the Discrete Wavelet Transform. The effectiveness of this approach to phase detection is tested by three simple loop kernel programs and two real-world programs (bzip2 and gzip).

3 Signal Processing

This section presents the necessary background to understand the aspects of signal processing leveraged by the phase detection algorithm in this paper. The reader can skip this section if they are comfortable with the Fourier Transform and wavelets. The presentation of the phase detection algorithm in Section 4 will identify the wavelet details that are especially pertinent and references to this section will be made clear.

3.1 The Fourier Transform

Before discussing the properties of wavelets, it will be useful to introduce frequency analysis with the Fourier Transform. Conceptually signal processing is simply about finding appropriate ways to measure aspects of a signal. The standard measuring tool for signal processing has its conceptual roots in the vector dot product:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=0}^{n} a_i b_i \tag{1}$$

The dot product between two vectors **a** and **b** will measure the similarity of vector **a** in terms of **b** or vice-versa. When we replace vectors **a** and **b** by functions defined over a continuous variable such as time t the discrete sum in equation (1) is replaced with an integral:

$$\langle f(t), g(t) \rangle = \int_{-\infty}^{+\infty} f(t)g(t)\, dt \tag{2}$$

Signal processing makes heavy use of this integral dot product. Letting $f(t)$ be the signal whose frequency content we are interested in analyzing and setting $g(t)$ equal to a function that we know oscillates at a specific frequency w will let us describe $f(t)$'s oscillatory behavior in terms of $g(t)$'s oscillatory behavior. Again this is the same concept as using a vector dot product to describe vector **a** in terms of vector **b**.

The Fourier Transform is then defined by setting $g(t)$ equal to the complex exponential $e^{-j2\pi\omega t}$ [15]:

$$\hat{f}(\omega) = \langle f(t), e^{-j2\pi\omega t} \rangle = \int_{-\infty}^{+\infty} f(t)e^{-j2\pi\omega t}\, dt \tag{3}$$

where the complex exponential $e^{-j2\pi\omega t} = \cos(2\pi\omega t) - j\sin(2\pi\omega t)$ and $j = \sqrt{-1}$.

Two reasons for describing $f(t)$ in terms of the complex exponential is that 1) Sines and Cosines are simple and well understood oscillatory functions. 2) Given the Fourier Transform of a function the original function can be reconstructed with the Inverse Fourier Transform [15]:

$$f(t) = \frac{1}{2\pi} \int_{-\infty}^{+\infty} \hat{f}(\omega)e^{j2\pi\omega t}\, d\omega \tag{4}$$

Before ending this brief introduction to frequency analysis, the following plot shows the Fourier Transform of the function $f(t) = \cos(2\pi t) + .5\sin(2\pi 2t)$. The cosine term is oscillating at $\omega = 1Hz$ and the sine term is oscillating at $\omega = 2Hz$. The first plot is the time-series of this function and the second plot is a graphical representation of the Fourier Transform called the *Periodogram*. The *Periodogram* plot has frequency ω for the horizontal axis and the vertical

axis is the complex norm of equation (3). The Fourier Transform/Periodogram plot clearly shows the frequencies that are present in the signal (the spikes at $\omega = 1$ and $\omega = 2$), but it does not give any indication of what *times* these frequencies occur. This is the fundamental difference between the Fourier Transform and the Wavelet Transform.

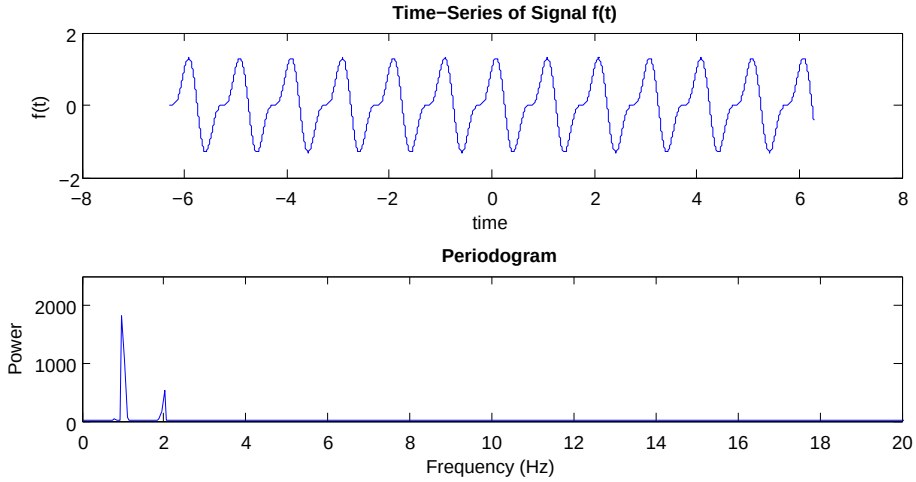

Figure 1: The time-series and Fourier transform of $f(t) = \cos(2\pi t) + .5\sin(2\pi 2t)$. The plot clearly shows the frequencies $\omega = 1Hz$ for the cosine term and $\omega = 2Hz$ for the sine term.

3.2 The Wavelet Function

In the next few subsections, the wavelet function and the properties needed to understand the technique that this paper uses to detect phases will be presented. The wavelet transform will be mentioned, but its equation will be introduced following the discussion in this section. It is enough for now to know that the wavelet transform is another integral measurement method similar to that of the Fourier Transform. What sets it apart from the Fourier Transform is its ability to handle functions/signals whose frequency content changes with time[1].

Wavelet theory is an incredibly deep subject and thus this section is merely scratching the surface. For a good introduction to wavelets and time-frequency methods see [16, 17] and for deeper results see [15].

3.2.1 What is a Wavelet?

Instead of using a complex exponential as the basis to compare a signal against, the wavelet transform replaces $g(t)$ in equation (2) with the wavelet function denoted as $\psi(t)$. To make this more concrete, the following figure taken from [18] is an example of what a wavelet looks like. The x-axis is time and the y-axis is the wavelet function's value. Notice how the wavelet is non-zero for only a limited time interval. This property, called localization in time, is common to all wavelet functions. Time localization is one reason why wavelets can handle signals whose frequency varies. Contrast this property with the Fourier Transform that uses sines and cosines as the comparison functions. Sines and cosines oscillate indefinitely over time and thus are not localized to a specific time interval.

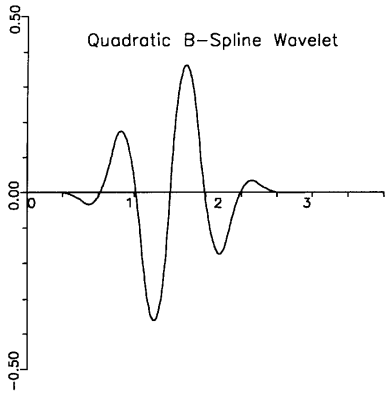

Figure 2: The quadratic spline wavelet function. Picture taken from [18].

[1]Other time-frequency tools exist such as the Short-time Fourier Transform and the Wigner-Ville distribution[16]. The current work will only discuss the Wavelet Transform.

3.2.2 The Wavelet Admissibility Condition

Not every function can be called a wavelet. Wavelet functions are defined so that the wavelet transform can be inverted and the original signal can be recovered (recall the Inverse Fourier Transform from Section 3.1).

Let $\hat{\psi}(w)$ be the Fourier Transform of the function $\psi(t)$. If

$$C_\psi = \int_0^{+\infty} \frac{\hat{\psi}_{u,s}(w)}{w}\, dw < \infty \tag{5}$$

This condition, called the admissibility condition, guarantees that a signal $f(t)$ can be reconstructed from its wavelet transform. For details of the proof, see theorem 4.4 of [15].

A property that will be exploited by the phase detection algorithm and is necessary for the admissibility condition to hold is: A wavelet has a zero time-average [15], i.e.

$$\int_{-\infty}^{+\infty} \psi(t)\, dt = 0 \tag{6}$$

This property will be identified during the discussion of the phase detection algorithm.

3.2.3 The Mother Wavelet

In the wavelet transform we actually use several versions of the wavelet function $\psi(t)$ to analyze a signal. These "wavelet measurement sticks" are called "wavelet-atoms" and are simply dilated[2] and time-shifted versions of $\psi(t)$. The following equation shows shows how $\psi(t)$ is dilated and time-shifted to produce the wavelet-atoms $\psi_{s,u}(t)$ [15].

$$\psi_{s,u}(t) = \frac{1}{\sqrt{s}} \psi\left(\frac{t-u}{s}\right) \tag{7}$$

The wavelet-atom consists of three parameters where t is time, u shifts the wavelet along the time axis and s is the time-scale/dilation parameter. Time-shifting the wavelet gives us the ability to measure a signal at a specific time intervals. Dilating time with s either shrinks or expands the wavelet's width allowing us to zoom in on the finer details of the signal or zoom out to get an overall view. The factor $\frac{1}{\sqrt{s}}$ ensures that total energy of the wavelet-atom equals the total energy of the mother wavelet. Section 3.2.4 takes a deeper look into the ramifications of changing the parameters s and u. As a final note, because $\psi(t)$ "produces" the wavelet-atoms, it is affectionately called the "mother wavelet."

3.2.4 The Heisenberg Uncertainty Principle

Although the Heisenberg Uncertainty Principle is most often associated with Quantum Mechanics, it also applies to any time-frequency analysis technique [15]. Understanding this property is critical to understanding time-frequency analysis. This principle shows that there is a limit to how much we can know simultaneously about the frequency and time behavior of a signal.

[2]Mathematicians define dilation to mean both contraction and expansion.

The equation describing this relationship is constructed in terms of the time-variance of a signal σ_t and the frequency-variance of a signal σ_ω.

Lets take a look at what is meant by time-variance σ_t of a signal. When you pinpoint an exact time for which to evaluate a signal, you know the exact value of the signal. In this case the variance in your knowledge of the signal's value with respect to time is zero, i.e. $\sigma_t = 0$. This is precisely the case for a time-series representation of the signal. The exact values of the signal in each moment in time are known but the consequence is that you have no idea what frequencies are present in the signal.

On the other hand, applying the Fourier Transform to a signal will reveal the frequencies that are present. Hence you know how the function behaves with respect to frequency, i.e $\sigma_\omega = 0$, but now you have no knowledge of the signal in time.

Recall how figure 1 showed the time-series representation and Fourier Transform of a signal. In this light, we can view the time-series and frequency representations as the two extremes about our knowledge of a signal. The Heisenberg Uncertainty Principle shows the trade-off between time and frequency knowledge when we are somewhere between these two extremes [15]:

$$\sigma_t \sigma_\omega \geq \frac{1}{2}$$

In this equation notice how the greater-than sign forces one variance to become larger when the other becomes smaller.

The following figures are presented to further elucidate this concept and how it manifests itself in the context of wavelets. The first picture shows a how a wavelet's "Heisenberg box" defined by $\sigma_t \sigma_w$, changes in relation to the wavelet's width. The horizontal axis represents time and the vertical access is frequency. The waveforms along the time axis show the wavelet and how its width affects σ_t. The wider the wavelet, the larger the time variance. The waveforms along the vertical axis represent is the magnitude of the Fourier Transform of the wavelet. Notice the inverse relationship between the wavelet width and frequency variance.

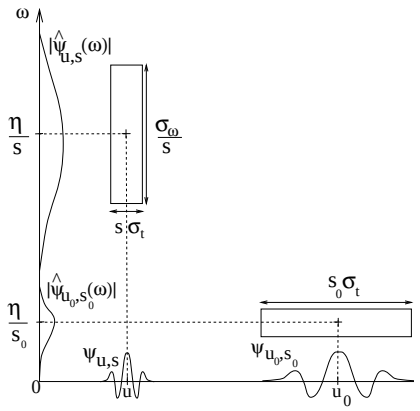

Figure 3: The effect of the Heisenberg Uncertainty principle. These Heisenberg boxes show how the wavelet's time and frequency variance responds with respect to the wavelet width. Picture taken from [15].

The next picture shows how the Heisenberg boxes tile the time-frequency plane as the scale increases. At small scales, the resolution of high frequencies is sacrificed to gain better information on where these frequencies occur in time. At large scales, time resolution is sacrificed to gain better frequency resolution in the lower frequencies.

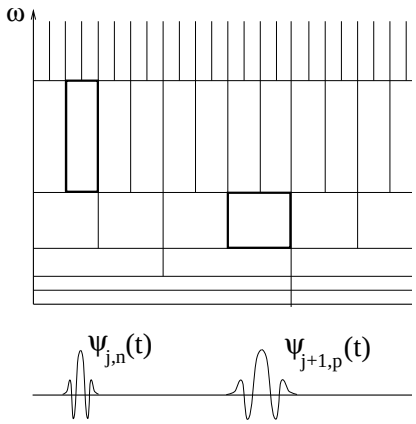

Figure 4: The Heisenberg box tiling of the time-frequency plane for wavelets. Picture taken from [15].

3.3 The Wavelet Transform

This section introduces the equations for the Continuous Wavelet Transform and the Discrete Wavelet Transform. Like the Fourier Transform the concept of these equations are also rooted in the vector dot product discussed in Section 3.1.

3.3.1 The Continuous Wavelet Transform

Given a mother wavelet, the Continuous Wavelet Transform (CWT) of a signal $f(t)$ is defined by

$$d_{s,u} = \int_{-\infty}^{+\infty} f(t)\psi_{s,u}^*(t)\,dt \tag{8}$$

where the wavelet coefficient $d_{s,u}$ is the projection of the function $f(t)$ onto the complex conjugate[3] of the wavelet-atom $\psi_{s,u}(t)$ [15]. This equation is known as the analysis equation.

Because of the admissibility condition given by equation (5), a signal can be reconstructed from its CWT. In the following equation C_ψ is the wavelet admissibility constant defined by equation (5) and it plays the role that 2π plays in the Inverse Fourier Transfrom from equation (4) [15]:

$$f(t) = \frac{1}{C_\psi}\int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} d_{s,u}\psi_{s,u}(t)\,ds\,du \tag{9}$$

This equation is known as the synthesis equation. For a proof of this result see [15].

3.3.2 Discrete Wavelet Transform

The Discrete Wavelet Transform (DWT) is a modification of the Continuous Wavelet Transform given in equation (8), where the wavelet is scaled and shifted by powers of two. The "discrete" aspect of the DWT reflects the fact that the wavelet is not shifted and scaled continuously.

The DWT also introduces a new concept called the "scaling function" denoted as $\phi(t)$. Conceptually the scaling function is employed as a course description of the signal, while the wavelet function represents the fine detail of a signal. A deeper look at why the scaling function is necessary won't be covered here, but can be found in [16]. For now, it is enough to think of the scaling function as the "overall" perspective on a signal and the wavelet function as the detailed perspective. These concepts will become clearer with the introduction of equation (12) and figure 5.

The equations defining the scaling coefficients and wavelet coefficients are [16]:

$$c_{k,n} = \int_{-\infty}^{+\infty} f(t)\frac{1}{\sqrt{2^k}}\phi\left(\frac{t-n}{2^k}\right)\,dt \tag{10}$$

[3]The conjugate of a complex number $z = x + j * y$ is $z^* = x - j * y$. When z is a Real number, i.e. $y = 0$, then $z = z^*$. This paper uses a real Wavelet function and hence $\psi(t)^* = \psi(t)$.

$$d_{k,n} = \int_{-\infty}^{+\infty} f(t) \frac{1}{\sqrt{2^k}} \psi \left(\frac{t-n}{2^k} \right) \, dt \tag{11}$$

Take special notice in the introduction of parameters 2^k and n that take the place of parameters s and u from the CWT. This is done to emphasize the fact that parameters k and n are *integer* numbers, whereas in the definition of the CWT s and u are *real* valued numbers.

To reconstruct the signal, the synthesis equation becomes a combination of the course level information given by the scaling coefficients and the detail information given by the wavelet coefficients [16]:

$$f(t) = \sum_n c_{m,n} \phi \left(\frac{t-n}{2^m} \right) + \sum_{k=m_0}^{m} \sum_n d_{k,n} \psi_{k,n}(t) \tag{12}$$

The first summation term is the course level approximation to the signal where $c_{m_0,n}$ and $\phi_{m_0,n}(t)$ are the scaling coefficients and scaling function respectively. The inner sum of the term is the detail of the signal encoded by the wavelet coefficients $d_{k,n}$ and wavelet-atoms $\psi_{k,n}(t)$ at the scale 2^k.

Notice that the double summation term of the wavelet-atoms and coefficients is essentially the discrete version of the continuous synthesis function given in equation (9) with one key difference: The number of scales in the discrete equation ranges from m_0 to m as opposed to $-\infty$ to ∞ for the continuous equation. This perspective in the context of equation (12) indicates that when we employ only a finite set of wavelet-atom scales, the scaling function is necessary to add the information that is lost from the wavelet scales that aren't being used.

3.3.3 A Multi-resolution Analysis Example

To reinforce the concepts of the scaling and wavelet functions the following figure shows an example of a multi-resolution analysis of a program trace using the quadratic b-spline wavelet[4] from figure 2. The plot labeled *Scaling Details* is the result of the first summation term of equation (12). Each plot labeled *Wavelet Details* at scales 2^3, 2^2 and 2^1 is computed by the inner sum of the second summation term of equation (12).

A few properties about this figure should be noted. First, because wavelets have a zero time average as a consequence of the admissibility condition, the *Wavelet Detail* plots always oscillate around zero. Second, as the scale increases peaks become more defined and indicate a border between different signal regions. This is not by accident. It's related to the way the wavelet's Heisenberg Boxes tile the time-frequency plane. The zero-average and peak-emergence properties will be exploited for phase detection.

[4]Section 4 will briefly discuss wavelet selection and why this wavelet is being used.

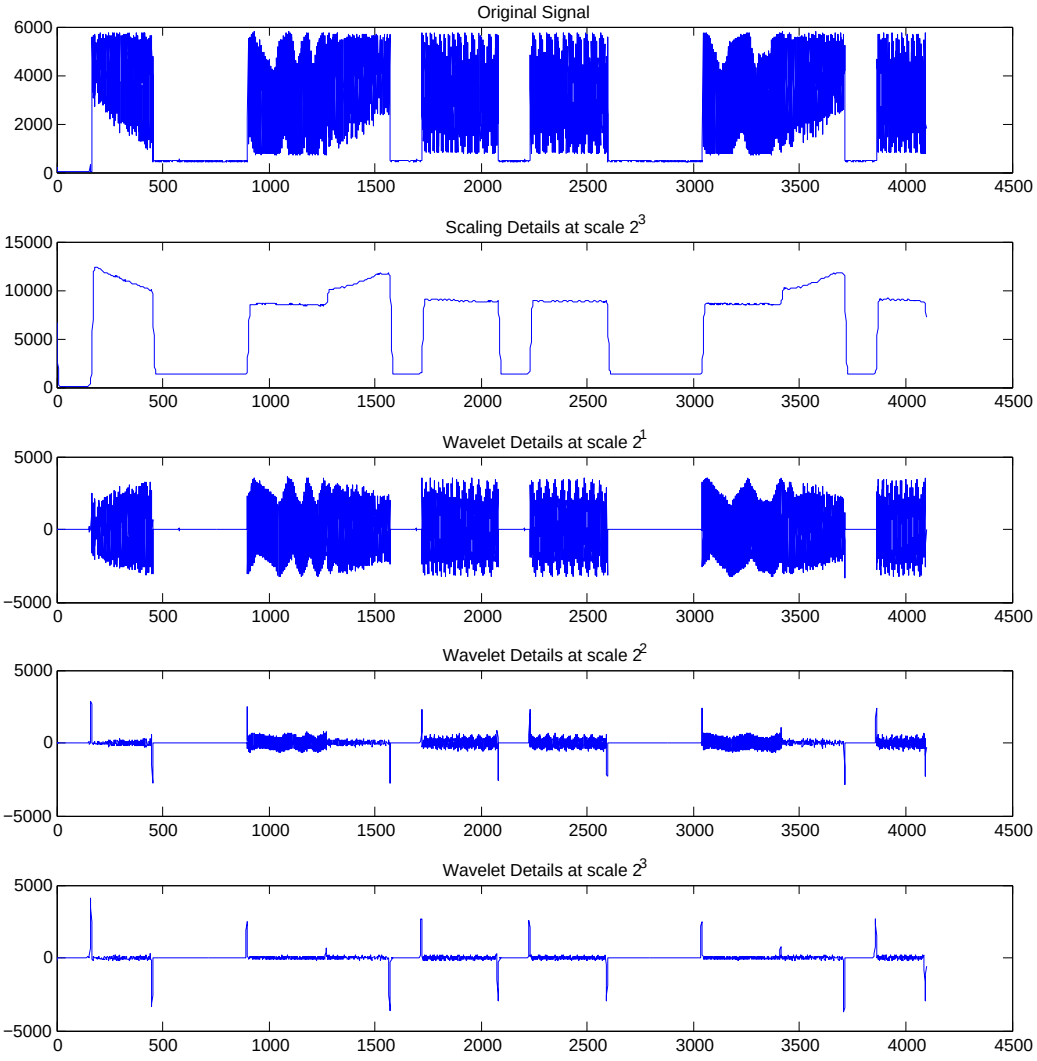

Figure 5: Multi-resolution Analysis. The top plot is the original signal.

4 Phase Detection Algorithm

As mentioned in the previous discussion of figure 5 this approach to phase detection will exploit the emergence peaks in the *Wavelet Details*. Also, because the *Wavelet Detail* data oscillates around zero, a very simple threshold can be utilized to identify peak locations.

This method is called the *Persistent Peak Algorithm*:

1: Perform the DWT with the Quadratic B-Spline Wavelet[5] on the time series
2: Discover the "best" resolution for identifying peaks
3: Find the "best" threshold to identify peak locations from this resolution.
4: Peak locations mark the program phases.

In step 1 the mechanics of the DWT were briefly defined in Section 3.3.2.

Defining the "best" resolution for peak identification is the heart of the algorithm and will be the focus of this section. In order to devise a scoring function for the quality of the peaks in a signal, we first need to define what is meant by peak. A peak is defined as a local maximum of the signal $f(t)$ in a neighborhood n about t

$$Peaks_n = \{f(t_p) \quad : \quad f(t_i) \leq f(t_p) \quad for \quad t_{p-n} < t_{p-n+1} < \cdots < t_p < t_{p+1} < \cdots t_{p+n}\}$$

The set of thresholded peaks is simply the peaks that are above a certain threshold

$$ThresholdPeaks_{n,threshold} = \{t_p \quad : \quad |f(t_p)| > threshold \quad for \quad f(t_p) \in Peaks_n\}$$

Using these definitions, a threshold score is computed as the number of thresholded peaks, i.e. the number of elements in the set $ThresholdPeaks_{n,threshold}$. The Persistent Peak Score is then the count of the largest sequence of contiguous thresholds where the threshold score is constant or close to constant. The intuition behind this score can be seen in figure 5. Notice as the scale increases peaks start becoming more and more definable. The persistence of a set of peaks in a given scale through a variety of thresholds is an indication that the phases are well defined. Again justification for this statement lies in the interaction of a wavelet's width and the time-frequency plane.

[5]Choosing the right wavelet function is an art guided by some deeper mathematical results that are beyond the scope of this paper and the current knowledge of the author. Instead, the choice of the Quadratic B-Spline Wavelet is based on visual inspection of the $WaveletDetail$ plots. During the investigation of various wavelet functions, the Quadratic B-Spline Wavelet produced peaks that were better defined than other wavelet functions. Currently this is thought to result from the wavelet's symmetry [19].

The procedure for computing the Persistent Peak Score is:

procedure $Persistent\_Peak\_Score(n, standard\_deviations, iterations)$

1: Compute the set $Peaks_n$
2: Set sd to the standard deviation of the data
3: $threshold\_increment = \dfrac{standard\_deviations * sd}{iterations}$
4: **for** $i = 1$ to $iterations$ **do**
5: $\quad threshold = i * threshold\_increment$
6: $\quad$ Compute the set $ThresholdPeaks_{n,threshold}$
7: $\quad$ Set $score(threshold) = |ThresholdPeaks_{n,threshold}|$, i.e. the number of thresholded peaks
8: **end for**
9: **return** The count of the largest sequence of contiguous thresholds where the threshold score is constant or close to constant.

The following plot shows the best and worst Persistent Peak Scores for the wavelet details of figure 5. The horizontal axis is the threshold increment, i.e. 1 = 1 threshold increment, 2 = 2 threshold increments, etc... The the vertical axis is the threshold score. Notice that the threshold score by definition is a decreasing function. Also notice that the Persistent Peak Score is defined as the largest region in this plot that is close to flat.

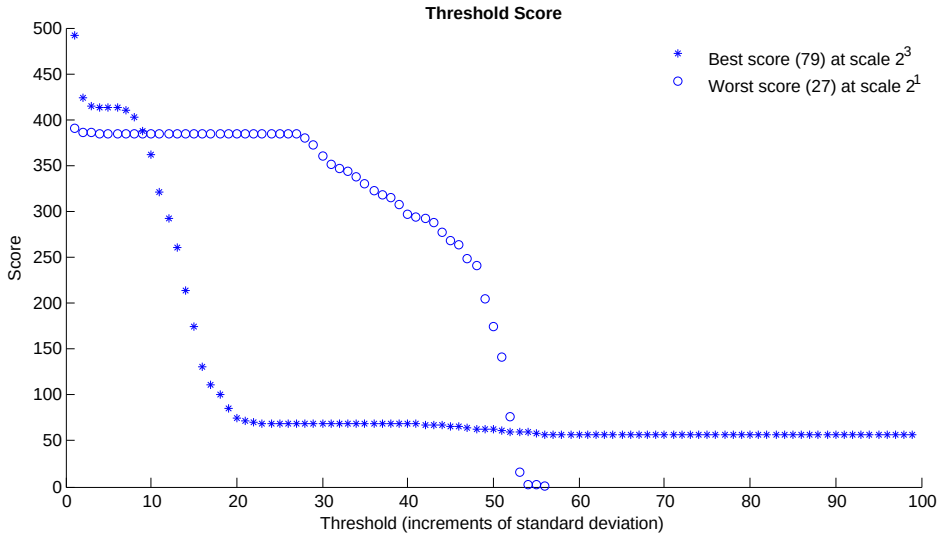

Figure 6: Threshold scores computed from the *Wavelet Detail* plots in figure 5.

Once a scale has been selected we must choose the peaks that will act as the phase markers. The *Persistent_Peak_Score* routine identifies a set of thresholds over which the peaks in the signal are stable. Any of these thresholds is a likely candidates to use. The results generated by this paper are based on choosing the largest threshold from the largest sequence of constant Persistent Peak Scores. The following figure shows the thresholds that were chosen for the wavelet scales from figure 5:

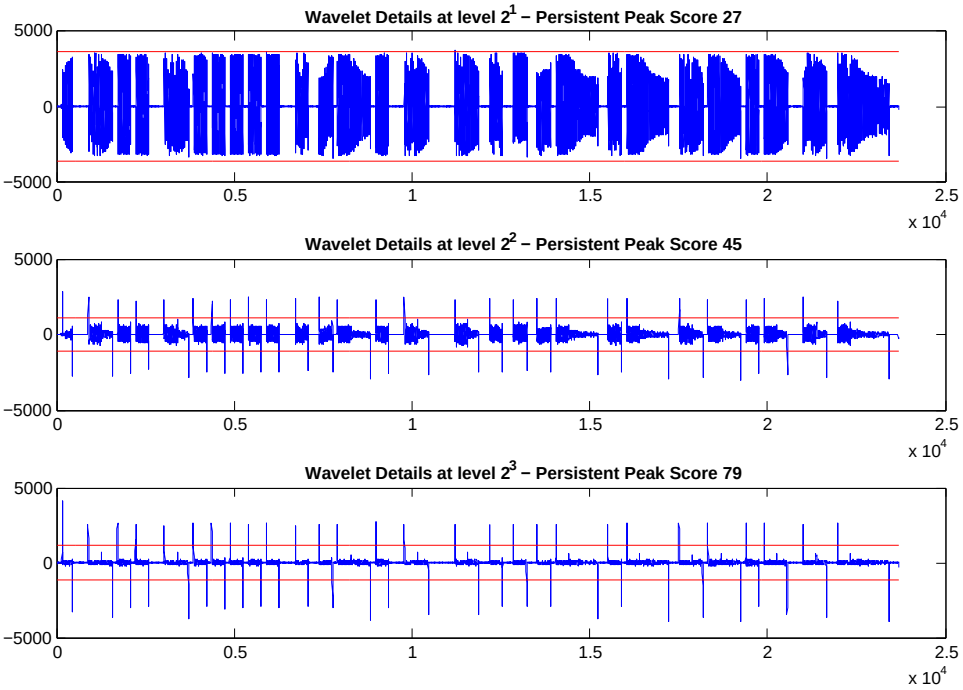

Figure 7: A look at the threshold that was chosen at each scale. The threshold lines appear in red. Note that scale 2^3 and its threshold line would be used to identify the phase markers.

5 Experiments

The experiments examined in this paper are based on sampling an L1 cache-miss hardware performance counter every 100,000 cycles. The reason for choosing to monitor cache memory is that over the past 30 years the performance gap between processors and memory has widened. In fact, the bulk of architecture advancements during this period has been to hide the latency of memory accesses [3]. Thus memory is seen as the most pressing bottleneck in improving overall system performance. These facts are also reflected in the amount of research being published in the phase detection/analysis of memory performance.

There are a variety of ways to capture a memory performance trace. Several papers use traces of detailed memory address pattern access [1]. The reason for choosing a cache-miss count time-series is that the instrumentation to capture this trace is very minimal. This data was collected from hardware performance counters through the Performance API (PAPI). For more information on PAPI see `http://icl.cs.utk.edu/papi`.

Five programs were tested. The first three are simple baseline tests that any good phase detection algorithm should pass. The source code for these tests is listed in Section 8. In a sense the phases of these programs are known, so they are a good first set of tests to run. The second two programs that are tested are bzip2 and gzip. They represent a first round of investigating real world programs as a way to determine if the Persistent Peak Algorithm merits further investigation.

Finally it should it should be noted that in this paper the validation of the results is by visual inspection. In visually assessing the phase detection two properties are looked for: First, phases should be regions in the signal that exhibit different behavior as indicated by the average cache-misses, the deviation about this average and frequency behavior in the region. Second the algorithm should be consistent in that if two regions satisfy the first requirement, they both will be detected as phases. This visual metric certainly lacks mathematical rigor, but serves as a first step in the assessment of this method. The Future Work section will discuss applying other metrics for validation.

In figures that follow, the horizontal axis is time increment in terms of 100,000 cycles. The vertical axis is the L1 cache-miss count. Also, phases are designated as the regions that fall between two adjacent red vertical lines.

5.1 Baseline Tests

The following three tests initialized two matrices, one in a row-major fashion and one in a column-major fashion. The difference between the three programs is in how they alternate between row-major and col-major initialization. The size of the matrices was chosen to be larger than the L1 cache size.

5.1.1 Simple Phase Program

The first test called the "Simple Phase" initializes one matrix in row-major fashion 10 times and then another matrix in col-major fashion 10 times. See the "Simple Phase" code listing for details. This will create two distinct phases in the cache-miss time-series. The row-major initialization phase will perform better than the col-major phase.

This fact can be seen in the first figure. The row-major phase occurs between times $t = 200$ and $t = 1500$. Note that the program initialization is occurring from $t = 0$ to around $t = 200$.

The column-major phase runs from $t = 1500$ to $t = 4000$. The second figure shows that the Persistent Peak algorithm is able to recognize these phases. Again, the region between adjacent red lines denotes a phase (note that no red line is plotted at $t = 0$ this is automatically a beginning of a phase). The figure clearly shows that row-major out performs col-major in its L1 cache use. The signature of the row-major and col-major phases appears in the next two experiments as well.

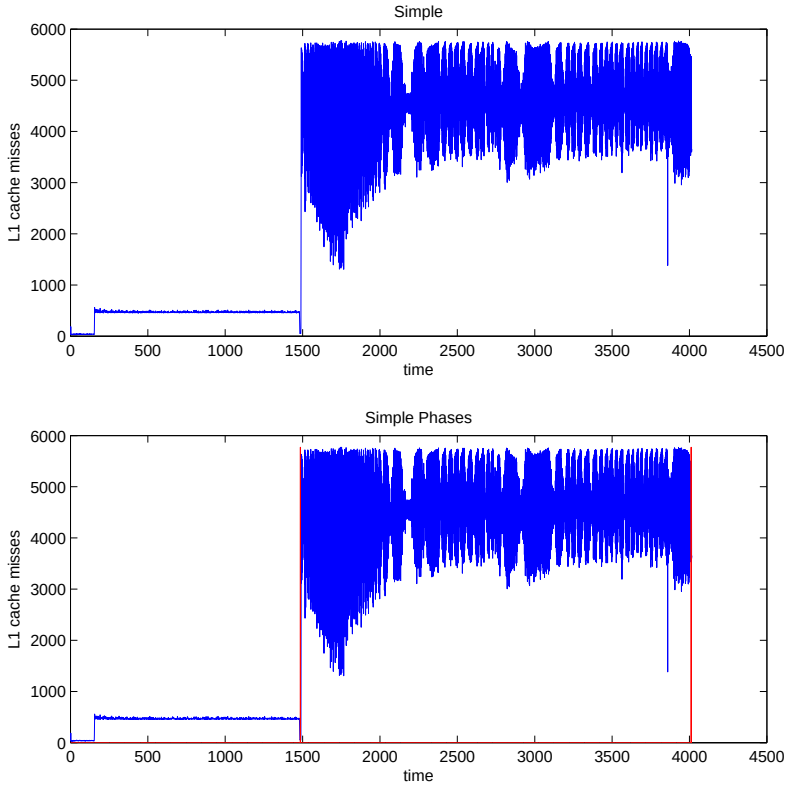

Figure 8: Simple Phase performance and phases discovered by the algorithm.

5.1.2 Alternating Phase Program

The "Alternating Phase" program repeatedly alternates between a row-major and column-major initialization of two matrices. See the "Alternating Phase" code listing for details.

This experiment represents a higher level of complexity than the previous experiment as several phases are now present. The phases are readily identifiable in the first plot where the alternating choice of row-major and column-major initializations is reflected in the alternating behavior of the time-series. As was the case in the first experiment the row-major phase is the lower average and less volatile portions of the figure.

The second figure shows that the Persistent Peak Algorithm is able to detect these phases:

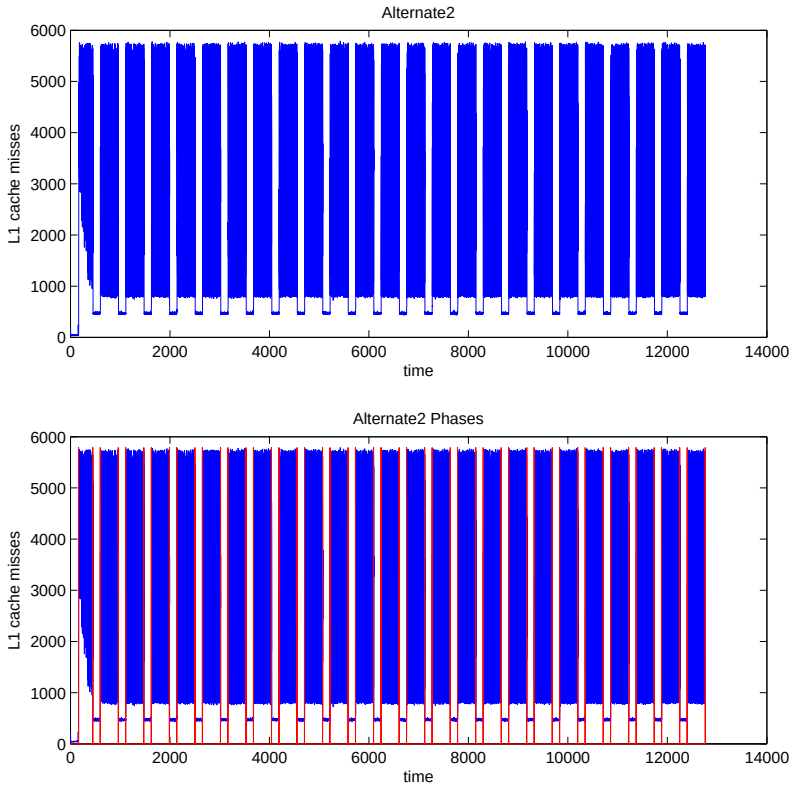

Figure 9: Phases discovered by the algorithm.

5.1.3 Random Phase Program

The "Random Phase" program randomly chooses either to do a row-major or column major initialization. See the "Random Phase" code listing for details. This experiment is slightly more complex than the "Alternating Phase" due to the introduction of a varying frequency in choice of row-major or column-major initializations.

Following the discussion of the previous two experiments, the row-major phases are the regimes of lower average and less volatility in the cache-miss behavior. The variable time length present in both phase types is a result of the random selection process.

The second figure shows that the Persistent Peak Algorithm is able to discover the phases:

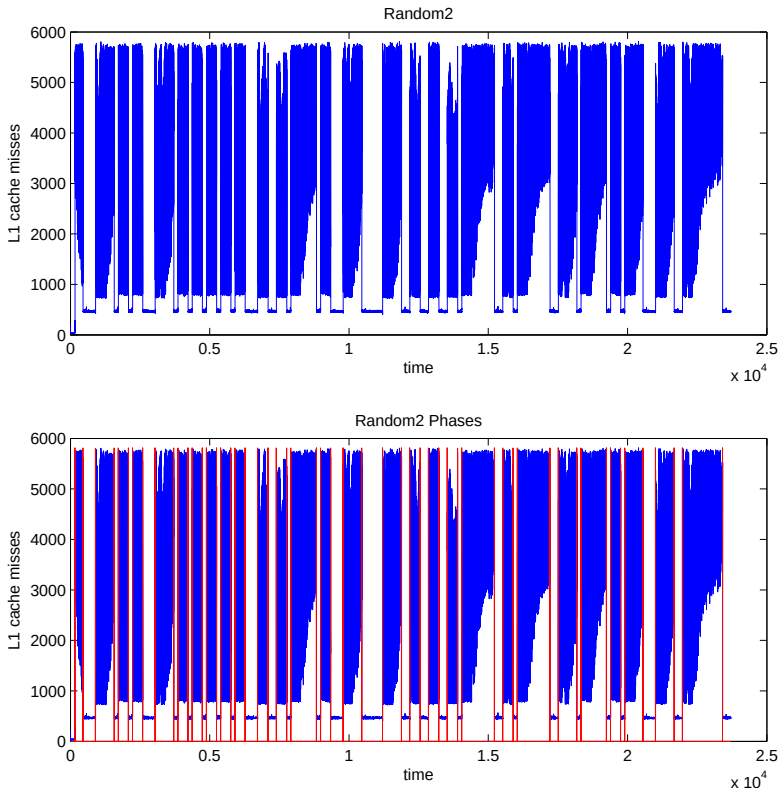

Figure 10: Phases discovered by the algorithm.

5.2 Real Program Tests

The following two experiments were conducted to test the validity of the Persistent Peak Algorithm on real-world applications. Bzip2 and gzip were chosen because they are often employed in other performance studies. The results show that the phases detected in bzip2 look reasonable according to the visual assessment of average cache performance, the spread about this average and the frequency content. The phases are also consistent. On the other hand the method fails on gzip due to a lack of definable phases in the signal.

5.2.1 bzip2

Because bzip2 works by repeatedly compressing a file until some stopping criteria is met, the program repeatedly executes code that accesses the cache in a common way. Thus phases are likely to be present in the cache-miss time-series. The first figure lends evidence to this view of bzip2. Notice the repeated structure that occurs six times throughout the signal. The second figure shows the results of the Persistent Peak Algorithm. The striking feature of this figure is the consistency of the detected phases across each of the six structures in the signal.

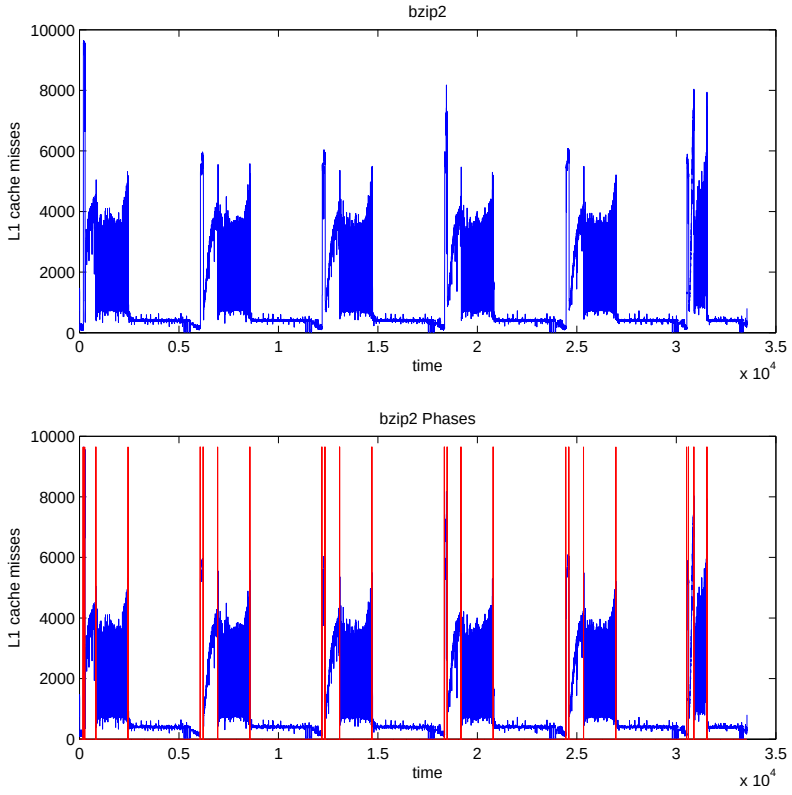

Figure 11: Phases discovered by the algorithm.

5.3 gzip

The following figures show the L1 cache-miss behavior of gzip. The cache behavior of gzip is distinctly different from the previous four experiments. Notice that there aren't any regions that differentiate behavior. Instead this signal looks simply like *one* region. Because the Persistent Peak Algorithm was designed to key in on the peaks/transitions that emerge in the DWT, this data series poses a huge problem. This is reflected in the second figure where the algorithm reports an unwieldy number of phases.

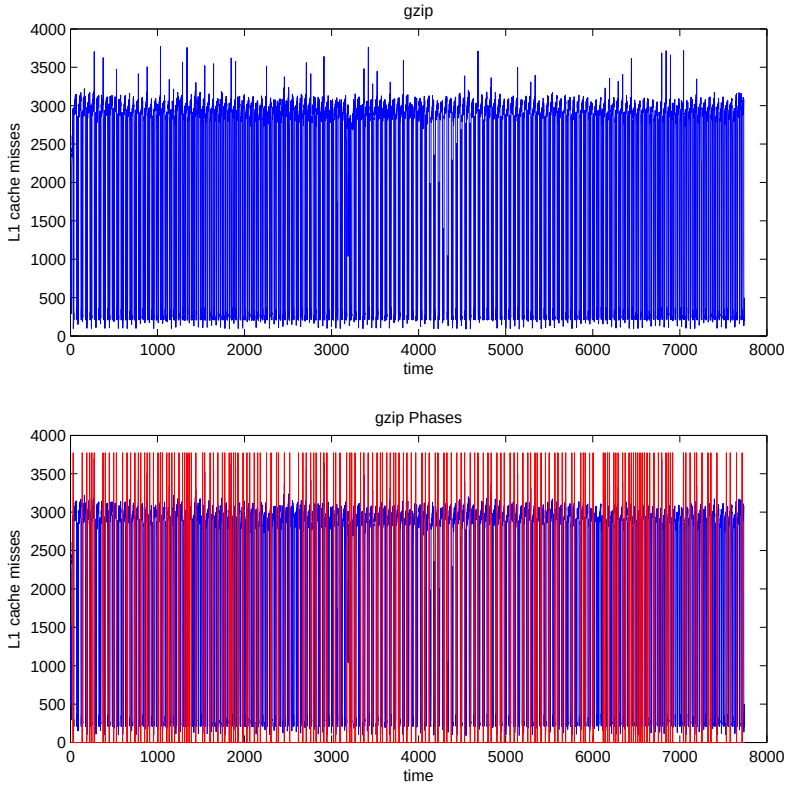

Figure 12: Phases discovered by the algorithm.

6 Conclusions and Future Work

This paper has provided a foundation to understand the wavelet perspective of time-frequency analysis. An application of the Discrete Wavelet Transform (DWT) to program phase detection was presented to solidify the concepts.

Evidence was provided to indicate that the Persistent Peak Algorithm is successful at identifying phases in signals that possess regions that display common behavior. Common behavior is defined by average cache-misses, the deviation about this average and frequency behavior. Furthermore the evidence suggests that when a signal satisfies these phase definition requirements, the algorithm is consistent in its choice of phases.

As shown by the gzip experiment though, this technique is not a solution for all time-series data. The failure is due to the definition of a phase taken in this paper. This is particularly interesting because it might lead to a category system whereby programs are described according to the phase-definitions they exhibit. As a result of this work, there are at least two categories of programs: bzip2 and the baseline kernels are in the category of the phase definition used in this paper, whereas gzip is not. A natural question is: Can the DWT be utilized to identify other notions of phases? Examining the DWT of more programs and designing new definitions of phases will be investigated in future work.

The most pressing question left unresolved in the current work is whether the phases discovered by the Persistent Peak Algorithm are correct. A metric needs to be identified to quantitatively answer this question. In [11], the Coefficient of Variation for several fixed sized intervals within a phase were used identify the internal consistency of the phase. The complexity metric defined by Cho and Li in [12] examines the phases at different wavelet scales. Another avenue that deserves investigation is a chaotic dynamical systems perspective discussed in [20]. Applying these metrics will also be investigated in future work.

Word Count: 5,471
(Not including Abstract, References and Code)

References

[1] T. Huffmire and T. Sherwood, "Wavelet-based phase classification," *PACT'15*, pp. 95–104, 2006.

[2] R. Balasubramonian, D. H. Albonesi, A. Buyuktosunoglu, and S. Dwarkadas, "Memory hierarchy reconfiguration for energy and performance in general-purpose processor architectures," *33rd International Symposium on Microarchitecture*, 2000.

[3] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*. Amsterdam: Morgan Kaufman Publishers, 4th ed., 2007.

[4] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. Sweeney, "Producing wrong data without doing anything obviously wrong!," *ASPLOS'14*, pp. 265–276, 2009.

[5] P. Hiremath, S. Shivashankar, and J. Pujari, "Wavelet based features for color texture classification with application to cbir," *IJCSNS International Journal of Computer Science and Network Security*, vol. 6, no. 9A, 2006.

[6] P. Oonicx, "Automatic phase detection in seismic data using the discrete wavelet transform," *Technical Report PNA-R9811*, 1998.

[7] J. Lau, S. Schoenmackers, and B. Calder, "Structures for phase classification," *ISPASS'04*, pp. 57–67, 2004.

[8] T. Sherwood, E. Perelman, and B. Calder, "Basic block distribution analysis to find periodic behavior and simulation points in applications," *Proceedings of the 2001 International Conference on Parallel Architectures and Compilation Techniques*, pp. 3–14, 2001.

[9] M. Zwick, M. Durkovic, F. Obermeier, and K. Diepold, "Setvectors for memory phase classification," *WCECS*, vol. 1, 2009.

[10] T. Sherwood, S. Sair, and B. Calder, "Phase tracking and prediction," *In the proceedings of the 30th Annual Intl. Symposium on Computer Architecture*, 2003.

[11] J. Lau, S. Schoenmackers, and B. Calder, "Transition phase classification and prediction," *11th International Symposium on High Performance Computer Architecture*, 2005.

[12] C. Cho and T. Li, "Complexity-based program phase analysis and classification," *PACT'15*, pp. 105–113, 2006.

[13] M. Casa, R. Badia, and J. Labarta, "Automatic phase detection of mpi applications," *Parallel Computing: Architectures, Algorithms and Applications*, vol. 38, pp. 129–136, 2007.

[14] X. Shen, Y. Zhong, and C. Ding, "Locality phase prediction," *ASPLOS'04*, 2004.

[15] S. Mallat, *A Wavelet Tour of Signal Processing: The sparse way*. Amsterdam: Academic Press, 3rd ed., 2009.

[16] S. Qian, *Introduction to Time-Frequency and Wavelet Transforms*. Upper Saddle River, NJ: Prentice Hall, 2002.

[17] A. Graps, "An introduction to wavelets," *IEEE Computational Science and Engineering*, vol. 2, no. 2, pp. 50–61, 1995.

[18] E. Micheli-Tzanakou, A. Ademoglu, and C. Enderwick, *Supervised and Unsupervised Pattern Recognition: Feature Extraction and Computational Intelligence (Industrial Electronics)*. New York: CRC Press, 1999.

[19] H.A.N.Dinh, D.K.Kumar, N.D.Pah, and P.Burton, "Wavelets for qrs detection," *Proceeding of the 23rd annual EMBS international conference*, 2001.

[20] Z. Alexander, T. Mytkowicz, A. Diwan, and E. Bradley, "Measurement and dynamical analysis of computer performance data," *IDA'9*, 2010.

7 Implementation of the Persistent Peak Algorithm

The Persistent Peak Algorithm was implemented using Matlab. The Matlab library *Wavelab* was used for wavelet processing. *Wavelab* can be found at `http://www-stat.stanford.edu/~wavelab`.

8 Code Listing

Listing 1: Simple Phases Program

```
int i, j, k;
const int size = 1000;

int a[size][size];
int b[size][size];

for(k=0; k<20; k++)
{
        if(k < 10)
        {
                for(i=0; i<size; i++)
                {
                        for(j=0; j<size; j++)
                        {
                                a[i][j] = 0;
                        }
                }
        }
        else
        {
                for(i=0; i<size; i++)
                {
                        for(j=0; j<size; j++)
                        {
                                b[j][i] = 0;
                        }
                }
        }
}
```

Listing 2: Alternating Phases Program

```
for(k=0; k<100; k++)
{
        int r=k % 2;

        if(r==0)
        {
                //initialize matrix a = 0 in row-major fashion
        }
        else
        {
                //initialize matrix b = 0 in col-major fashion
        }
}
```

Listing 3: Alternating Phases Program

```
srand ( time(NULL) );

for(k=0; k<100; k++)
{
        int r=rand() % 2;

        if(r==0)
        {
                //initialize matrix a = 0 in row-major fashion
        }
        else
        {
                //initialize matrix b = 0 in col-major fashion
        }
}
```