# Compiling Telescoping Languages: A Study of Some Matlab Codes

***Arun Chauhan***

and

***Ken Kennedy***

## Motivation

- End−users prefer a simpler and "high−level" programming language
  - Should be close to mathematical level
- High Performance programming is hard
  - Shortage of programmers
  - Problem of portability
- Accessible High Performance Programming

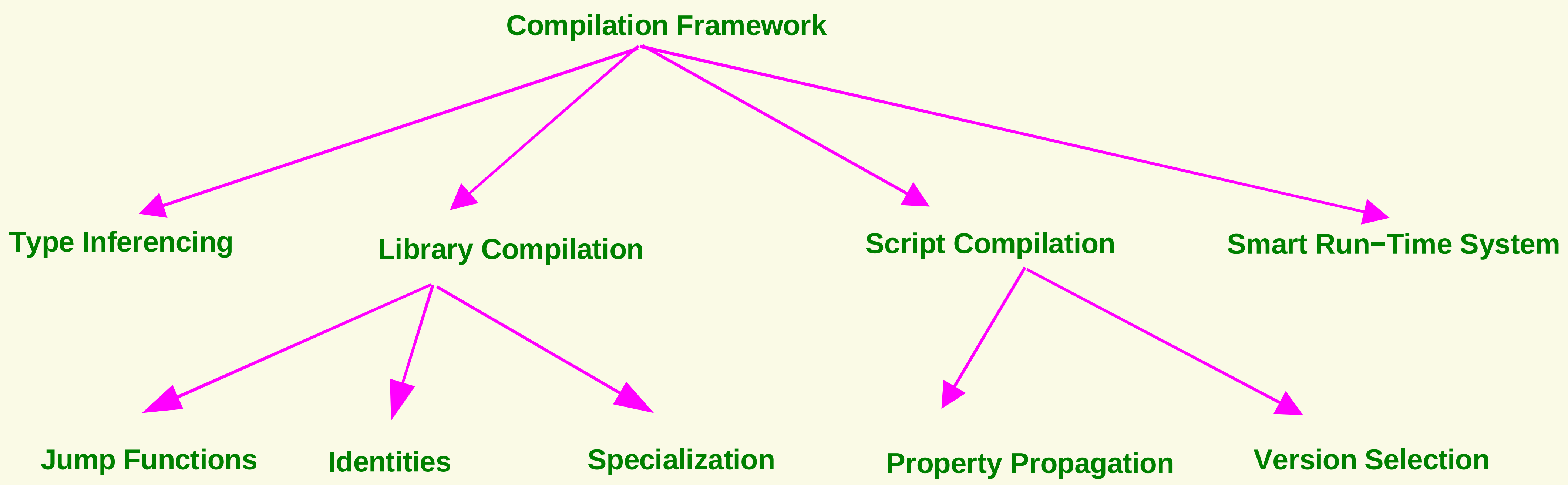
## Current Scenario

- Object Oriented Languages
  - Targetted towards professionals
  - Still not sufficiently high−level for end−users
- Functional Programming Languages
  - Suffer from performance problems
- Scripting Languages (e.g., Matlab)
  - Preferred and used by end−users
  - Have domain specific libraries
  - But, no fast and effective compilers

# Focus: Library and Scripts

- Library Compilation
  - Summarizing inter−procedural properties (inter−procedural jump functions)
  - Library identities (specified by library writer)
  - Code specialization
- Script Compilation
  - Variable property propagation
  - Loading and linking (version selection)
- Smart Run−Time System
  - Dynamic compilation

# Compiling Telescoping Languages



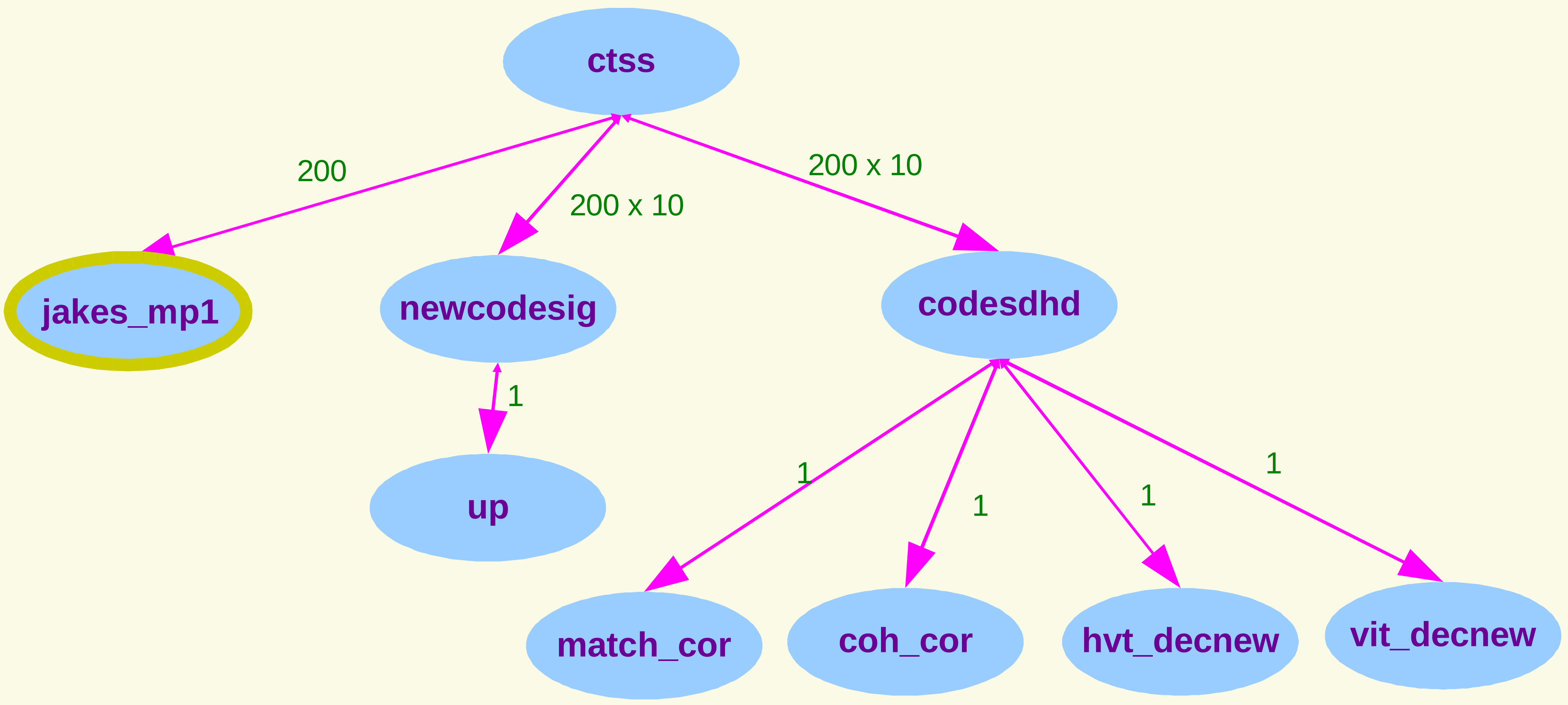
# Example Codes

- Real codes used by research groups in ECE
- Long Running codes, potential for optimization
- Written in Matlab (even though slow)
  - ctss – simulation of overalpped convolution coding
  - sar  – image processing code
  - star – another image processing code
- Parts of the codes re–used extensively (candidates for domain–specific lib routines)

# ctss main

```
<initialization>;

for ii = 1:200
  chan = jakes_mp1 (16500, 160, ii, num_paths);

  for k = 1:num_paths
    chan(k,:) = sqrt(sig_pow_paths(k)) * chan(k,:);
  end

  err1 = [];
  err2 = [];
  err3 = [];

  for snr = 2:2:20
    M = 15;
    NO = 1000;
    l = 511;

    [s,x,ci,h,L,a,y,n0] = newcodesig (NO, l, num_paths, M, snr, chan, sig_pow_paths);

    B = 0;
    Tm = 2;
    Bd = 3;
    [o1,d1,d2,d3,mf,m]= codesdhd (y, a, h, NO, Tm, Bd, M, B, n0);

    Nbits = NO/2;
    ee = ci(1:M:M*NO) - d1;
    err1 = [err1 length(find(ee~=0))];
    ee = (2*x - 1) - d2;
    err2 = [err2 length(find(ee(1:Nbits - 10)~=0))];
    ee = (2*x - 1) - d3;
    err3 = [err3 length(find(ee(1:Nbits - 10)~=0))];
  end

  noc511mpsd100_2(ii,:) = err1;
  c511mphd100_2(ii,:) = err2;
  c511mpsd100_2(ii,:) = err3;
end

<saving and printing>;
```

# ctss: Call Graph



# ctss: jakes_mp1

```
function z = jakes_mp1 (blength, speed, bnumber, N_Paths)

freq = 1800;                   % centre frequency, MHz
mob_speed = speed/3.6;         % mobile speed, m/s
t_step = 0.0000033*31/15;      % sample period, s
Num = blength;                 % number of symbols
N = 34;                        % number of effective cosines added
Num_osc = 8;                   % 0.5*(N/2 - 1)
wave_len = 300 / freq;
omega = 2 * pi * mob_speed / wave_len;

z = zeros(N_Paths,Num);

for k = 1:N_Paths

  j = [1:Num]' + blength * ((bnumber + (k-1)*1000) - 1) ;
  n = [1:Num_osc]';

  jp = j(:,ones(1,length(n)))';
  np = n(:,ones(1,length(j)));

  xc = sqrt(2)*cos(omega*t_step*j') ...
       + 2*sum(cos(pi*np/Num_osc).*cos(omega*cos(2*pi*np/N)*t_step.*jp));
  xs = 2*sum(sin(pi*np/Num_osc).*cos(omega*cos(2*pi*np/N)*t_step.*jp));

  % for j = 1 : Num
  %    if j/1000 == round(j/1000)
  %       disp(j)
  %    end
  %    xc(j) = sqrt(2) * cos (omega * t_step * j);
  %    xs(j) = 0;
  %    for n = 1 : Num_osc
  %        cosine = cos(omega * cos(2 * pi * n / N) * t_step * j);
  %        xc(j) = xc(j) + 2 * cos(pi * n / Num_osc) * cosine;
  %        xs(j) = xs(j) + 2 * sin(pi * n / Num_osc) * cosine;
  %    end
  % end

  z(k,:) = sqrt(17) \ (xc + sqrt(-1) * xs);     % normalized complex response

end
```

# ctss: jakes_mp1 – possibilities

```
function z = jakes_mp1 (blength, speed, bnumber, N_Paths)

<scalar initialization>;

for k = 1:N_Paths
  j = [1:Num]' + blength * ((bnumber + (k-1)*1000) - 1) ;
  n = [1:Num_osc]';

  jp = j(:,ones(1,length(n)))';
  np = n(:,ones(1,length(j)));

  xc = sqrt(2)*cos(omega*t_step*j') ...
       + 2*sum(cos(pi*np/Num_osc).*cos(omega*cos(2*pi*np/N)*t_step.*jp));
  xs = 2*sum(sin(pi*np/Num_osc).*cos(omega*cos(2*pi*np/N)*t_step.*jp));

  z(k,:) = sqrt(17) \ (xc + sqrt(-1) * xs);    % normalized complex response
end
```

- Vectorization
- CSE
- Identities
- Strength Reduction

# ctss: jakes_mp1 transformed

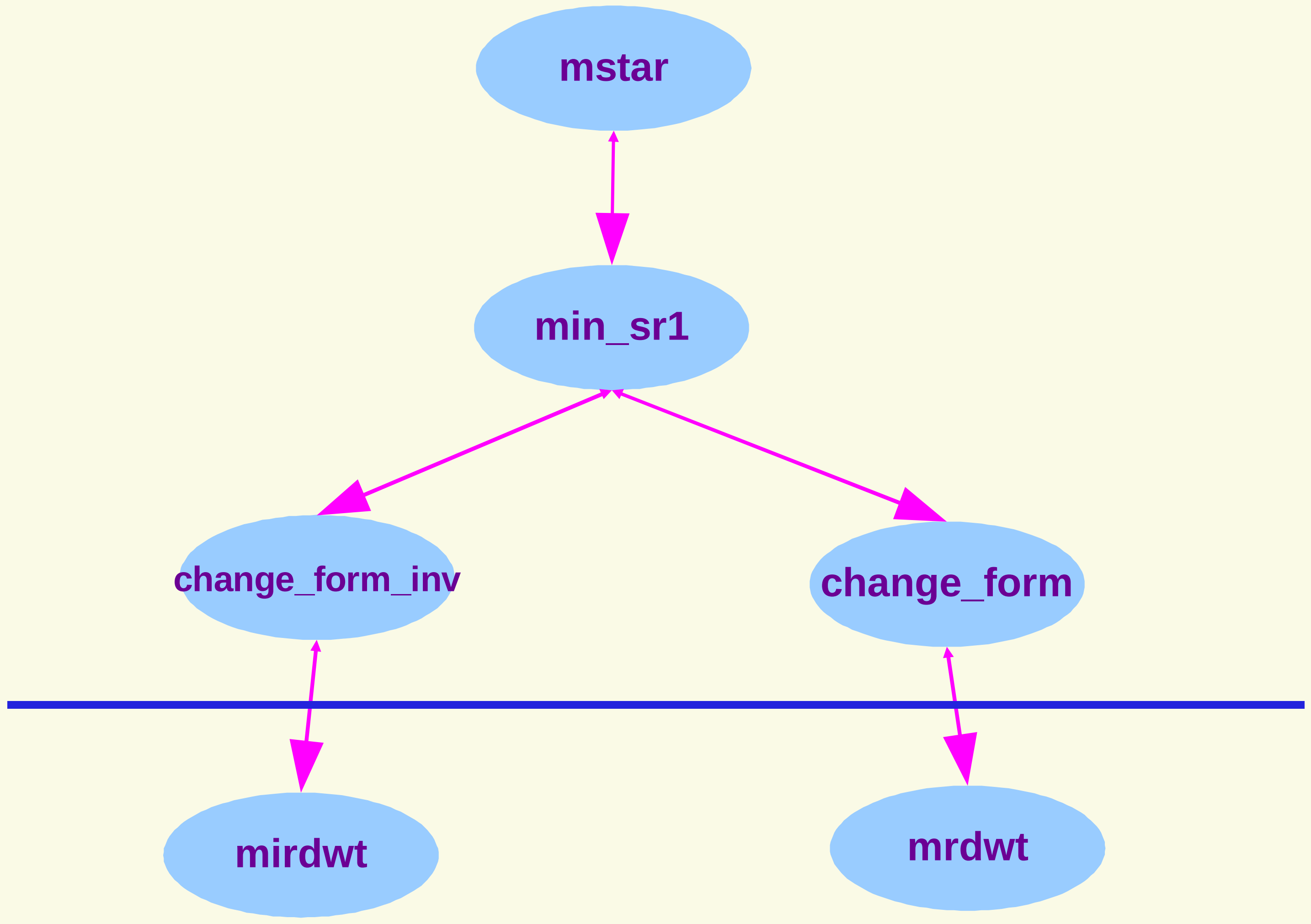
```
function z = jakes_mp1_init (blength, speed, N_Paths)

<scalar initialization>;
n = [1:Num_osc]';
np = n(:,ones(1,Num));
C1 = sqrt(2);
C2 = omega*t_step;
C3 = cos(pi*np/Num_osc);
C4 = omega*cos(2*pi*np/N)*t_step;
C5 = sin(pi*np/Num_osc);
C6 = sqrt(17);
```

```
function z = jakes_mp1_delta (bnumber)

for k = 1:N_Paths
  j = [1:Num]' + blength * ((bnumber + (k-1)*1000) - 1) ;
  jp = j(:,ones(1,length(n)))';
  xc = C1*cos(C2*j') + 2*sum(C3.*cos(C4.*jp));
  xs = 2*sum(C5.*cos(C4.*jp));
  z(k,:) = C6 \ (xc + sqrt(-1) * xs); % normalized complex response
end
```

# mstar

```
clear;
L = 6;
[data1,map] = tiffread('brick.tiff');
t2 = [-70:.5:70];
data1 = real(kron(x3(1:256),ones(256,1)));
[h,g,h1,g1] = wfilters('haar');
[nx ny] = size(data1);
keyboard
blk = 2^L;
s0 = zeros(floor(nx/blk),floor(ny/blk),L,3);
alpha = 1;

for i = 1 : floor(nx/blk)
    for j = 1 : floor(ny/blk)
        data = data1((i-1)*blk+1 : i*blk,(j-1)*blk+1 : j*blk);
        [s,r,j_hist] = min_sr1 (data, h, L, alpha);
        s0(i,j,:,:) = reshape (s, L, 3);
    end
end

ss = squeeze(mean(mean(s0)));
ss = ss/norm(ss,'fro');
save highsinc_proj_reshape ss
ss = squeeze(mean(mean(abs(s0))));
ss = ss/norm(ss,'fro');
save highsinc_proj_reshape_abs ss
```

# mstar: Call Graph



# mstar: min_sr1

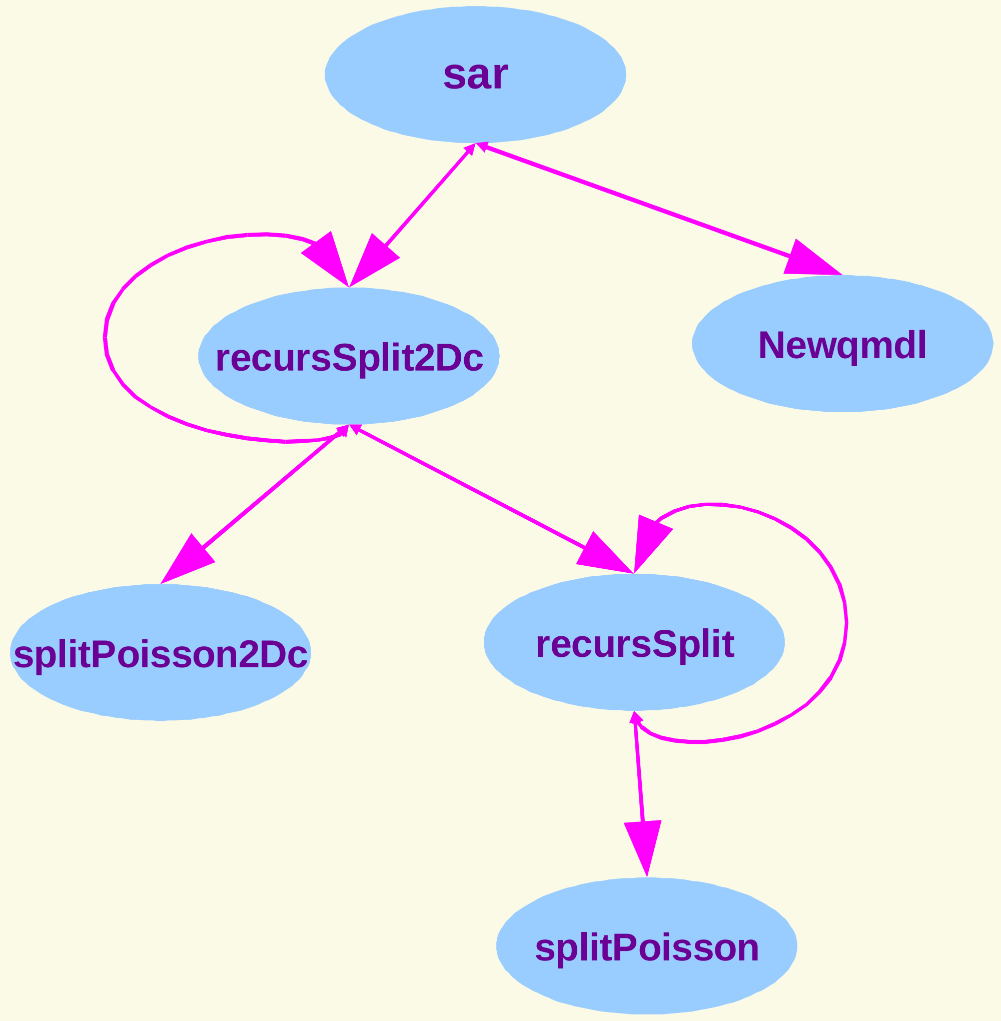
```
function [s, r, j_hist] = min_sr1 (xt, h, m, alpha)
...
while ~ok
    ...
    invsr = change_form_inv (sr0, h, m, low_rp);
    big_f = change_form (xt-invsr, h, m);
    ...
    while iter_s < 3*m
        ...
        invdr0 = change_form_inv (sr0, h, m, low_rp);
        sssdr =  change_form (invdr0, h, m);
        ...
    end
    ...
    invs r = change_form_inv (sr0, h, m, low_rp);
    big_f  = change_form (xt-invsr, h, m);
    ...
    while iter_r < n1*n2
        ...
        invdr0 = change_form_inv (sr0, h, m, low_rp);
        sssdr  = change_form (invdr0, h, m);
        ...
    end
    ...
end
```

# mstar: change_form*

```
% This is a function that changes the format of a 2−D matrix to 4−D redundant DWT matrix by a special stacking operation,
% and then computes the corresponding 2−D image.

function x = change_form_inv (data, h, L, yl);
[n1,n2] = size(yl);
data = reshape (data, L, 3, n1, n2);
yh = zeros(n1,3*L*n2);
for k = 1:L
   for l = 1:3
      yh(:, 3*(k−1)*n2+(l−1)*n2+1 : 3*(k−1)*n2+l*n2) = squeeze (data(k,l,:,:));
   end;
end;
x = mirdwt (yl, yh, h, L);
```

```
% This is a function that computes the redundant DWT of the iinput 2−D matrix, and then changes the format of the
% resulting 4−D DWT matrix to a 2−D matrix by a special stacking operation.

function x = change_form (data, h, L);
[n1,n2] = size(data);
[yl,yh,L] = mrdwt (data, h, L);
x = zeros(L,3,n1,n2);
for k = 1:L
   for l = 1:3
      x(k,l,:,:) = yh(:, 3*(k−1)*n2+(l−1)*n2+1 : 3*(k−1)*n2+l*n2);
   end;
end;
x = reshape (x, 3*L, n1*n2);
```

# sar


sar
recursSplit2Dc
Newqmdl
splitPoisson2Dc
recursSplit
splitPoisson

# sar: recursSplit2Dc

```
function y = recursSplit2Dc(x)

[Nx,Ny] = size(x);
if (Nx==0)|(Ny==0)
   y = [];
else
   if (Nx==1)&(Ny==1)
      y = mean(mean(x))*ones(Nx,Ny);
   else
      if (Nx~=1)&(Ny~=1)
          k = NewsplitPoisson2Dc(x);
          if (k(2)==0)&(k(1)==0)
             y = mean(mean(x))*ones(Nx,Ny);
             %disp('bottom 2D')
          else
            y(1:k(1),1:k(2)) = recursSplit2Dc(x(1:k(1),1:k(2)));
            y(1+k(1):Nx,1:k(2)) = recursSplit2Dc(x(1+k(1):Nx,1:k(2)));
            y(1:k(1),1+k(2):Ny) = recursSplit2Dc(x(1:k(1),1+k(2):Ny));
            y(1+k(1):Nx,1+k(2):Ny) = recursSplit2Dc(x(1+k(1):Nx,1+k(2):Ny));
          end
      else
          y = recursSplit(x);
      end
   end
end
```

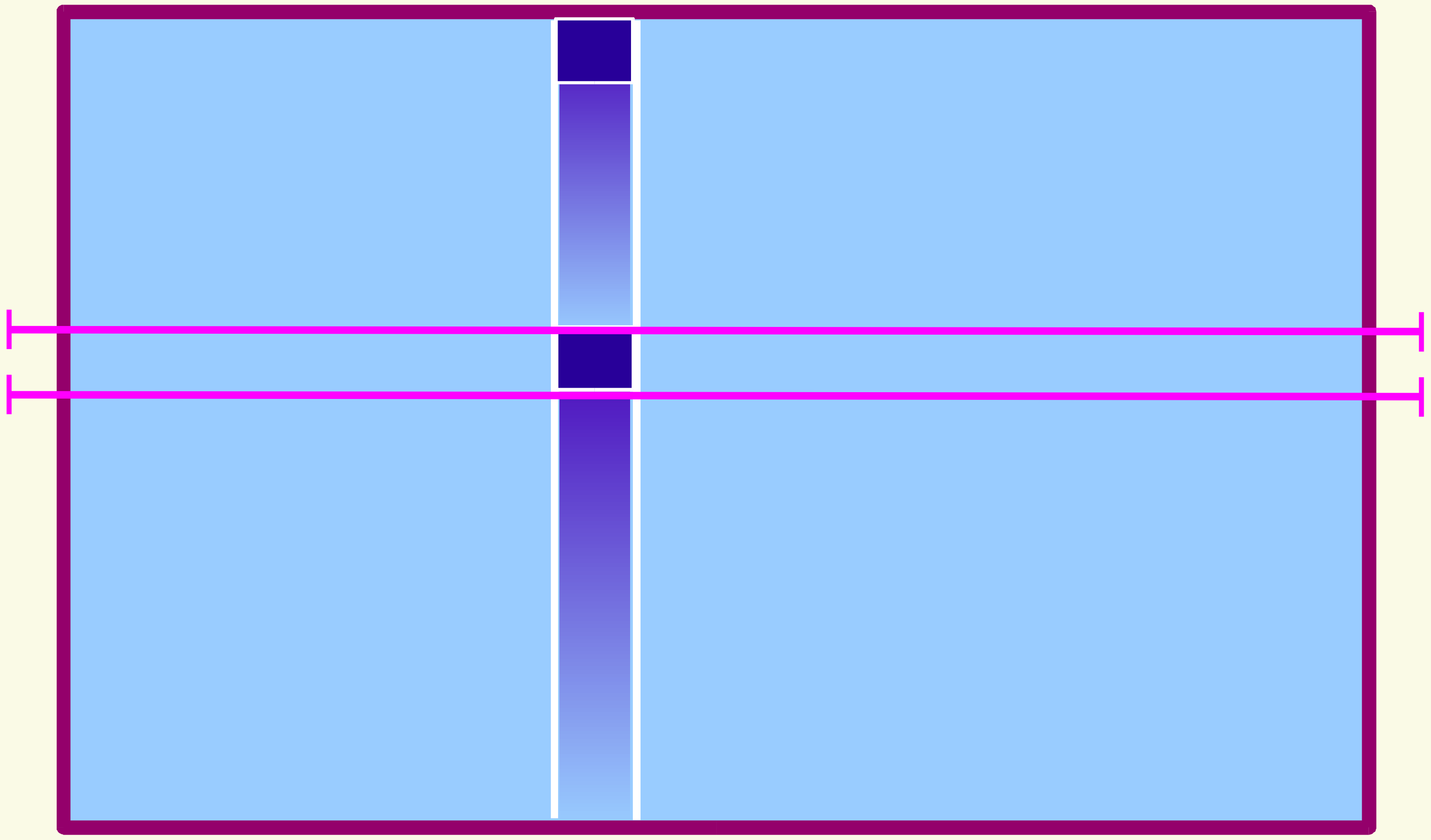
# sar: splitPoisson2Dc

```
function k = NewsplitPoisson2Dc(x)

[Nx Ny] = size(x);
total = sum(sum(x));
l_0 =  total * log(Nx*Ny);

for kx = 1:Nx-1
  p1 = sum(x(1:kx,:),1);
  p2 = sum(x(kx+1:Nx,:),1);
  s1 = cumsum(p1);
  s2 = cumsum(p2);
  ts1 = s1(Ny);
  s1 = s1(1:Ny-1);
  ts2 = s2(Ny);
  s2 = s2(1:Ny-1);
  l(kx,:) = - s1 .* log(eps+(s1./(total*kx*[1:Ny-1]))) ...
        -(ts1-s1) .* log(eps+((ts1-s1)./(total*kx*[Ny-1:-1:1]))) ...
        - s2 .* log(eps+(s2./(total*(Nx-kx)*[1:Ny-1])))  ...
        -(ts2-s2) .* log(eps+((ts2-s2)./(total*(Nx-kx)*[Ny-1:-1:1]))) ...
        + log(eps+ts1+1) + log(eps+total-ts1+1);
end
...
```

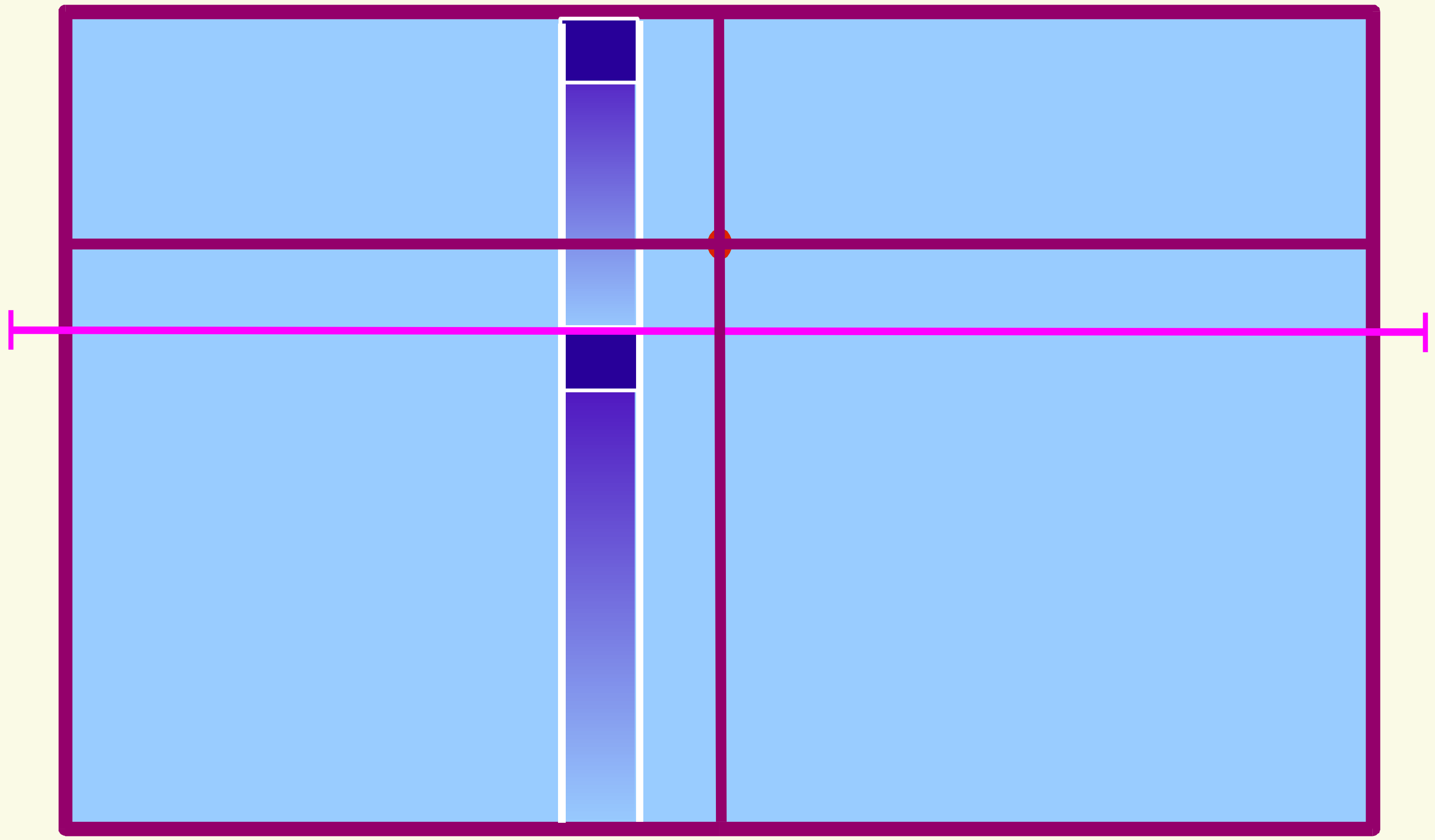
# sar: working of splitPoisson2Dc

Cumulative
Cumulative

## sar: working of splitPossion2Dc (contd)



# sar: working of splitPossion2Dc (contd)



**Dynamic Programming Techniques**

## Lessons

- Constant Propagation
- Vectorization
- Reduction in Strength applied to functions
- Library identities
- Array re−shaping (APL style?)
- Other traditional optimization techniques
  - common subexpression elimination
  - code motion
- Dynamic programming techniques (?)