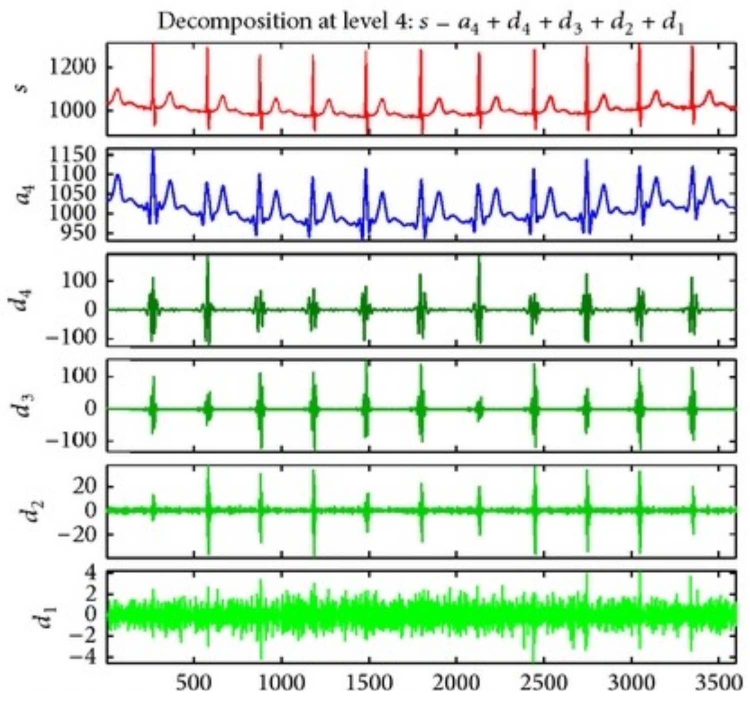
DISCRETE WAVELET TRANSFORM USING CHAPEL

WAVELET TRANSFORM
Background

A wavelet transform is a common signal processing technique used to decompose a signal into a set of sub-signals — each reflecting various degrees of detail from the original. This is useful in a variety of domains such as image compression (JPEG uses a discrete wavelet transform) or signal de-noising:



ECG Signal

Average Component of DWT

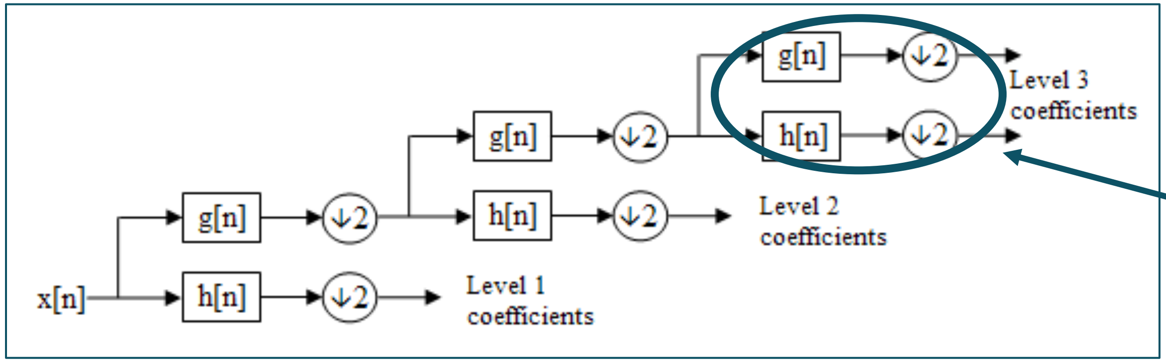
Detail Components of DWT

The highest detail-level is essentially all noise. By discarding this signal, and computing an inverse DWT, one can obtain a denoised version of the original signal.

https://www.researchgate.net/figure/Wavelet-denoising-for-ECG-record-no-103m-using-Biorthogonal-1D-wavelet_fig1_262698301

WAVELET TRANSFORM
Background

A DWT of a signal *x* is typically computed using a cascaded bank of filters and down-samplers:



https://en.wikipedia.org/wiki/Discrete_wavelet_transformhttps://en.wikipedia.org/wiki/Discrete_wavelet_transform

Note that each layer of filtering can be computed concurrently

A variety of filter types can be used for *g[n]* and *h[n]*. In this example, we'll use the two-element **HARR** wavelet, which is one of the simplest wavelet functions. These are their impulse responses:

$h[n] = [1, 1]$ (computes the local "average" of the signal)

$g[n] = [1, -1]$ (computes the "derivative" of the signal)

A filter *g* is applied to a signal *x* using a discrete convolution:

$$y[n] = (x * g)[n] = \sum_{m=-M}^{M} x(n-m)g[m]$$

where g is defined over {-M, ..., M}

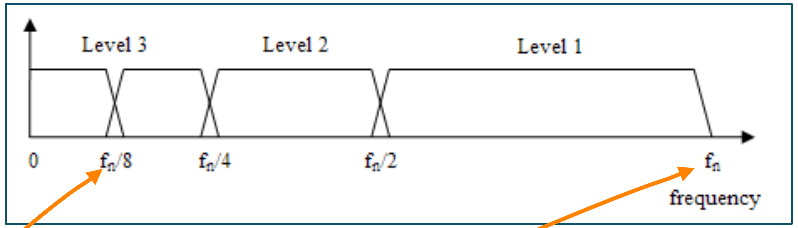
CHAPEL IMPLEMENTATION OF 1D DWT

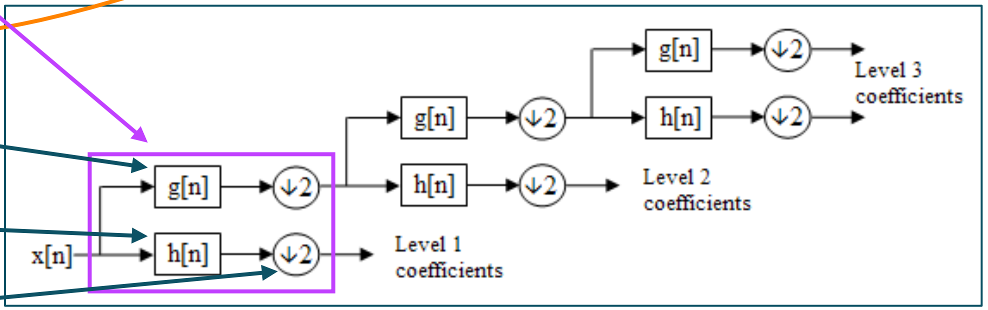
Signatures of each functions used in the 1D DWT

The following slides will show each procedure in more detail

```
// compute an n-level DWT of x
proc haarWavelet1D(x, n){ … }

// compute one step of the wavelet computation on the input signal
// this procedure relies on the following three procedures to complete each step
proc hwRec(signal, output, fmax, fstop){ … }

// convolution with the HAAR wavelet high-pass filter
proc haarHP(x) { … }

// convolution with the HAAR wavelet low-pass filter
proc haarLP(x) { … }

// down-sample by a factor of 2
proc downSample2(x) { … }
```

Output Array





The full program can be found here

WAVELET TRANSFORM

Implementation – Filter Convolution and Down-sampling

```
// convolution with the HAAR wavelet high-pass filter
proc haarHP(x) {
   var y : [x.domain] int = x;
   forall i in x.domain#(x.domain.size-1) do y[i] -= x[i+1];
   return y;
}

// convolution with the HAAR wavelet low-pass filter
proc haarLP(x) {
   var y : [d] int = x;
   forall i in x.domain#(x.domain.size-1) do y[i] += x[i+1];
   return y;
}

// down-sample by a factor of 2
proc downSample2(x) {
   var y : [{x.domain.first..(x.domain.last/2)}] int;
   y = x[x.domain by 2];
   return y;
}
```

WAVELET TRANSFORM

Implementation – Progressive Filtering using a Recursive Procedure

```
// recursive helper for wavelet computation
proc hwRec(signal, output, fmax, fstop) {
  // check termination condition
  if fmax == fstop {
    // store the final layer of high-pass coefficients
    output[{0..<fmax}] = signal;
  } else {
    cobegin {
      // compute and store the low-pass coefficients
      output[{fmax/2..<fmax}] = downSample2(haarLP(signal));

      // compute the high-pass coefficients and start the next layer of filtering
      hwRec(downSample2(haarHP(signal)), output, fmax/2, fstop);
    }
  }
}
```

WAVELET TRANSFORM
Implementation – Calling Recursive Procedure

```
// compute an n-level DWT of x
proc haarWavelet1D(x, n) {
   assert(2**log2(d.size) == d.size, "array size must be a power of 2");

   var output : [x.domain] int;
   hwRec(x, output, x.size, x.size / 2**n);
   return output;
}

// apply a 3-level DWT to a simple signal
var signal = [i in {0..<32}] (i-16)**2;
var wt = haarWavelet1D(signal, 3);

writeln(signal);
writeln(wt);
```

CHAPEL IMPLEMENTATION OF 1D DWT — WITH TYPE ANNOTATIONS

Updated signatures for each of the same procedures using *type annotations*, *type queries* and *formal intents*.

```
// compute an n-level DWT of x
proc haarWavelet1D(x: [?d] ?t, n): [d] t

// compute one step of the wavelet computation on the input signal
// this procedure relies on the following three procedures to complete each step
proc hwRec(const signal,ref output, fmax: int, fstop: int){ … }

// convolution with the HAAR wavelet high-pass filter
proc haarHP(x: [?d] ?t): [d] t { … }

// convolution with the HAAR wavelet low-pass filter
proc haarLP(x: [?d] ?t): [d] t { … }

// down-sample by a factor of 2
proc downSample2(x: [?d] ?t): [d] t { … }
```

The following slides will show each of these in more detail

It is important to note that these procedures will behave the same as the unannotated procedures on the previous three slides

The full program can be found here

WAVELET TRANSFORM

Implementation – Filter Convolution and Down-sampling

```
// convolution with the HAAR wavelet high-pass filter
proc haarHP(x: [?d] ?t): [d] t {
   var y : [d] t = x;
   forall i in d#d.size-1 do y[i] -= x[i+1];
   return y;
}

// convolution with the HAAR wavelet low-pass filter
proc haarLP(x: [?d] ?t): [d] t {
   var y : [d] t = x;
   forall i in d#d.size-1 do y[i] += x[i+1];
   return y;
}

// down-sample by a factor of 2
proc downSample2(x: [?d] ?t) {
   var y : [{d.first..(d.last/2)}] t;
   y = x[d by 2];
   return y;
}
```

The '[?d]' annotation on the arrays type, allows us to replace calls to 'x.domain' with 'd'

The '?t' annotation on the arrays type, allows us to make these procedures generic over any type that supports the `+=` operation.

The simpler implementation just uses 'int' to define the array 'y'

WAVELET TRANSFORM

Implementation – Progressive Filtering using a Recursive Procedure

```
// recursive helper for wavelet computation
proc hwRec(const signal, ref output, fmax: int, fstop: int) {
  // check termination condition
  if fmax == fstop {
    // store the final layer of high-pass coefficients
    output[{0..<fmax}] = signal;
  } else {
    cobegin {
      // compute and store the low-pass coefficients
      output[{fmax/2..<fmax}] = downSample2(haarLP(signal));

      // compute the high-pass coefficients and start the next layer of filtering
      hwRec(downSample2(haarHP(signal)), output, fmax/2, fstop);
    }
  }
}
```

The 'const' formal intent designates that this procedure cannot modify the signal array

The 'ref' formal intent designates that calls to this procedure should not make copies of the 'output' array

Note: Neither of the formal intents are technically necessary here — chapel arrays are always taken by reference by default. However, the annotations can be useful for documentation purposes

The types of 'fmax' and 'fstop' are designated as 'int' to clarify that these are discrete frequency indices, and not scalar frequencies

WAVELET TRANSFORM
Implementation – Calling Recursive Procedure

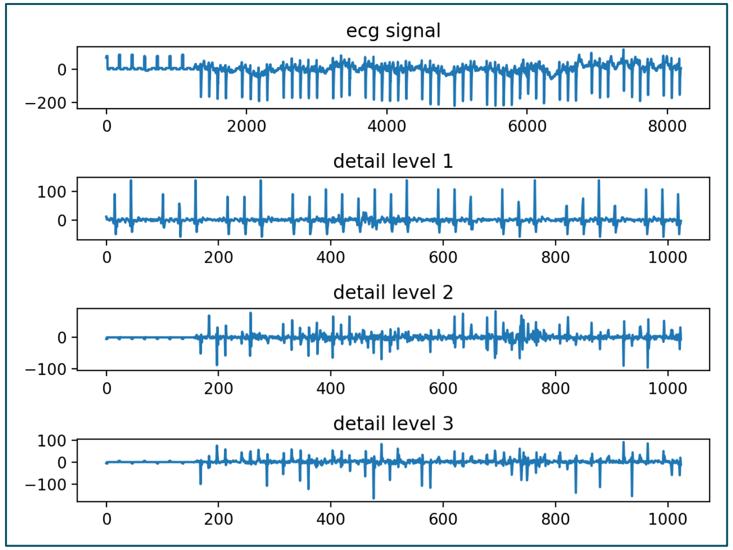
```
// compute an n-level DWT of x
proc haarWavelet1D(x: [?d] ?t, n: int): [d] t {
   assert(2**log2(d.size) == d.size, "array size must be a power of 2");

   var output : [d] t;
   hwRec(x, output, d.size, d.size / 2**n);
   return output;
}
```

The queries on 'x's domain and type can be used to define the type of the output array — also useful for documentation purposes

```
// apply a 3-level DWT to a simple signal
var signal = [i in {0..<32}] (i-16)**2;
var wt = haarWavelet1D(signal, 3);

writeln(signal);
writeln(wt);
```

ECG EXAMPLE
Using code from previous slides on some real data

- Looking at this example: https://github.com/jeremiah-corrado/chpl_dwt_example
  - Compile with: '*chpl wavelet.chpl --fast*'
  - Run with: '*./wavelet*'
- You should see an output like this in the results folder:



- Run again with: '*./wavelet --nLevels=5*' to see different results