Homework #1: Wavelet Compression on Imagine
Due Date: Tuesday, 30 April 2002
Checkpoints: Tuesday, 23 April 2002; Friday, 26 April 2002

The purpose of this assignment is to give you a feel for stream programming. To achieve this, you will program a simple application for Imagine using the Imagine tool chain. The application is a one-dimensional signal compression based on wavelets.

Please work on this assignment in groups of up to four people, and hand in one write-up per group.

I'll give a very brief introduction to wavelets and wavelet compression, and then a more detailed description of the algorithm you will implement. If you are interested in what exactly is required in this assignment, go to Section 3 on page 6.

1 What is Wavelet Compression

This section provides a very brief description of compression using wavelets.

1.1 What are Wavelets

The most basic definition of a wavelet is simply a function with a well defined temporal support that "wiggles" about the $x-$axis (it has exactly the same area above and below the axis).

This definition however does not help us much, and a better approach is to explain what the wavelet transform and wavelet analysis are.

The basic Wavelet Transform is similar to the well known Fourier Transform. Like the Fourier Transform, the coefficients are calculated by an inner-product of the input signal with a set of orthonormal basis functions that span $\Re^1$ (this is a small subset of all available wavelet transforms though). The difference comes in the way these functions are constructed, and more importantly in the types of analyses they allow.

The key difference is that the Wavelet Transform is a *multi-resolution* transform, that is, it allows a form of time–frequency analysis (or translation–scale in wavelet speak). When using the Fourier Transform the result is a very precise analysis of the frequencies contained in the signal, but no information on when those frequencies occurred. In the wavelet transform we get information about when certain features occurred, and about the *scale* characteristics of the signal. Scale is analogous to frequency, and is a measure of the amount of detail in the signal. Small *scale* generally means coarse details, and large scale means fine details (scale is a number related to the number of coefficients and is therefore counter-intuitive to the level of detail).

The Discrete Wavelet Transform can be described as a series of filtering and sub-sampling (decimating in time) as depicted in Figure 1. In each level in this series, a set

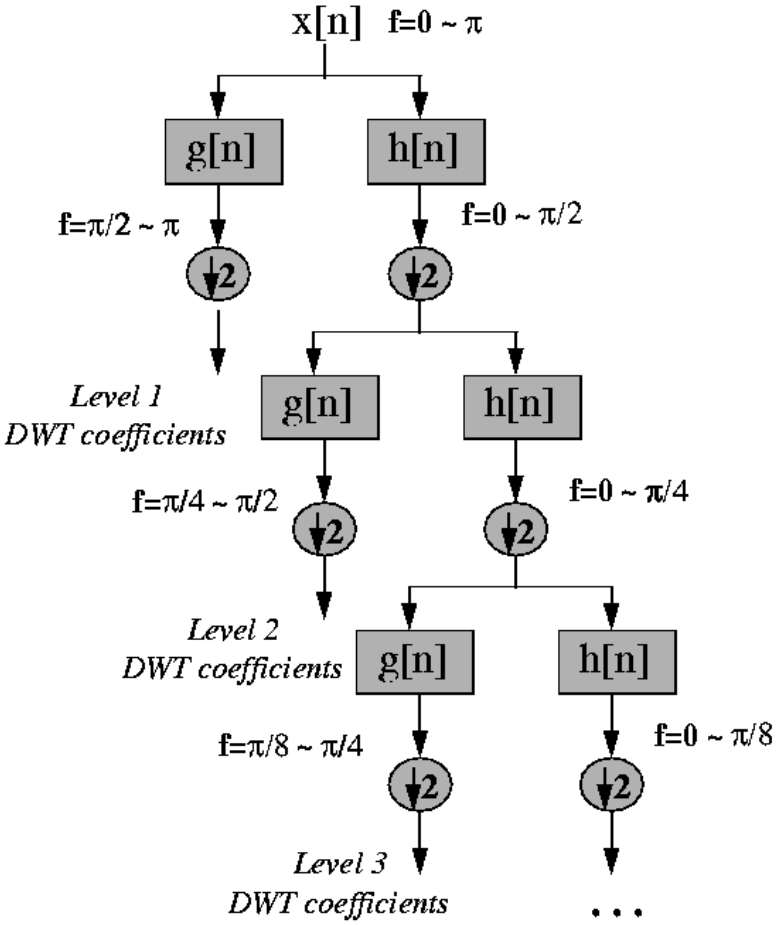

Figure 1: Discrete Wavelet Transform (from http://www.public.iastate.edu/~rpolikar/WAVELETS/WTpart4.html

of 2^{j-1} coefficients are calculated, where $j < J$ is the *scale* and $N = 2^J$ is the number of samples in the input signal. The coefficients are calculated by applying a high-pass *wavelet filter* to the signal and down-sampling the result by a factor of 2. At the same level, a low-pass *scale filtering* is also performed (followed by down-sampling) to produce the signal for the next level. Both the wavelet and scale filters can be obtained from a single Quadrature Mirror Filter (QMF) function that defines the wavelet.

Each set of scale-coefficient corresponds to a "smoothing" of the signal and the removal of details, whereas the wavelet-coefficients correspond to the "differences" between the scales. Wavelet theory shows that from the coarsest scale-coefficients and the series of the wavelet-coefficients the original signal can be reconstructed. The total number of coefficients (scale + wavelet) equals the number of samples in the signal.

1.2 Wavelet Compression

How do we use these transform coefficients to perform compression? The distribution of values for the wavelet coefficients is usually centered around 0, with very few large coefficients. This means that almost all the information is concentrated in a small fraction of the coefficients and can be efficiently compressed. This is done by quantizing the values based on the histogram and encoding the result in an efficient way, e.g. Huffman Encoding. For this homework we will use a simpler method, and instead of quantizing we will discard all but the M largest coefficients. This provides a compression ratio of roughly $2M/N$ (the factor of 2 is for storing both the coefficient value and index).

1.3 Where to Get More Information

A very good book on wavelets is "A Wavelet Tour of Signal Processing, 2^{nd} edition" by Stéphane Mallat.

I also found several reasonable tutorials on the web such as:

- cas.ensmp.fr/~chaplais/Wavetour_presentation/Wavetour_presentation_US.html
- http://www.public.iastate.edu/~rpolikar/WAVELETS/WTtutorial.html
- http://perso.wanadoo.fr/polyvalens/clemens/wavelets/wavelets.html

2 Algorithm Description

The compression algorithm has two parts. The first is a wavelet transform that uses the Fast Wavelet Transform taken from the Wavelab library developed at the Statistics Department here at Stanford. This is a rich library of Matlab functions dealing with wavelets and wavelet analysis, and you are encouraged to check out their web page at http://www-stat.stanford.edu/ wavelab/. After calculating the transform coefficients a sort is applied, and all but the largest n coefficients are discarded. The signal can be reconstructed by performing an Inverse Wavelet Transform using the n stored coefficients and zeroing out all other coefficients.

I will now describe in more detail the simplest wavelet transform which is an orthogonal and periodic transform, and implemented by the FWT_PO.m function. You can find this function and the entire Wavelab distribution in
/afs/ir.stanford.edu/class/ee482c/wavelab/Orthogonal.

The pseudocode for the Fast Wavelet Transform algorithm appears below. The algorithm consists of a main loop that has two parts. This first part calculates the wavelet coefficients in the current scale (high-pass filter), and the second part calculates the scale coefficients by low-pass filtering and essentially shifts the scale down (towards less details). This loop is repeated $O(\log n)$ times, once for each scale. Notice that the amount of computation in each iterations shrinks by a factor of two for each scale lowered. As a result the total computation cost is $O(n)$, and the filter is applied to roughly $4n$ elements.

Both the low-pass and high-pass filtering are performed on a periodically padded version of the current-scale signal ($\varphi_j(n)$), and the result is decimated by 2. The filter kernels define the wavelet and are computed from its characteristic QMF.

2.1 Fast Wavelet Transform Pseudocode

```
// I've used some Matlab like notation:
// 1. Arrays are indexed from 1 to their length instead
//    of 0 - (length - 1)
// 2. x(start:end) means all elements of x from start index
//    to end index
// 3. [x y] mean concatenate y to x
// 4. x + 1 means add 1 to all elements of x
// 5. [start:end] is a vector which includes all numbers from start to end
// 6. x(y) - a vector of elements of x with indices y
// 7. [start:stride:end] a vector from start to end with a constant stride
//           [1:2:10] = [1 3 5 7 9]


function WFT(x, qmf) {
// x - input signal
// qmf - qmf of the wavelt function
  for (i = 1; i<=length(qmf); i++) {
    hi_filter(i) = -(((-1)^i) * qmf(i)); // high pass filter kernel for the wavelets
    lo_kernel(i) = qmf(length(qmf)-i+1); // low pass filter kernel for the scales
}

  FinestScale = log2(length(x)) - 1; // the scale of the input
  CoarsestScale = L; //  a parameter, usually 1 < L < 8, and it represents the coarsest
                     //  level of space detail we care about
                     //  for good results L << log2(x)

  xx = x;
  for (j = FinestScale; j >= CoarsestScale; j--) {
    tmp = DownScale_HiPass([xx(2:length(xx)) xx(1)], hi_kernel); // Wavelet coefs for this sc
    CurrentScaleIdx = [2^j+1:2^(j+1)];
    result(CurrentScaleIdx) = tmp;
    xx = DownScale_LoPass(xx, lo_kernel); // phase coefs for next scale
  }
  FinalScale=[1:2^CoarsestScale];
  result(FinalScale) = xx; // final coefficients
  return result;
}
```

```
function DownScale_HiPass(x, filt) {
// x - input signal
// filt - filter kernel

  if( length(filt) <= length(x) ) {
    xpadded = [x(length(x)+1-length(filt):length(x)) x]; // periodical padding
  }
  else { // the signal in the current scale is shorter than the filter
    tmp = [1:length(filt)];
    imod = 1 + ((length(filt)*(length(x)-1) + tmp - 1)%length(x));
    xpadded = [x(imod) x]; // a periodic padding based on the ''period'' of the filter
  }

  result  = FIR(xpadded, filt);
  result  = result((length(filt)+1):(length(x)+length(filt))); // unpad the result
  result  = result(1:2:length(result)-1); // decimate (down-sample) by 2
}

function DownScale_LoPass(x, filt) {

  if( length(filt) <= length(x) ) {
    xpadded = [x x(1:length(filt))]; // periodical padding
  }
  else { // the signal in the current scale is shorter than the filter
    tmp = [1:length(filt)];
    imod = 1 + (tmp - 1) % length(x);
    xpadded = [x x(imod)]; // a periodic padding based on the ''period'' of the filter
  }

  result  = FIR(xpadded, filt);
  result  = result(length(filt):(length(x)+length(filt)-1)); // unpad the result
  result  = result(1:2:length(result)-1); // decimate (down sample) by 2
}

function FIR(x, filt) {
// handle elements that use ''negative'' time
  for i = 1 to (length(filt) - 1) {
    result(i)=0;
    for j = 1 to i {
      result(i)+=x(j)*filt(length(filt)-i+j);
    }
  }
// filter the rest of the input
  for i = length(filt) to length(x) {
    result(i)=0;
```

```
    for j = 1 to length(filt) {
      result(i)+=x(i-length(filt)+j)*filt(j);
    }
  }
}
```

3 What You Have to Do

The assignment will have four steps, and we would like you to inform us of your progress:

- Step 1 - FIR kernel and StreamC test program due Tuesday 4/23.
- Step 2 - Wavelet transform and StreamC test due Friday 4/26.
- Step 3 - Integrate the transform with a provided sort kernel.
- step 4 - Final write-up due Tuesday 4/30.

3.1 Step 0

Just preliminary setup operations:

1. Set up the Imagine environment as described in the Beginner's Guide.
2. Test the environment by compiling and running one of the demo applications.
3. Download the empty Imagine project from http://cva.stanford.edu/ee482c/downloads/hw1.zip, or copy it from /afs/ir.stanford.edu/class/ee482c/hw1.

3.2 Step 1

Begin by programming the FIR kernel in KernelC, and a simple StreamC program to test it. Think about how you would program a general FIR kernel (one that accepts filter of arbitrary lengths) but concentrate on optimizing two specific versions of it. One for a 24-tap filter and one for a 6-tap filter (the numbers are chosen to reflect the lengths of filters required for two types of wavelets). For simplicity we will be using floating-point only.

By Tuesday 4/23, we would like you to give us a short report on how you implemented the kernels, what optimizations you performed, and the scheduler output (how long is the main loop). Also, please provide us with your code by emailing it to mattan.erez@stanford.edu.

3.3 Step 2

Once the FIR kernels are written you should write a StreamC program that performs the Fast Wavelet Transform described in Section 2. Assume that you can fit the entire input signal, the resulting transform coefficients, and all temporary data in the SRF.

Also, remember that for the wavelet transform the input length must be a power of 2, and please assume that the CoarsestScale >= 3.

Run your application using the two wavelet QMFs and the input data that will be provided in /afs/ir.stanford.edu/class/ee482c/hw1.

By Friday 4/26 please send us your opinion on the performance of the application, including some results obtained by running the cycle accurate simulator (isim). As before make your code available as well.

3.4 Step 3

Now that you have a working wavelet transform you can complete the compression application by sorting the transform coefficients and selecting the n largest ones. Use the sorting kernel we will provide and output the result to a file. Then, test the compression algorithm by reconstructing the signal from the sparse coefficients in Matlab.

3.5 Step 4

Complete the homework by writing up your results and methods. The report should include the following:

- Description of the general FIR filter kernel and an evaluation of its suitability to Imagine.
- Description of the 24-tap optimized filter including the reasoning behind your algorithm choice, optimizations you performed, and scheduling results.
- Description of the 6-tap optimized filter including the reasoning behind your algorithm choice, optimizations you performed, and scheduling results.
- Analysis of the wavelet compression application. Describe the application, what problems you encountered, simulation results, and the characteristics of the application (compute-bound/memory-bound, where performance was lost, ...).
- Analysis of the compression results. How do the two wavelets compare to each other, and how does the reconstruction quality depend on the compression ratio (numbers of coefficients stored after sorting).
- Finally, describe how you would handle the case of a very long input that would not fit in the SRF.