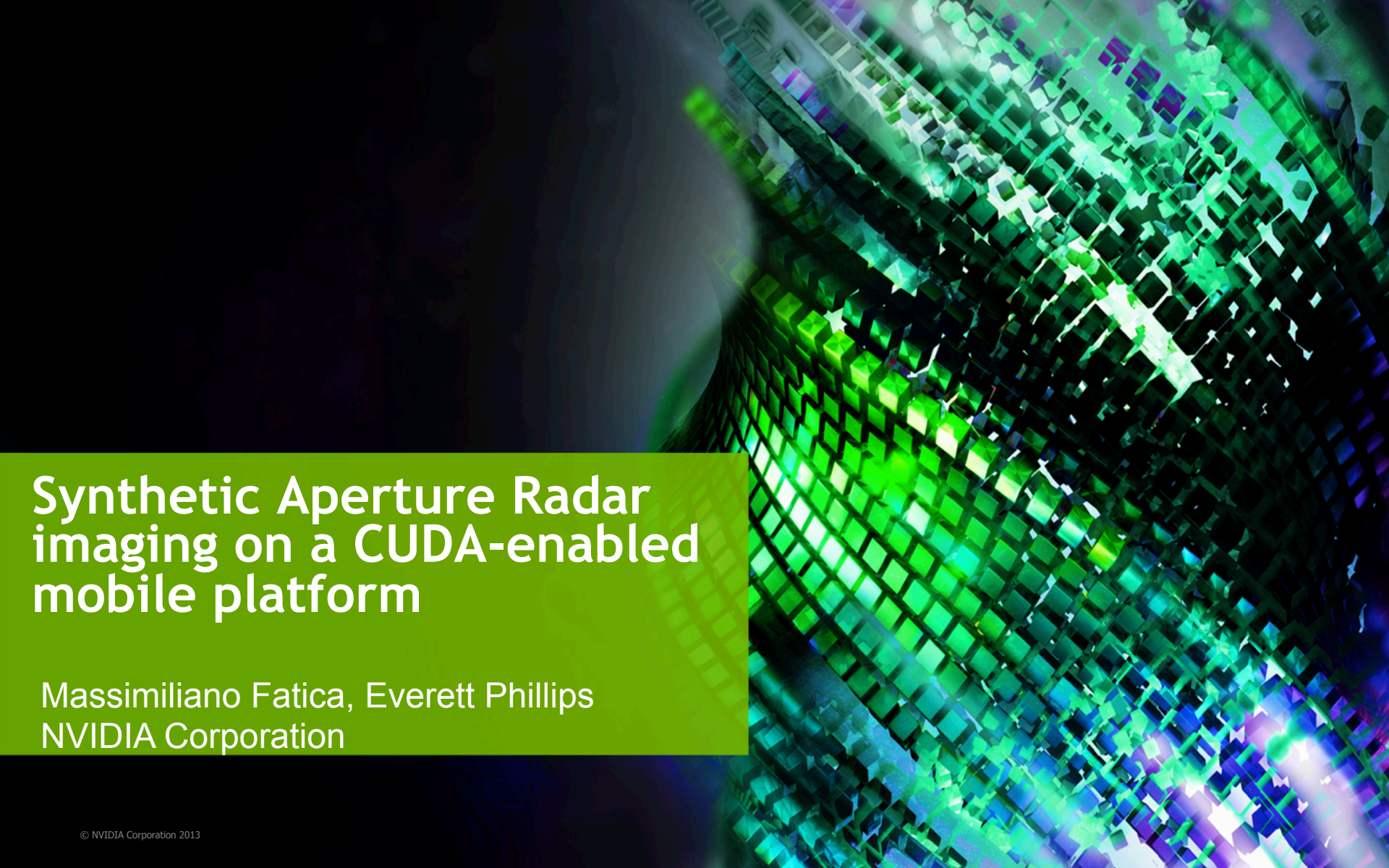


Synthetic Aperture Radar imaging on a CUDA-enabled mobile platform

Massimiliano Fatica, Everett Phillips
NVIDIA Corporation

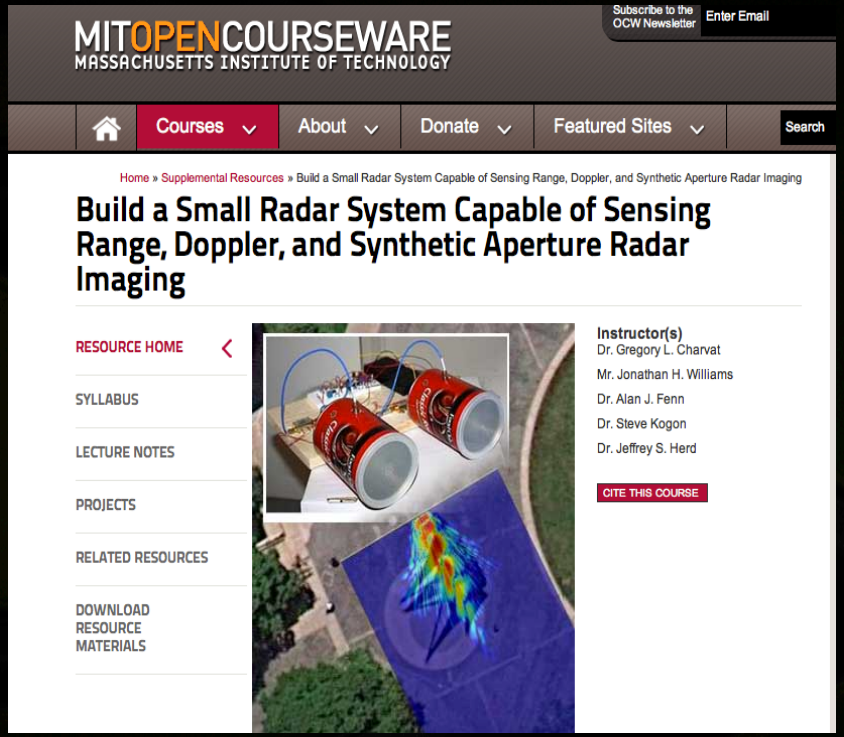
Outline

- Motivation and objective
- GPU computing
- SAR implementation and results
- Conclusions

Motivation

- GPU computing is now a reality in High Performance and Technical Computing
- CUDA capable platforms have been used in the embedded space (SAS, SAR, image processing)
- New CUDA capable SOC opens new opportunities in the embedded space: low power, low weight, battery powered

Objective

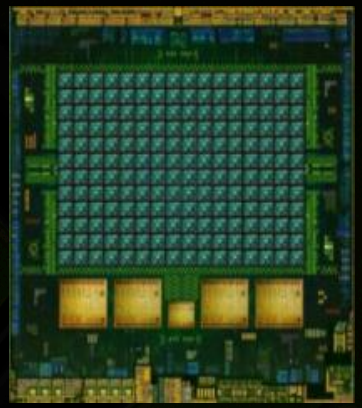


Assess the capabilities of a CUDA enabled embedded platform for Synthetic Aperture Radar porting the code from the MIT course:

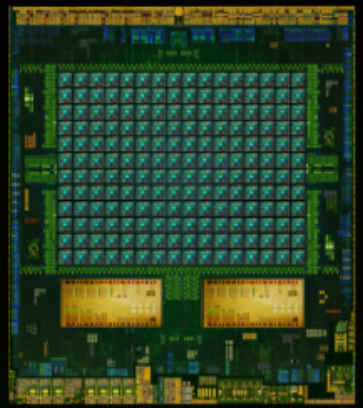
- Raw data, MATLAB scripts for preprocessing and image generation
- Use Octave + CUDA via mex files.

TEGRA K1

One Chip — Two Versions, First CUDA capable ARM SOC

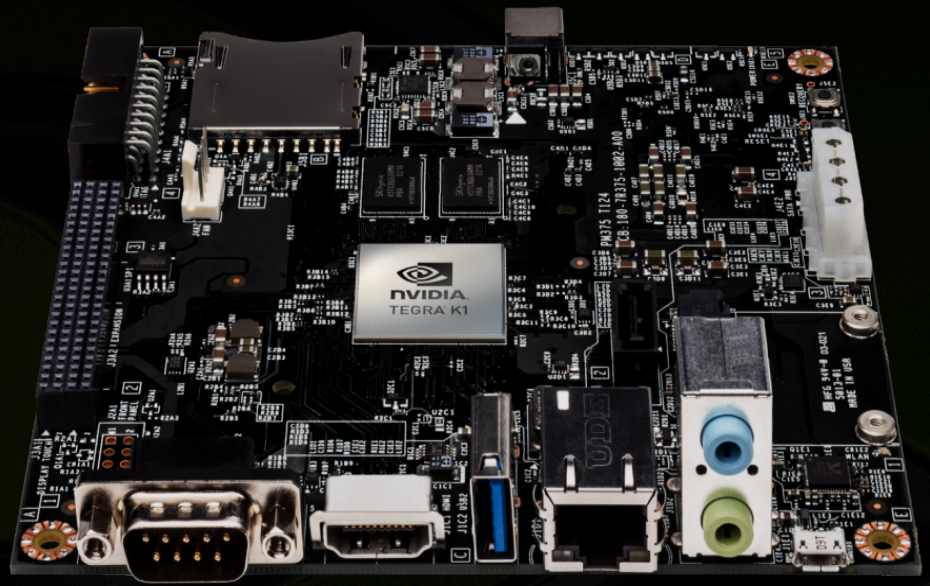


Pin Compatible



Quad A15 CPUs	Dual Denver CPUs
32-bit	64-bit
3-way Superscalar	7-way Superscalar
Up to 2.3GHz	Up to 2.5GHz
192 Kepler GPU cores	192 Kepler GPU cores

JETSON TK1 Developer board



- Tegra K1 32 bit
- 2 GB
- USB 3.0, GigE, GPIO, mini Pci-e, dual MIPI CSI-2 camera interfaces, SATA
- Hardware H.264 encoding and decoding
- Typical power consumption below 11W
- Developer friendly Ubuntu Linux software environment
- CUDA 6.0 toolkit and libraries (FFT, BLAS,NPP, Thrust, OpenCV)

CUDA & MATLAB (2014)

Several ways to use GPUs:

- GPU-enabled MATLAB functions in several toolboxes:

```
A = gpuArray(rand(2^16,1));
B = fft(A);
```

- CUDA kernel integration in MATLAB applications

- MEX interface *(this is the only path available for Octave)*

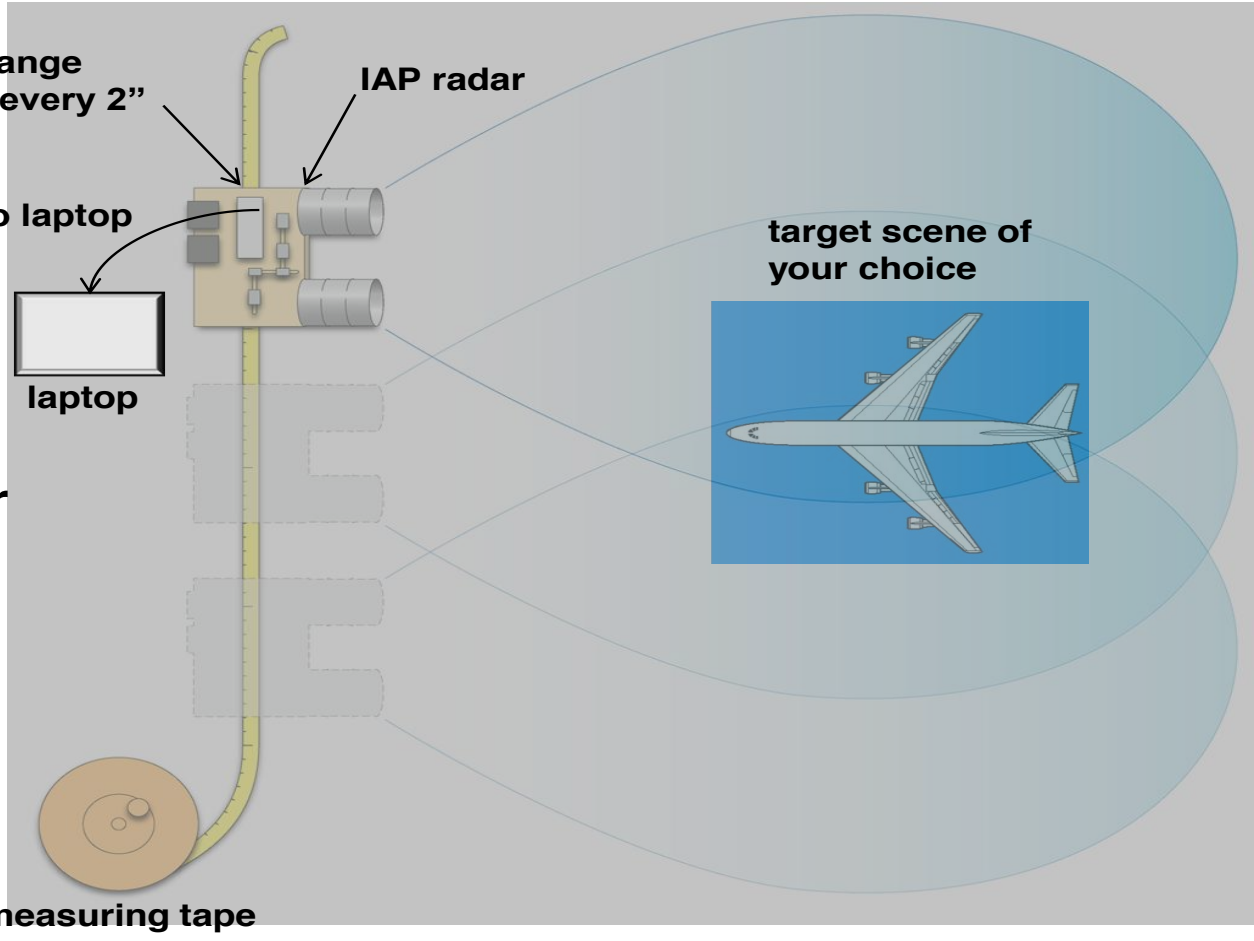
Mex files for CUDA/Octave

A typical mex file will perform the following steps:

1. Allocate memory on the GPU
2. Transfer the data from the host to the GPU
3. Rearrange the data layout for complex data if needed
4. Perform computation on GPU (library, custom code)
5. Rearrange the data layout for complex data
6. Transfer results from the GPU to the host
7. Clean up memory and return results to MATLAB

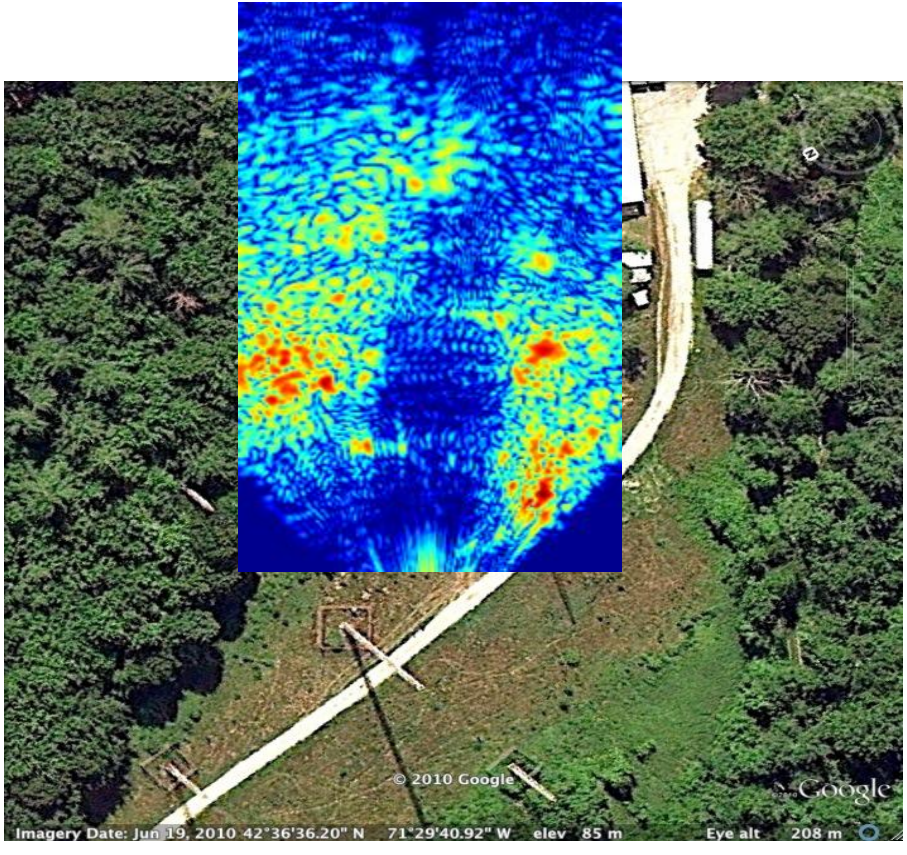
SAR

IAP SAR Geometry and Processing

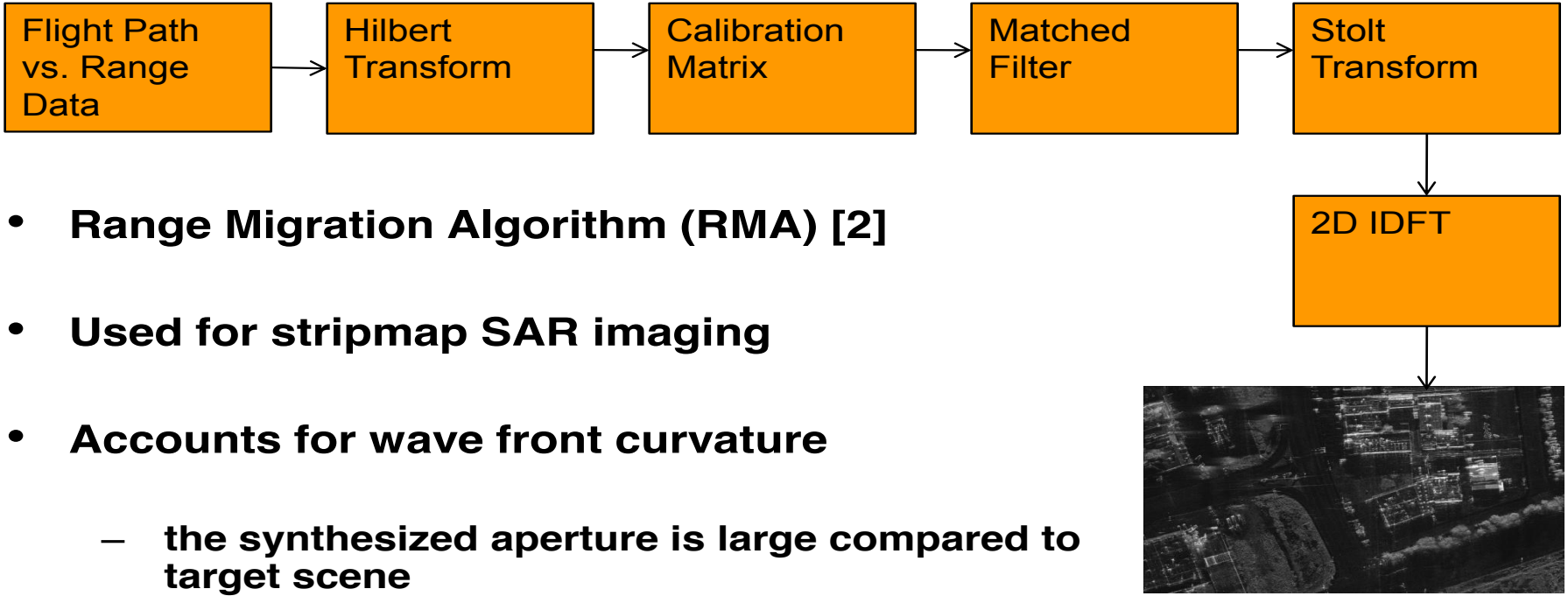


- implement rail SAR by manually moving radar down straight path
- record range profiles incrementally every 2”
- process with SAR_image.m

Example: SAR image of Back of Warehouse using IAP '11 Radar



Real-Time Imaging SAR Algorithm



- **Range Migration Algorithm (RMA) [2]**
- **Used for stripmap SAR imaging**
- **Accounts for wave front curvature**
  - **the synthesized aperture is large compared to target scene**

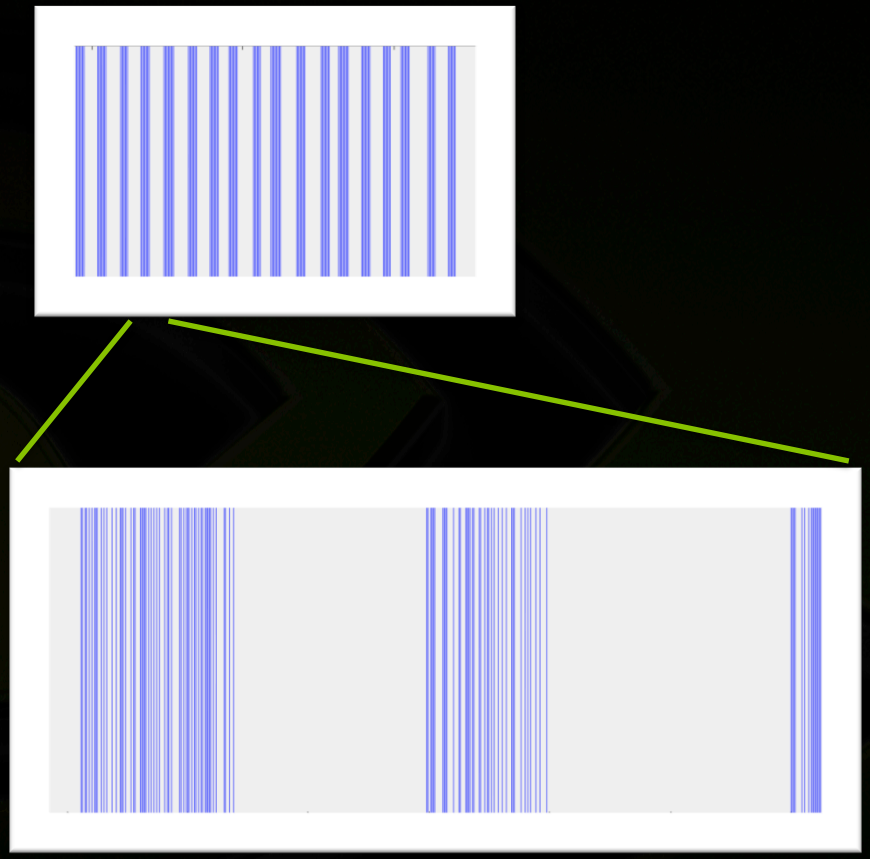
Pre-processing of the raw data

- Reading the data: the original data is in 16bit format, the left and right channels are stored in a sound file in .wav format. Test case contains 15,570,944 samples.

- Parse data by position: identify the start of the pulses looking for silence in the data

- Parse data by pulse: after locating the rising edge of the sync pulses, accumulate them and apply a Hilbert transform

Looking for silence

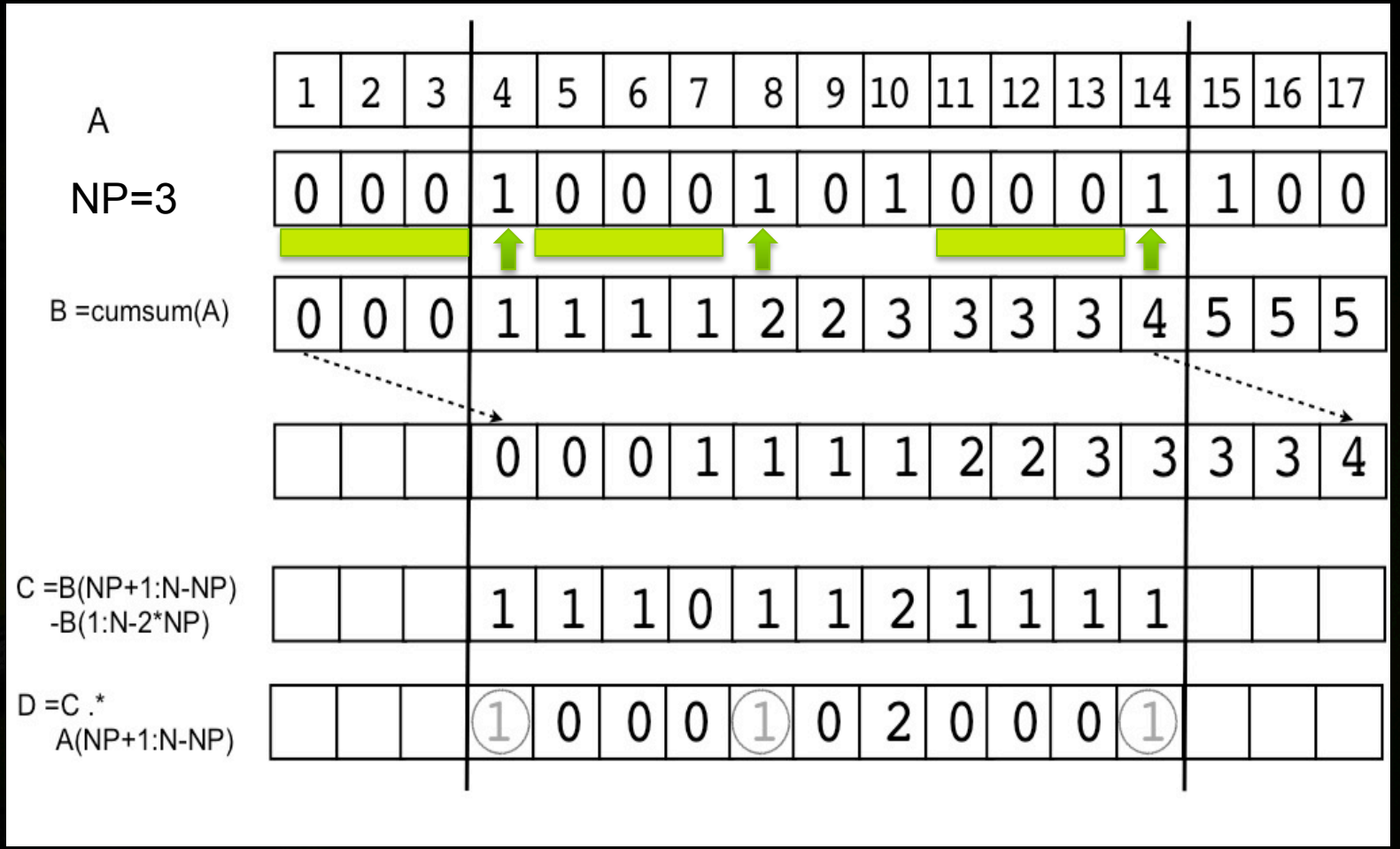


```
%parse data here by position
%(silence between recorded data)
rpstart = abs(trig)>mean(abs(trig));
count = 0;
Nrp = Trp*FS;
%min # samples between range profiles
for ii = Nrp+1:size(rpstart,1)-Nrp
 if rpstart(ii)==1 & sum(rpstart(ii-Nrp:ii-1))==0
   count = count + 1;
   RP(count,:)     =    s(ii:ii+Nrp-1);
   RPtrig(count,:) = trig(ii:ii+Nrp-1);
 end
```

For the pulse time and frequency used
In the data acquisition, Nrp=11025

Processing time 887s

Find voids in parallel



Looking for silence (CPU only)

```
%parse data  by position
%(silence between recorded data)
rpstart = abs(trig)>mean(abs(trig));
count = 0;
Nrp = Trp*FS;
%min # samples between range profiles
for ii = Nrp+1:size(rpstart,1)-Nrp
 if rpstart(ii)==1 & sum(rpstart(ii-Nrp:ii-1))==0
   count = count + 1;
   RP(count,:)     =     s(ii:ii+Nrp-1);
   RPtrig(count,:) = trig(ii:ii+Nrp-1);
 end
```

Processing time 887s

```
%parse data  by position in parallel
%(silence between recorded data)
rpstart = abs(trig)>mean(abs(trig));
Nrp = Trp*FS;
psum=cumsum(rpstart);
dis=psum(Nrp+1:size(rpstart,1)-Nrp)
     -psum(1    :size(rpstart,1)-2*Nrp);
dis=dis.*rpstart(Nrp+1:size(rpstart,1)-Nrp);
ind=find(dis==1);

for ii = 1:size(ind)
    istart=ind(ii)+Nrp;
    iend  =istart+Nrp-1;
    RP(ii,:)     =     s(istart:iend);
    RPtrig(ii,:) = trig(istart:iend);
end
```

Processing time 0.63s

Looking for silence on GPU

```
/* Compute the mean */
 trig_s_kernel<<<16,128>>>(buf,trig,s,avg,f);

/* Compute rpstart */
 rpstart_kernel<<<16,128>>>(trig,avg,rpstart,f);

/* Compute cumsum with Thrust */
 thrust::inclusive_scan(dp_rpstart,dp_rpstart+f,dp_psum);

/* Compute final value of array */
 ind_kernel<<<16,128>>>(rpstart,psum,ind,count_h,f,sr/4);

/* Select elements equal to 1  with Thrust */
 thrust::copy_if(dp_ind,dp_ind+f,dp_ind,is_pos());
```

Processing time 0.1s

Parse data by pulse

```
%parse data by pulse
thresh = 0.08;
for jj = 1:size(RP,1)
   %clear SIF;
   SIF = zeros(N,1);
   start = (RPtrig(jj,:)> thresh);
   count = 0;
   for ii = 12:(size(start,2)-2*N)
      [Y I] =  max(RPtrig(jj,ii:ii+2*N));
      if mean(start(ii-10:ii-2)) == 0 & I == 1
         count = count + 1;
         SIF = RP(jj,ii:ii+N-1)' + SIF;
      end
   end
   %hilbert transform
   q = ifft(SIF/count);
   sif(jj,:) = fft(q(size(q,1)/2+1:size(q,1)));
end
```

Processing time 162s

```
%parse data by pulse
thresh = 0.08;
for jj = 1:size(RP,1)
  %clear SIF;
   SIF = zeros(N,1);
   start = (RPtrig(jj,:)> thresh);
   psum=cumsum(start(1:Nrp-2*N));
   dis=1+psum(10:Nrp-2*N-2)-psum(1:Nrp-2*N-11);
   dis=dis.*start(12:Nrp-2*N);
   ind=find(dis==1)+11;
   count = 0;
   for ii=1:size(ind,2)
     myindex=ind(ii);
      [Y I] =  max(RPtrig(jj,myindex:myindex+2*N));
      if I ==1
         count = count + 1;
         SIF = RP(jj,myindex:myindex+N-1)' + SIF;
      end
   end
   %hilbert transform
   q = ifft(SIF/count);
   sif(jj,:) = fft(q(size(q,1)/2+1:size(q,1)));
end
```

Processing time 0.165s

Generating SAR image

1) Apply windowing function, pad the data, apply 1D FFT and FFTSHIFT: the original input array of size (55x441) is transformed in an array of size (2048x441).

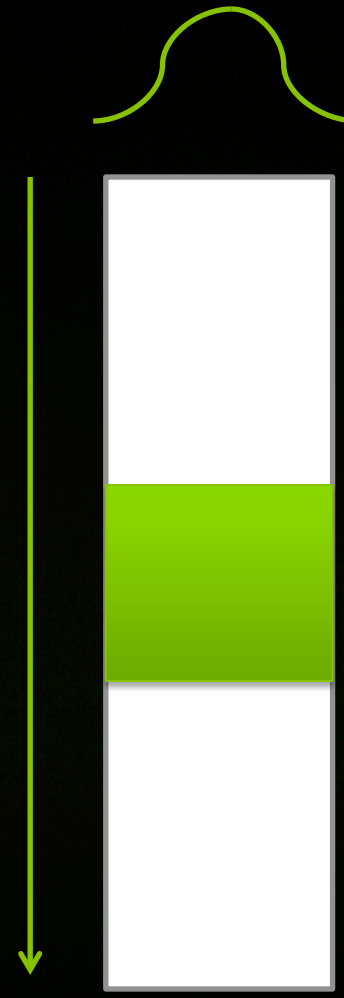
2) Apply matched filter

3) Perform Stolt interpolation: correct for range curvature. At the end of this phase the array is of dimension (2048x1024). Additional windowing function is applied to clean up the data

4) Pad the array to (8192x4096) and apply inverse 2D FFT.

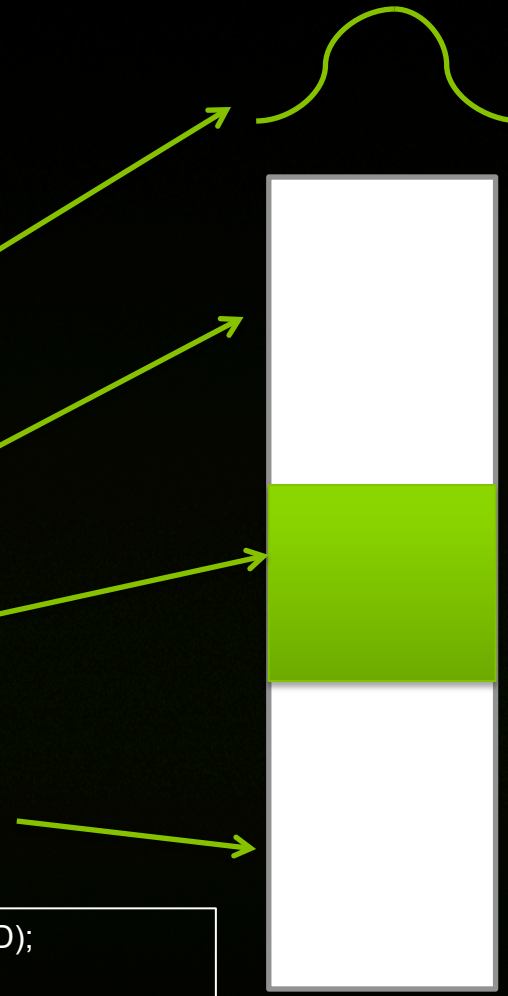
Phase 1

```
%apply Hanning window to data first
N = size(sif,2);
for ii = 1:N
   H(ii) = 0.5 + 0.5*cos(2*pi*(ii-N/2)/N);
end
for ii = 1:size(sif,1)
   sif_h(ii,:) = sif(ii,:).*H;
end
sif = sif_h;

zpad = 2048; %cross range symmetrical zero pad
szeros = zeros(zpad, size(sif,2));
for ii = 1:size(sif,2)
   index = round((zpad - size(sif,1))/2);
   szeros(index+1:(index + size(sif,1)),ii) = sif(:,ii); end
sif = szeros;

S = fftshift(fft(sif, [], 1), 1);
```



Phase 1 on GPU

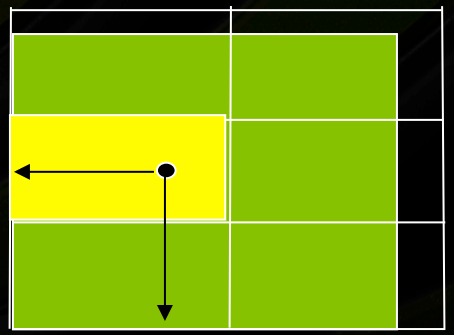
```
__global__ void along_track (double *sif_r, double *sif_i, int M, int N,
                             cufftDoubleComplex *sif_out, int zpad)
{
   int totalThreads = gridDim.x * blockDim.x;
   int ctaStart = blockDim.x * blockIdx.x;
   int start= round( (float)(zpad-M)/2);

   for ( int j = ctaStart + threadIdx.x; j < N; j += totalThreads)
   {
      double angle= 2.*M_PI*(j+1-(double)N/2)/(double)N;
      double h=0.5+0.5*cos(angle);

      for(int i=0;i<start;i++) { sif_out[i+j*zpad].x = 0.; sif_out[i+j*zpad].y = 0.;}

      for(int i=start;i<start+M;i++)
       {
          double a = 1-2*((i)&1); // FFTSHIFT
          sif_out[i+j*zpad].x = a *h *sif_r[i-start+j*M] ;
          sif_out[i+j*zpad].y = a *h *sif_i[i-start+j*M] ;
       }

      for(int i=start+M;i<zpad;i++) {sif_out[i+j*zpad].x = 0.; sif_out[i+j*zpad].y = 0.; }
   }

}
```



```
along_track<<<(N+127)/128,128>>>(sif_r,sif_i,M,N,sif_out,ZPAD);
cufftExecZ2Z(plan, sif_out, sif_out, CUFFT_FORWARD) ;
```

Phase 2 (match filter)

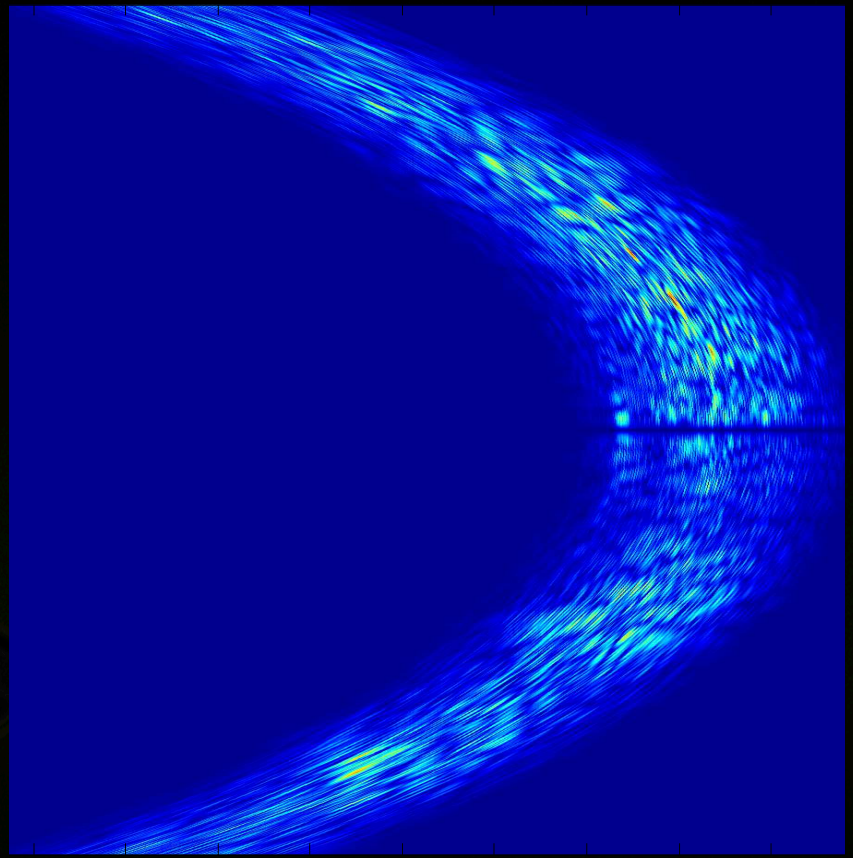
```
%step through each time step row to find phi_mf
for ii = 1:size(S,2)
   %step through each cross range
   for jj = 1:size(S,1)
      phi_mf(jj,ii) = Rs*sqrt((Kr(ii))^2 - (Kx(jj))^2);
   end
end

%apply matched filter to S
smf = exp(j*phi_mf);
S_mf = S.*smf;
```




```
__global__ void matched_filter(cufftDoubleComplex *S, int M, int N,
                               double Rs, double Kx0, double dx,
                               double Kr0, double dr)
{
   int i = blockIdx.x * blockDim.x + threadIdx.x;
   int j = blockIdx.y * blockDim.y + threadIdx.y;

   double c,s;

   if (i<M && j<N)
   {
    double kx=Kx0+i*dx;
    double kr=Kr0+j*dr;
    double angle=Rs*sqrt(kr*kr-kx*kx);
    sincos(angle,&s,&c);
    double Sx=S[i+j*M].x*c-S[i+j*M].y*s;
    double Sy=S[i+j*M].x*s+S[i+j*M].y*c;
    S[i+j*M].x=Sx;
    S[i+j*M].y=Sy;
   }
}
```

Phase 3 (Stolt interpolation)

```
%FOR DATA ANALYSIS
kstart =73;
kstop = 108.5;
Ky_even = linspace(kstart, kstop, 1024);
%create evenly spaced Ky for interp for real data

count = 0;
for ii = 1:zpad;
   count = count + 1;
    Ky(count,:) = sqrt(Kr.^2 - Kx(ii)^2);
   S_st(count,:) = (interp1(Ky(count,:), S_mf(ii,:),
Ky_even));
end
S_st(find(isnan(S_st))) = 1E-30;
%set all Nan values to 0
```



Interpolate data on a equispaced mesh

Timing CPU implementation

```
octave:1> SBAND_RMA_IFP
  Along track FFT in      0.204203 seconds.
  Matched filter in      40.384115 seconds.
  Stolt interpolation in 20.263402 seconds.
  2D inverse FFT         14.344351 seconds.
  SAR processing in     110.220000 seconds.
```

Jetson TK1, Octave 3.6.4

```
matlab> SBAND_RMA_IFP
  Along track FFT in      0.056983 seconds.
  Matched filter in       0.073197 seconds.
  Stolt interpolation in  1.185652 seconds.
  2D inverse FFT          0.741271 seconds.
  SAR processing in       8.080000 seconds.
```

Macbook Pro, 2.3GHz Intel Core i7, MATLAB 2013b

Timing GPU implementation

```
octave:1>
zpad=2048;
v= sar_gpu(sif,zpad,Kr,Rs,delta_x);
Data is complex, M=55, N=441, ZPAD=2048
***** cudaMalloc time:     0.088 seconds
***** fftplan    time:      0.654 seconds
***** warmup GPU time:  0.040 seconds
***** copy H2D   time:     0.001 seconds
***** GPU Comp   time:   0.332 seconds
***** Copy D2H   time:    1.316 seconds
***** unpack     time:      0.232 seconds
***** free       time:      0.072 seconds
***** Total      time:      2.936 seconds
  GPU 3.033000 seconds.
```

Times obtained with wallclock

Detailed timing GPU implementation

Time(%)	Time	Name
79.02%	1.37451s	CUDA memcpy DtoH]
8.06%	140.22ms	spVector8192D::fftDirection_t=1
6.75%	117.38ms	spRadix0064B::fftDirection_t=1
1.69%	29.466ms	[CUDA memcpy DtoD]
1.33%	23.151ms	set_to_zero(float2*, int, int)
1.21%	21.068ms	matched_filter(double2*,...
0.97%	16.866ms	interp_kernel_float(double2 ..
0.77%	13.467ms	dpVector2048D::fftDirection_t=-1
0.19%	3.3606ms	along_track(double*, ...)
0.00%	73.833us	[CUDA memcpy HtoD]

Times obtained with nvprof

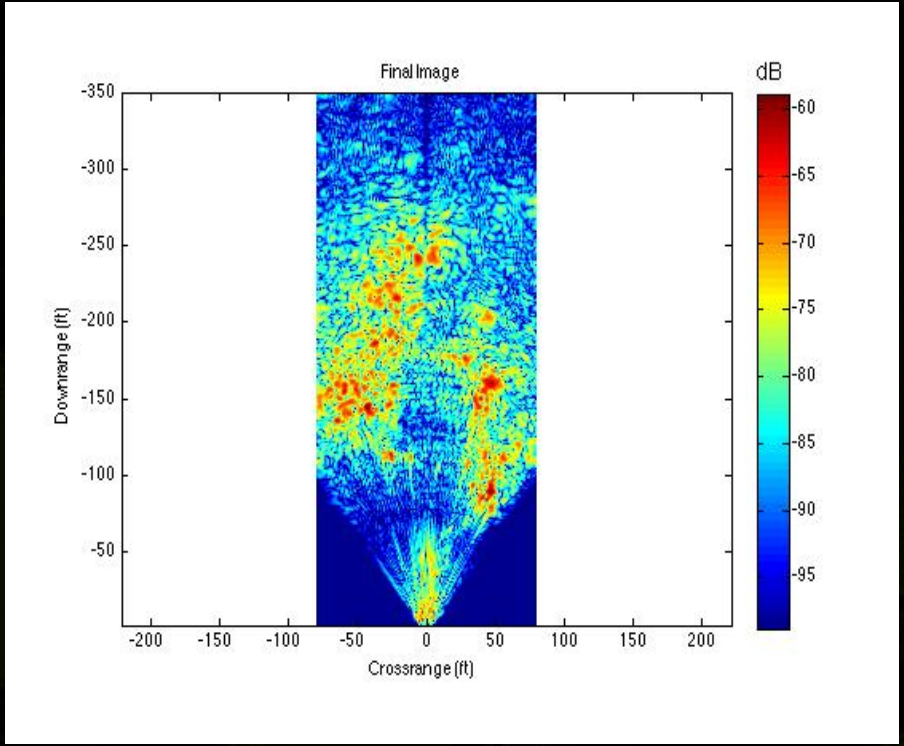
Optimized GPU implementation

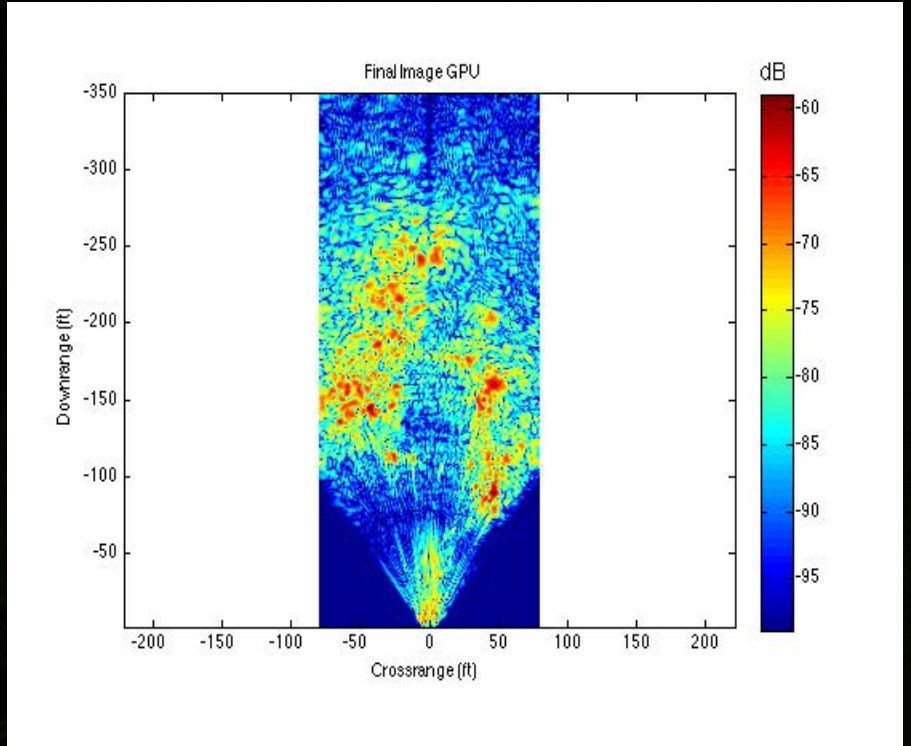
Bring back only the data needed for the image:

```
octave:1> SAR_gpu
Data is complex, M=55, N=441, ZPAD=2048
***** cudaMalloc time:          0.086784 seconds
***** fftplan    time:          0.507067 seconds
***** copy H2D   time:          0.000811 seconds
***** warmup GPU time:          0.002435 seconds
                ***** set_zero   time:          0.020761 seconds
                ***** along_trk  time:          0.002271 seconds
                ***** 1D FFT     time:          0.006936 seconds
                ***** mat filter time:          0.010135 seconds
                ***** Stolt interp time:        0.008533 seconds
                ***** 2D IFFT    time:          0.204319 seconds
                ***** flip rot   time:          0.026690 seconds
                ***** scale      time:          0.010833 seconds
                ***** thrust max time:          0.004489 seconds
maxx = -5.886394e+01
***** TOT GPU    time:          0.295100 seconds
***** Copy D2H   time:          0.157 seconds
***** free       time:          0.042222 seconds
***** Total      time:          1.118386 seconds

 GPU 1.142000 seconds.
```

```
octave:2> SAR_gpu
Data is complex, M=55, N=441, ZPAD=2048
***** cudaMalloc time:          0.032615 seconds
***** fftplan    time:          0.025240 seconds
***** copy H2D   time:          0.000718 seconds
***** warmup GPU time:          0.002434 seconds
                ***** set_zero   time:          0.020470 seconds
                ***** along_trk  time:          0.002399 seconds
                ***** 1D FFT     time:          0.006938 seconds
                ***** mat filter time:          0.009853 seconds
                ***** Stolt interp time:        0.008353 seconds
                ***** 2D IFFT    time:          0.209214 seconds
                ***** flip rot   time:          0.027359 seconds
                ***** scale      time:          0.011313 seconds
                ***** thrust max time:          0.004484 seconds
maxx = -5.886394e+01
***** TOT GPU    time:          0.300488 seconds
***** Copy D2H   time:          0.156 seconds
***** free       time:          0.043313 seconds
***** Total      time:          0.590023 seconds

 GPU 0.631000 seconds.
```

Comparison between GPU and CPU images





Complete implementation in CUDA

The whole processing, from reading the file to generating the data for the image is now completed in less than 1.5s

```
ubuntu@tegra-ubuntu:$ ./sar
 *** init free0   : 43.123 ms
 *** init mallc   : 0.333 ms
 *** init free    : 0.285 ms
 *** init cufft   : 475.635 ms
 *** cudaMallocHst: 68.579 ms
 *** read file    : 224.996 ms
 *** fftplan  time: 27.328 ms
 *** warmup GPU   : 1.639 ms
 *** trig_s_kernel: 19.407 ms
 *** rpstrt_kernel: 10.355 ms
 *** thrust scan  : 23.686 ms
 *** memset 1     : 5.094 ms
 *** ind_kernel   : 13.529 ms
 *** thrst sort   : 1.337 ms
 *** cudaMallocHst: 3.547 ms
 *** packRP_kernel: 6.582 ms
 *** thrust scan 2: 10.556 ms
 *** memset 2     : 0.247 ms
*** pind_kernel  : 5.050 ms
 *** max_kernel   : 4.045 ms
 *** iFFT         : 15.066 ms
 *** FFT          : 9.202 ms
*** sub_mean      : 0.460 ms
 *** set_zero     : 20.388 ms
 *** along_trk    : 10.601 ms
 *** 1D FFT       : 6.648 ms
 *** mat filter   : 10.127 ms
 *** sto interp   : 13.356 ms
 *** 2D IFFT      : 221.160 ms
 *** flip rot     : 40.547 ms
 *** scale        : 13.961 ms

GPU processing    : 0.465878 seconds
total time        : 1.442989 seconds
```

Conclusions

- Full SAR imaging ported on a CUDA-capable SOC using Octave and CUDA.

- Jetson TK1 is a very capable platform:
  - standard software toolchain
  - excellent performance with low power consumption
  - major limitation is the limited amount of memory, 2 GB

- Looking at the possibility to deploy Jetson on UAVs