SCALING CONSTITUENT ALGORITHMS OF A TREND AND CHANGE DETECTION POLYALGORITHM

Rishu Saxena
Layne T. Watson

Valerie A. Thomas
Randolph H. Wynne

Departments of Computer Science and Mathematics
VPI & SU, Blacksburg, VA 24061
rishus@vt.edu

Department of Forest Resources and Environmental Conservation
VPI & SU, Blacksburg, VA 24061
thomasv@vt.edu

ABSTRACT

Earth observation satellites (EOS) such as Landsat provide image datasets that can be immensely useful in numerous application domains, by extracting information via time series analysis. While the literature is replete with algorithms, the size of the datasets itself is prohibitive, currently of the order of petabytes and growing, which makes them computationally unwieldy — both in storage and processing. An EOS image stack typically consists of multiple images of a fixed area on the Earth’s surface (same latitudes and longitudes) taken at different time points. Meaningful time series analysis on such interannual, multitemporal stack with existing state of the art codes can take several days on multicore servers. This work lays the foundation for a polyalgorithm based on two change detection algorithms, EWMACD and BFAST, for time series analysis of satellite image stacks, and presents speedup results for those two algorithms.

Keywords: Time series analysis, big data, change detection, parallel computing, load balancing.

1. INTRODUCTION

Land use and land cover change (LULCC) is of crucial importance globally. With anthropogenic activities such as deforestation and urbanization increasing exponentially through the past century, there have been significant changes in land cover in several parts of the world [8]. Simultaneously, significant changes in the global climate have also been observed, driven in part by LULCC (e.g., [7]). LULCC also has impacts on a wide variety of other ecosystem services. Much research is, therefore, being directed towards Earth monitoring.

Earth observation satellites (EOS) such as Landsat provide image datasets that, if harnessed well, can be immensely helpful towards LULCC monitoring. These images hold valuable information that can be very helpful in understanding and managing our natural resources.

Time series analysis (or, temporal trajectory analysis) is an excellent way of analyzing the satellite datasets for Earth monitoring and has been receiving increasing attention in the last decade, specifically, after the Landsat data became freely accessible in 2008 [17]. In time series analysis, several images of the scene under consideration, taken over a period of time, are stacked together chronologically, and are subsequently analyzed. Typically, the data set is converted into a collection of time series, each time series corresponding to a particular spectral band for one pixel. The objective is to discover a ‘trend’ in how different relevant variables (indicators) evolve over time. The analysis made is based on the behaviors of the time series of these variables. When the trajectory of one or more of the variables departs from the normal (or, predicted),
a change is detected. Several time series analysis algorithms have been proposed by different groups in the remote sensing community.

While several time series analysis algorithms have been proposed, the size of the datasets itself is prohibitive, currently of the order of petabytes and growing. An EOS image stack typically consists of multiple images of a fixed area on the Earth’s surface (same latitudes and longitudes) taken at different time points. Experiments on multicore servers indicate that carrying out meaningful time series analysis on a single interannual, multitemporal stack with existing state of the art codes can take several days. An HPC platform for time series analysis of satellite images obtained from MODIS was presented in [3]. In contrast with Landsat images, MODIS images have much coarser spatial resolution but much better temporal resolution. Knowing the need for scalable computations in remote sensing, several architectures have also been proposed. These mainly include massive parallel clusters, heterogeneous clusters, grid computing, GPUs and the like [9], [12], [14]. Considerable HPC work on classification (e.g., [13]) has also been carried out.

This paper presents parallel results for the constituents of a polyalgorithm under development for trend and change detection in Landsat images. The polyalgorithm consists of two algorithms, fundamentally distinct from each other, both by construction and in the phenomenon they capture. The algorithms are implemented in the scientific programming language Fortran 2003. Parallelization across pixels is implemented and further possibilities for speeding up the individual algorithms as well as the combined algorithm are discussed.

Experimental results of applying the codes to an image with approximately $10^8$ pixels are presented.

The two algorithms are presented in Section 2. Results for the algorithms individually are presented and discussed in Section 3. Section 4 describes parallel implementations, with conclusions and future work on the polyalgorithm in Section 5.

2. ALGORITHMS AND IMPLEMENTATION

Notation and definitions: For an $m \times n$ matrix $A$, an $n$-vector $x$, $I \subset \{1, \ldots, m\}$, $J \subset \{1, \ldots, n\}$, let $A_{IJ}$ denote the submatrix of $A$ formed from the rows indexed by $I$ and the columns indexed by $J$, and $x_J$ denote the subvector of $x$ indexed by $J$. $A_I$, $(A_J)$ are the rows (columns) of $A$ indexed by $I$ (J), respectively. An image is an $R \times C$ matrix $D$, where each $D_{rc}$ (pixel) is an $S \times B$ matrix, whose $(s, b)$ element $(D_{rc})_{sb}$ is the signal value at time index $s$ and frequency band index $b$.

2.1. Exponentially Weighted Moving Average Change Detection (EWMACD) [4], [5]

Algorithm EWMACD.

for band $b := 1$ step 1 until $B$
  for row $r := 1$ step 1 until $R$
    for column $c := 1$ step 1 until $C$
      begin
        Step 1: Write the time series data in the column $(D_{rc})_b$ as $(D_{rc})_b = \begin{pmatrix} u \\ v \end{pmatrix}$, where the $M$-dimensional vector $u$ is deemed training data and the $(S - M)$-dimensional vector $v$ as the test data. Let
        $$X = \begin{bmatrix} 1 & \sin t_1 & \cos t_1 & \cdots & \sin Kt_1 & \cos Kt_1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & \sin t_M & \cos t_M & \cdots & \sin Kt_M & \cos Kt_M \end{bmatrix}$$
        be the Gram matrix for the time points $t_1, \ldots, t_M$, using $K$ harmonics, where $M > 2K + 1$. The least squares fit to the training data $u$ is then written as $u(t) = \alpha_0 + \sum_{i=1}^{K}(\alpha_{2i-1}\sin it + \alpha_{2i}\cos it)$ with coefficients $\alpha = (X^tX)^{-1}X^tu$ and residual $E(\alpha) = u - X\alpha$. 
      end
Remark 1. In practice \( \alpha \) is computed via a QR factorization of \( X \), not by computing \((X^TX)^{-1}\) explicitly.

Next let \( I = \{i \mid |E(\alpha)_{i}| < \gamma_1\} \), where \( \gamma_1 \) is a user defined threshold and \( |I| > 2K + 1 \). Calculate the coefficients for an improved fit to the underlying signal as \( \alpha^* = ((X_I)^T X_I)^{-1} (X_I)^T u_I \). With the refined coefficients \( \alpha^* \), calculate the residuals for

(i) the complete time series \( (D_{rc})_{cb} \) as \( E^* (\alpha^*) = (D_{rc})_{cb} - \tilde{X} \alpha^* \), where \( \tilde{X}_s = (1, \sin t_s, \cos t_s, \ldots, \sin K t_s, \cos K t_s) \), for \( s = 1, \ldots, S \).

(ii) the outlier-free time series as \( (E^* (\alpha^*))_{\bar{I}} \), where \( \bar{I} = \{s \mid |E^* (\alpha^*)_s| < \gamma_2\} \), \( \gamma_2 \) is a user defined threshold, and

(iii) the outlier-free training set \( \hat{I} = \bar{I} \cap \{1, \ldots, M\} \) as \( (E^* (\alpha^*))_{\hat{I}} = u_{\hat{I}} - X_{\hat{I}} \alpha^* \), where \( |\hat{I}| > 2K + 1 \).

Remark 2. In the present implementation,

\[
\gamma_2 = \begin{cases} 
1.5\eta, & i \in [1, M], \\
20\eta, & i \in (M, S],
\end{cases}
\]

where \( \eta \) is the standard deviation of the first \( M \) elements of the residual vector \( E^* (\alpha^*) \).

Step 2: Define the control limit vector \( \tau \) by \( \tau_i = \mu + \sigma L \sqrt{\frac{2\lambda}{2-\lambda}(1 - (1 - \lambda)^2)} \), for \( i = 1, 2, \ldots, |\hat{I}| \), where \( \mu = 0 \) is used here, \( \sigma \) is the standard deviation of the outlier-free training data errors \((E^* (\alpha^*))_{\hat{I}} \), \( L \) is the multiple of this standard deviation \( \sigma \), and \( \lambda \in (0, 1] \) is the weight given to the most recent residual in the exponentially weighted moving average (EWMA) defined next. \( L \) is typically set to \( 3 \) or slightly smaller depending on the value of \( \lambda \).

Step 3: Let \( \tilde{I} = \{j_1, j_2, \ldots, j_{|\hat{I}|}\}, j_1 < j_2 < \cdots < j_{|\hat{I}|} \). Define the vector \( z \) by

\[
z_1 = (E^* (\alpha^*))_{j_1}, \quad z_i = (1 - \lambda) z_{i-1} + \lambda (E^* (\alpha^*))_{j_i}, \quad i = 2, \ldots, |\hat{I}|.
\]

This is the exponentially weighted moving average (EWMA) of the residual \((E^* (\alpha^*))_{\tilde{I}} \).

Step 4: Define the flag history \( S \)-vector \( f \) by

\[
f_s = \begin{cases} 
\text{sgn} (z_s) \left\lfloor \frac{|z_s|}{\tau_s} \right\rfloor, & s = j_i \in \tilde{I}, \\
0, & \text{otherwise}.
\end{cases}
\]

If there is a run of \( +1 \) or \( -1 \) in the values \( \text{sgn} (\Delta f_s) = \text{sgn} (f_{s+1} - f_s) \) of length \( \varpi \), called the ‘persistence’, signal a change at the index \( s \) beginning the (nonzero) run.

Remark 3. Missing data is automatically handled by not assuming that the time points \( t_i \) are equally spaced. Alternatively, missing data for time point \( t_k \) can be handled by including \( t_k \) in the sequence \((t_1, t_2, \ldots, t_M)\), but excluding \( t_k \) from the training sequence \((t_1, t_2, \ldots, t_M)\) and \( k \) from the sets \( I, \tilde{I} \), and \( \hat{I} \), which is equivalent to treating \((D_{rc})_{kb} \) as an outlier and to setting the flag \( f_k = 0 \).
2.2. Breaks For Additive and Seasonal Trend (BFAST) [16]

Algorithm BFAST.

Let $T = (t_1, \ldots, t_S)$ be the sequence of given time points and the $S$-vector $u$ denote the time series data in the column $(D_{rc})_b$, i.e., $u = (D_{rc})_b$. Assume that the general model is of the form $u = \mathcal{V} + \mathcal{W} + \epsilon$, where $\mathcal{V}$ and $\mathcal{W}$ denote the iteratively computed trend and seasonal components, respectively, present in the data and $\epsilon$ is the noise. The trend $\mathcal{V}$ may be piecewise linear and the seasonal component $\mathcal{W}$ may be piecewise harmonic. Let $N$ be the maximum number of iterations, $n$ be the iteration number, and $\mathcal{V}^n$ and $\mathcal{W}^n$ be the trend and seasonal components, respectively, computed at the $n$th iteration. Let $h, 0 \leq h \leq 1$, denote the proportion of data points by which two consecutive breakpoints $t_i$ and $t_j$ (including $t_1$ and $t_S$) must be separated. Thus $[Sh] \leq j - i - 1$. Take the length of moving windows to be $[Sh]$, initialize the iteration number $n := 1$, and initialize the seasonal component as $\mathcal{W}^0(T) = (w_0^0, \ldots, w_S^0)$.

**Step 1.1: Determine the possibility of breakpoints in trend.**

Eliminate the seasonal component from the data $u^n = u - \mathcal{W}^{n-1}(T)$. The ordinary least squares (OLS) estimator for the trend is given as

$$\hat{\alpha} = (X'X)^{-1}X'u^n$$

where

$$X = \begin{pmatrix} 1 & t_1 \\ \vdots & \vdots \\ 1 & t_S \end{pmatrix}.$$ 

The prediction error (or residual vector or the OLS residual) is defined as $E^o = u^n - X\hat{\alpha}$, where the superscript ‘$o$’ is used to signify the fact that these residuals are OLS regression based. Consider the process defined by the moving sums (MOSUM) of these OLS residuals

$$Q^o = \left\{ \frac{1}{\sigma \sqrt{|Sh|}} \sum_{i=k-[Sh]+1}^{k} E^o_i \right\}_{k=[Sh]}^{S},$$

where $\sigma$ is the sample standard deviation of all the OLS residuals.

Compute the OLS-MOSUM test statistic $\hat{f}^o = \max_{1 \leq k \leq S-[Sh]+1} |Q^o_k|$ as the maximum absolute value of this process, then compute the asymptotic critical value of the OLS-MOSUM test using the two-sided boundary-crossing probability $p_T = P[f^o > \hat{f}^o]$, where $p_T$ is read from the Brownian Bridge table.

A $p$-value less than a user defined parameter $\tau_V \in (0, 1)$ indicates the presence of breakpoints.

**Remark 1:** As discussed in [6], under the null hypothesis, the OLS-MOSUM process converges in distribution to the increments of a Brownian Bridge process.

**Step 1.2: Locate trend breakpoints.**

Suppose $p_T \leq \tau_V$. To locate the breakpoints, consider all possible partitions of the domain, compute OLS fits for each partition, and settle with a partition that yields minimum squared error.

Let $X_{[i,j]}$ denote the matrix formed from rows $i$ through $j$ of the matrix $X$, and $\alpha_{[i,j]}$ denote the least squares coefficients computed using the matrix $X_{[i,j]}$ with time points $t_i, \ldots, t_j$, and data $u^n_{[i,j]} = u^n_{[i,...,j]}$. For
i = 1, ..., S − [Sh] + 1, consider each window \([t_i, ..., t_{j-1}]\), \(i + 2 \leq j \leq S\), and the linear fit in this window. The recursive residual at point \(t_j\) is then defined as the weighted prediction error

\[
E^r_{ij} = \frac{u^n_{[j,j]} - X_{[j,j]}X_{[i,i-1]}^{-1}X_{[j,j]}}{\sqrt{1 + X_{[j,j]}^2}X_{[i,i-1]}^{-1}X_{[j,j]}}.
\]

The superscript ‘\(r\)’ is used to signify the fact that the process/statistic is recursive residual based.

Suppose a breakpoint has been found at \(t_j\). Then the cost of placing the next breakpoint at \(t_k\) is calculated as the accumulated sum of squared recursive residuals in the interval \([t_1, t_{k-1}]\), i.e., \(\rho_{ik} = \sum_{j=i+2}^{k-1} (E^r_{ij})^2\).

All possible positions for the breakpoints can thus be calculated by considering the moving sums of squared recursive residuals, i.e., the process defined by

\[
Q^r = \left\{ \left\{ \sum_{j=i+2}^{k} (E^r_{ij})^2 \right\} \right\}_{k=i+2}^{S-[Sh]+1}_{i=1}.
\]

Given the number \(\mu\) of desired interior breakpoints, let \(k_1, ..., k_\mu\) be integers such that \(k_{i+1} - k_i > [Sh]\), \(k_1 > [Sh] + 1\), and \(k_\mu < S - [Sh]\). Determine \(K = (1, k_1, ..., k_\mu, S)\) to minimize the moving sums of squared recursive residuals

\[
\sum_{i=3}^{k_1-1} (E^r_{1,i})^2 + \sum_{i=k_1+2}^{k_2-1} (E^r_{k_1,i})^2 + \sum_{i=k_2+2}^{k_3-1} (E^r_{k_2,i})^2 + \cdots + \sum_{i=k_\mu+2}^{S} (E^r_{k_\mu,i})^2.
\]

Then \((t_{k_1}, ..., t_{k_\mu})\) are the interior breakpoints in the trend component.

**Remark 2:** The breakpoints \(t_1, t_{k_1}, ..., t_{k_\mu}, t_S\) are optimal in the sense of the above moving sums of squared recursive residuals criterion.

**Remark 3:** If \(p^T > \gamma^\ast\), then there are only two breakpoints \((t_1\) and \(t_S)\) and no interior breakpoints. So this step is skipped and there is simply one linear fit over the entire domain \([t_1, t_S]\) (Step 1.3).

**Step 1.3:** Let \(k_0 = 1, k_{\mu+1} = S\), and \(I_0 = [t_{k_0}, t_{k_1}], I_1 = [t_{k_1}, t_{k_2}], ..., I_{\mu} = [t_{k_\mu}, t_{k_{\mu+1}}]\). For each interval \(I_i\), determine the linear regression coefficients

\[
\gamma^i = \left(X_{[k_i, k_{i+1}]}X_{[k_i, k_{i+1}]}^{-1}\right)^{-1}X_{[k_i, k_{i+1}]}u^n_{[k_i, k_{i+1}]}
\]

and construct the (discontinuous) piecewise linear fit \(V^n(t) = \sum_{i=0}^{\mu} \Gamma^i(t)\), where

\[
\Gamma^i(t) = \begin{cases} 
\gamma^0 + \gamma^i t, & t \in I_i, \\
0, & \text{otherwise}.
\end{cases}
\]

Let \(V^n(T) = (v^n_1, ..., v^n_S)\) be the sequence of values estimated at \(t_1, ..., t_S\) using this piecewise linear fit.

**Step 2.1:** Determine the possibility of breakpoints in seasons.

Eliminate the estimated trend component from the observed data \(\tilde{u}^n = u - V^n(T)\). The Gram matrix for the seasonal (harmonic) component is given by

\[
Y = \begin{pmatrix}
1 & \sin t_1 & \cos t_1 & \cdots & \sin K t_1 & \cos K t_1 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
1 & \sin t_S & \cos t_S & \cdots & \sin K t_S & \cos K t_S
\end{pmatrix},
\]
where $K$ is the degree of the trigonometric polynomial used for regression. The trigonometric regression coefficients for the seasonal component are computed as $eta = (Y^T Y)^{-1} Y^T \hat{u}$. The prediction error for this fit is defined as $E^n = \hat{u}^n - Y \beta$. The OLS-MOSUM process for these errors is given by

$$Q^n = \left\{ \frac{1}{\sigma \sqrt{|S|}} \sum_{i=k-[S]+1}^{k} E^n_i \right\}^S_{k=[S]},$$

and the OLS-MOSUM test statistic is $\hat{g}^n = \max_{1 \leq j \leq S-[S]+1} |Q^n_j|$. The two-sided boundary-crossing probability $p_S = P[\hat{g}^n > \hat{g}^o]$ is read from the Brownian Bridge table.

A $p$-value less than a user defined parameter $\tau_W \in (0, 1)$ indicates the presence of seasonal breakpoints.

**Step 2.2: Locate seasonal breakpoints.**

Suppose $p_S \leq \tau_W$. Using the same notation as for the trend breakpoints,

$$E^r_{ij} = \frac{\hat{u}^n_{[j,j]} - Y_{[j,j]}^T \beta_{[i,j-1]}}{\sqrt{1 + Y_{[j,j]}^T (Y_{[i,j-1]}^T Y_{[i,j-1]})^{-1} Y_{[j,j]}^T}}$$

is the recursive residual at time $t_j$, obtained by trigonometric regression in the time window $[t_i, t_{j-1}]$.

Given the number $\nu$ of desired seasonal interior breakpoints and a minimum number of data points separating breakpoints (as for the trend), let $l_1, \ldots, l_\nu$ be integers such that $l_{\nu+1} - l_1 > [S], l_1 > [S] + 1,$ and $l_\nu < S - [S]$. Determine $L = (l_1, l_1, \ldots, l_\nu, S)$ to minimize the moving sums of squared recursive residuals

$$\sum_{i=3}^{l_1-1} (E^r_{1,i})^2 + \sum_{i=l_1+2}^{l_2-1} (E^r_{1,i})^2 + \sum_{i=l_2+2}^{l_3-1} (E^r_{1,i})^2 + \sum_{i=l_3+2}^{l_4-1} (E^r_{1,i})^2 + \cdots + \sum_{i=l_{\nu+2}}^{S} (E^r_{1,i})^2.$$

Then $(t_{l_1}, \ldots, t_{l_\nu})$ are the interior breakpoints in the seasonal component.

**Remark 4:** If $p_S > \tau_W$, then there are only two breakpoints $(t_1$ and $t_S$) and no interior breakpoints. So this step is skipped and there is simply one trigonometric polynomial fit over the entire domain $[t_1, t_S]$ (Step 2.3).

**Step 2.3:** Let $l_1 = 1, l_{\nu+1} = S$, and $J_0 = [t_1, t_{l_1}), J_1 = [t_{l_1}, t_{l_2}), \ldots, J_\nu = [t_{l_\nu}, t_{l_{\nu+1}}]$. For each interval $J_j$ determine the trigonometric polynomial regression coefficients

$$\delta^j = (Y_{[l_j,l_{j+1}]}^T Y_{[l_j,l_{j+1}]})^{-1} Y_{[l_j,l_{j+1}]}^T \hat{u}^n_{[l_j,l_{j+1}]}$$

and construct the (discontinuous) piecewise trigonometric polynomial $W^n(t) = \sum_{j=0}^{\nu} \Delta^j(t)$, where

$$\Delta^j(t) = \begin{cases} \delta^j_1 + \sum_{k=1}^{K} \delta^j_{2k} \sin kt + \delta^j_{2k+1} \cos kt, & t \in J_j, \\ 0, & \text{otherwise.} \end{cases}$$

Let $W^n(T) = (w^n_1, \ldots, w^n_S)$ be the sequence of values estimated at $t_1, \ldots, t_S$ using this piecewise trigonometric polynomial approximation.

**Step 3: Compare the breakpoints between iterations $n - 1$ and $n$.**

If the Hamming distance between the two breakpoint vectors $(t_{k_1}, \ldots, t_{k_\nu}, t_{l_1}, \ldots, t_{l_\nu})$ at iterations $n - 1$ and $n$ is less than some defined tolerance or the number of iterations has reached $N$, then exit. Otherwise, increment the iteration number $n$ and repeat Steps 1.1 to 3.
Figure 1. Processed NDVI values from Landsat images for 2009 (left) and 2014 (right). The $x$- and $y$-axes represent relative pixel coordinates of the extracted image.

3. RESULTS AND DISCUSSION

The algorithms EWMACD and BFAST were tried on study areas located in Oregon and South Carolina, with results presented for the latter. Figure 1 shows the satellite images taken from Landsat path 18, row 37, on dates January 3rd, 2009, and February 16th 2014, corresponding to the beginning and end of the image stack (henceforth referred to as SC1837) under consideration, which has 198 time points and one band, the normalized difference vegetation index \( \text{NDVI} = \frac{\text{NIR} - R}{\text{NIR} + R} \), where NIR is the near infrared (band 4, biomass) and $R$ is the visual red (band 3, vegetation slopes). NDVI is known to be a good metric for vegetation cover, where negative values of NDVI are deemed irrelevant as they correspond to water, clouds, or missing observations. For processing, positive values of NDVI are scaled by 10,000 and negative values are masked out (set to $-9,999$). The image stack dimensions are $R = 7411$, $C = 8801$, $S = 198$, $B = 1$. Like most time series analysis algorithms, both algorithms rely on user defined parameters, requiring, in general, a priori knowledge of the scene. The experiments here adhered to the published values of the parameters for each of these algorithms, which are listed in Table 1.

For EWMACD, all the time points in the years 2009 to 2011 were used as training data, so the length $M$ of the training period varied from pixel to pixel.

<table>
<thead>
<tr>
<th>Table 1. Algorithm parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>EWMACD</strong></td>
</tr>
<tr>
<td>$K = 2$</td>
</tr>
<tr>
<td>$L = 0.5$</td>
</tr>
<tr>
<td>$\lambda = 0.3$</td>
</tr>
<tr>
<td>$\varpi = 7$</td>
</tr>
</tbody>
</table>
Validation is based on tree canopy cover (TCC) data [15]. For a $30m \times 30m$ area (one pixel) the TCC is defined as the proportion of the area that is covered by tree canopy versus “not tree canopy”. Methods to measure the TCC for a pixel are known. For the study area SC1837, TCC data exists for 457 pixels. For each of these pixels, at the NDVI data band, the two algorithms were run on the image stack. Results for two such pixels are displayed in Figure 2. TCC for the pixel in Figure 2(left) reduced from 89.67% in 2009 to 67.2% by the year 2013, an approximately 22% loss. BFAST captures the loss correctly indicating a decline in vegetation cover, with a quick, short recovery towards the end of the year 2014 (the available TCC data does not cover late 2014). EWMACD clearly captures the loss correctly.

The pixel displayed in Figure 2(right), on the other hand, gained approximately 18% in TCC. The NDVI values also, in general, show an increase in mean. BFAST correctly captures the trend and indicates gradual recovery throughout, with some disturbance towards the end of 2014. EWMACD, however, indicates a disturbance just after it’s training period (beginning of 2012) — a sharp loss followed by some recovery and stability thereafter. This behavior can be attributed to ill-chosen parameter values for this pixel.

The logical mathematical description of an image stack uses the index order $(r, c, s, b)$, but because of Fortran array element storage order and the hardware effects of cache misses and paging, the image stack is actually stored and processed in the index order $(s, c, r, b)$. Failure to use this latter index order can result in a cache miss rate as high as 28%. The outcomes of both the sequential and the parallel (Fortran 2003) codes match with those of the original codes (written in R) completely for EWMACD and closely for BFAST. The slight deviation in results for BFAST can be attributed to its recursive use of models, which makes the algorithm sensitive to round-off error. Figure 3 displays the EWMACD results on a few (4) time points. At any given time point, a black pixel indicates being flagged by EWMACD as having no disturbance, a green pixel as in recovery, and a red pixel as in loss. So an area to the northeast of pixel $(4500, 4000)$ was in vegetation loss in September 2013 while by February 2014, the entire area to the right of $(4000, :)$ had substantially recovered.

Both algorithms involve a fair number of intermediate array variables, so global arrays were used for all work arrays, and Fortran vector instructions were utilized wherever possible.

4. PARALLEL IMPLEMENTATION

One image from a given Landsat path/row typically consists of more than $10^7$ pixels. The sequential implementation of BFAST in Fortran takes over 7 hours to analyze the time series of a $1000 \times 1000 = 10^6$
Figure 3. EWMACD flags for four late time points of image stack when the data in 2009–2011 was used as training data.

pixel image. EWMACD is faster but still takes 1 hour and 20 minutes for $10^6$ pixels. Knowing that (i) the image stack discussed in this paper consists of only 198 time points (2009 to 2014) while there are currently 900 time points (1984 to 2014) actually available, (ii) the run times discussed here are for a single path/row only while there are 450 path/rows in the US alone, and other similar facts, scaling the codes is imperative for any meaningful analysis. The sequential codes described in the previous sections were parallelized using OpenMP. The hardware bottlenecks and computational hot spots are systematically identified and addressed.

The sequential code already harnesses vector instructions wherever possible. The input and output arrays are the only large arrays; the indexing mentioned in Section 3 ensures good memory locality. The remaining significant hardware bottleneck is load imbalance. Specifically, since the time series processing for any given pixel is independent of that for any other pixel, the algorithms are apparently embarrassingly parallel with respect to pixels. Landsat images, however, suffer from missing observations (due to factors beyond human control), thereby resulting in ‘invalid’ pixels (a pixel is declared invalid if there are fewer than $2^K + 1$ observations in the entire time span, cf. Section 2.1). These invalid pixels are randomly distributed across the data. This induces a very high work load imbalance across the pixels. Attempting to weed out invalid pixels in a preprocessing step and execute the PARALLEL DO loop for only valid pixels leads to CPU underutilization (from 99.9% to 70–80%), simultaneously increasing the OpenMP time. This presumably is
due to memory contention: the memory access pattern for the latter approach is such that multiple threads try to access the same memory bank(s). Furthermore, even amongst the valid pixels, the total number of observations \(S\) available for one pixel can be much less than the number of observations available for some other pixel. So, even with this preprocessing approach the work load balance is not guaranteed.

Finally, allocating/deallocating arrays within each thread is inefficient. Allocatable global arrays in modules can be used by threads via THREADPRIVATE, but this data copy mechanism does not work with dynamic loop scheduling [11], which is desirable because of the large variance in pixel analysis times (including missing data for a pixel). The best alternative is using Fortran automatic arrays with OpenMP PRIVATE.

Next, the computational hot spots are identified. For EWMACD, more than 50\% of the time is spent in least squares fitting (cf. Section 2, Step 1), specifically in DGELS (LAPACK) calls. LAPACK [2] is already optimized for the hardware. 22\% of the total time is in the calculation of residuals (again, cf. Section 2.1, Step 1). This subroutine has two DO loops with dependencies and cannot be vectorized.

For BFAST, approximately 97\% of the OpenMP time is spent in computing the recursive residuals (cf. Section 2.2, steps 1.2 and 2.2). Essentially, linear and harmonic least squares fits are done in every permissible interval, and the least squares fitting is already optimized.

In summary, after considering and testing several alternatives, the best approach found was to (1) perform the raw binary stream input data order \((r,c,s,b)\) conversion to \((s,c,r,b)\) order in parallel; (2) cull invalid (including missing) pixels inside the subroutines EWMACD and BFAST, which are called from within a PARALLEL DO (a pixel is declared invalid if there are fewer than \(2K+1\) observations in the entire time span, cf. Section 2.1); (3) convert the nested DO loop \(\text{DO } r=1,R; \text{ DO } c=1,C\) into a single PARALLEL DO loop \(\text{DO } k=1,R*C,A\); (4) use OpenMP SCHEDULE(DYNAMIC, 1); (5) process a chunk of \(A\) pixels indexed by \(k\) on each call to EWMACD and BFAST; (6) use automatic rather than allocatable arrays for all small work arrays in the subroutines, and allocate/deallocate just one large work array in both EWMACD and BFAST; (7) perform all I/O outside parallel OpenMP constructs to reduce memory and disk contention. Note that manually collapsing the nested loops and chunking within the pixel processing subroutine is more efficient than collapsing and chunking at the OpenMP directive level, since the latter would call the subroutines, which allocate and deallocate numerous work arrays, for each pixel index. The difficulty of load balancing “embarrassingly parallel” applications is analyzed theoretically in [1].

Figure 4(left) shows the scaled speedup for BFAST, i.e., increasing both the problem size and the number of cores. The isoefficiency (constant efficiency as both the problem size and number of cores are scaled up)
decreases significantly, indicating some combination of poor load balancing (the pixel chunk size $A = 100$), main memory contention, and increasing thread management overhead, as yet unresolved.

Parallel results for the full scene, which consists of $7411 \times 8801 = 65224211$ pixels at one band, processed only with EWMACD, are shown in Figure 4(right). Using 64 cores, the full scene is processed in 1.86 minutes, with a speedup of roughly 46. The input binary image is 25GB. For this image, the code needs 74GB of memory. When a single thread (core) is used, the cache miss rate is 0.566% and 1.03 instructions per cycle are executed. For 64 cores, the cache miss rate is 0.721% and 0.52 instructions per cycle are executed. On 64 cores, the FLOPS performance of the EWMACD code is 48.54 GFLOPs. The peak theoretical performance for this machine is 358.4 GFLOPs, yielding performance to peak ratio of $48.54/358.4 = 13.54\%$.

For the algorithms to be used together in a polyalgorithm, speeding up BFAST for a single pixel needs to be aggressively explored, lest BFAST be used only on a need basis ($> 30$ hours for this full image).

For all the Fortran codes, the input (as well as output) image stacks were in binary file format. Fortran I/O with streaming access was utilized to read and write these files. The results presented in this paper were obtained on a single machine: 64 core AMD Opteron 6276, 1.4GHz CPU, 2MB cache per core, 265 GB main memory, CentOS, gfortran compiler version 4.8.

5. CONCLUSIONS AND FUTURE WORK

Given the inconsistent trend predictions between the different algorithms, the sometimes erratic behavior of a given algorithm on a given image stack, the sensitivity to parameters for some algorithms, and the prohibitive execution times for serial codes, there is clearly a need for a parallel polyalgorithm (an intelligent, adaptive union of multiple algorithms). The work begun here, assessing the scalability and memory footprint, the parameter sensitivity, and the range of applicability of individual algorithms is but the first step toward such a parallel polyalgorithm for hypertemporal Landsat image stacks.

ACKNOWLEDGEMENTS

The authors would like to thank the anonymous referees for their useful comments, Jill Derwin for remote sensing technical assistance and Dr. E. B. Brooks for useful discussions. This work was supported in part by NSF Grant CNS-1565314 and USDA Grant 422350.

REFERENCES


Saxena, Watson, Thomas, and Wynne


**AUTHOR BIOGRAPHIES**

**R. SAXENA** (Ph.D., Arizona State, 2008) has interests in image processing, numerical analysis, high performance computing, and big data analytics.

**LAYNE. T. WATSON** (Ph.D., Michigan, 1974) has interests in numerical analysis, mathematical programming, bioinformatics, and data science. He has been involved with the organization of HPCS since 2000.

**VALERIE A. THOMAS** (Ph.D., Queen’s, Ontario, 2006) has interest in the use of remote sensing to examine forest canopy physiology, structure, and function across space and time.

**RANDOLPH H. WYNNE** (Ph.D., Wisconsin, 1996) has interests in the applications of remote sensing to forest monitoring, modeling, and management. He has long worked at the interface of computing with basic and applied remote sensing science.