

EDHPC 2023

European Data Handling & Data Processing Conference for Space



Hardware Accelerated Backprojection Algorithm on Xilinx UltraScale+ SoC-FPGA for On-Board SAR Image Formation

RUI POLICARPO DUARTE¹, HELENA CRUZ², MÁRIO VÉSTIAS¹, HORÁCIO NETO²

¹INESC-ID/ ISEL/IPL (LISBON, PORTUGAL)

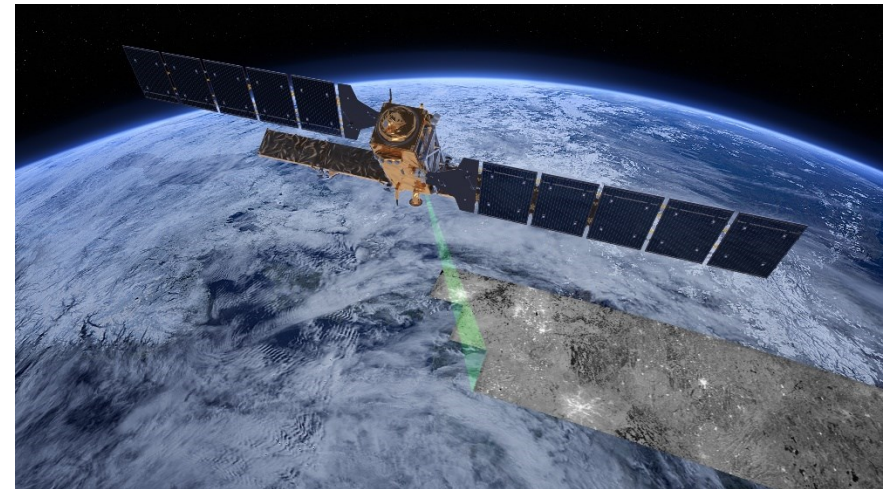
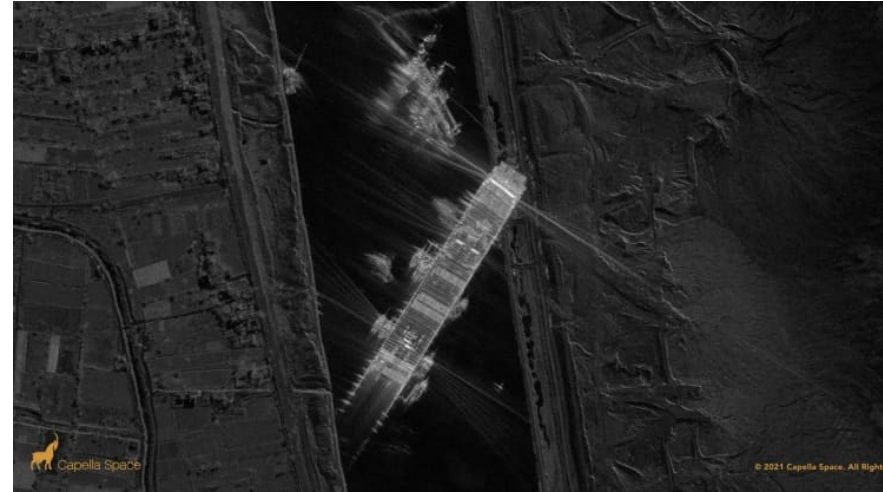
²INESC-ID IST/UL / (LISBON, PORTUGAL)

Introduction

- Motivation: improve the performance and image quality for on-board processors
 - Tight SWaP requirements – Low Size, Weight and Power
 - No Matlab or HPC-like infrastructure
- Objectives:
 - Implemente state-of-the-art algorithm on HW/SW architecture on a SoC-FPGA
 - Custom Optimization:
 - Data representation
 - Wordlength reduction
 - Explore algorithms parallelism
 - Provide good quality outputs

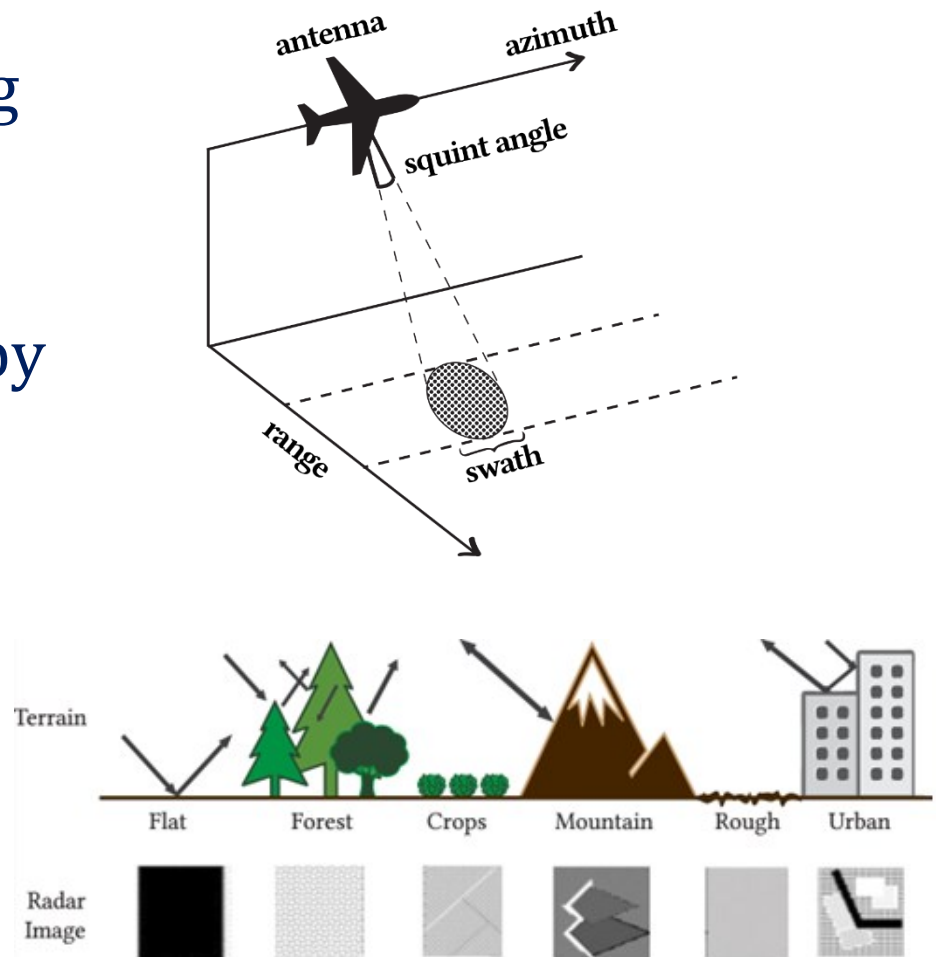
Synthetic Aperture Radar (I)

- SAR is a radar used to generate images of Earth or objects and can be used at any time of day and regardless of weather conditions.
- Has many applications: monitor sea ice and ship routing, crops, oil spills, etc.



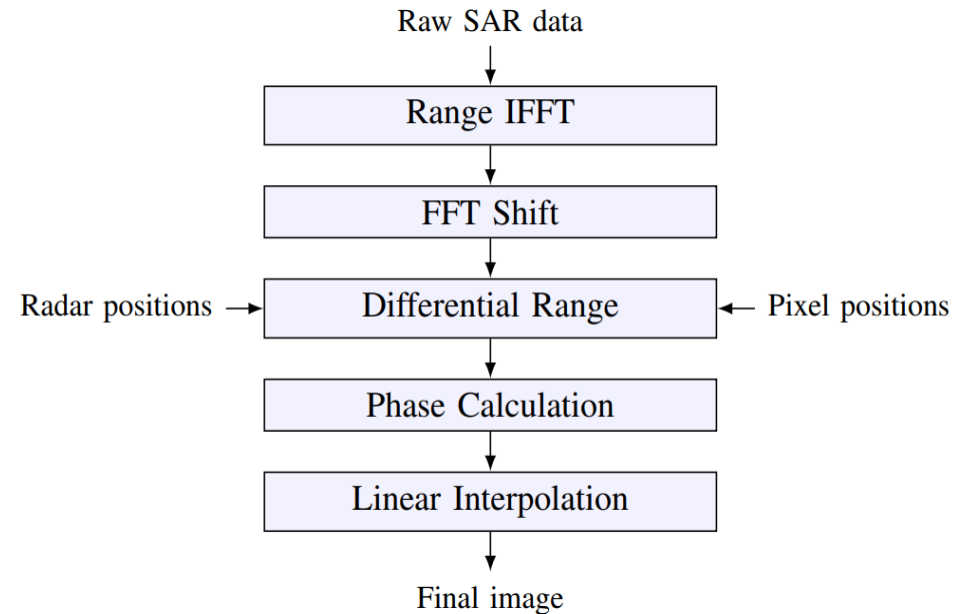
Synthetic Aperture Radar (II)

- SAR systems move along the azimuth direction while emitting pulses. These pulses are reflected and captured by the antenna.
- The Backprojection algorithm is a complex image generation algorithm with high computational complexity for SAR imaging systems.



Backprojection Algorithm Explained (I)

- The Backprojection algorithm is one of the best considering image quality.
- It has a computational complexity of $O(N^3)$ compared to the complexity of most SAR algorithms, $O(N^2 \log_2 N)$.



Backprojection Algorithm Explained (II)

```
1  for pulse in pulses:
2      rc = ifft(pulse)
3      fftshift(rc)
4
5  for pulse in pulses:
6      for pixel in image:
7          x_dist = (ant_x[pulse] - x_mat[pulse]) * (ant_x[pulse] - x_mat[pulse])
8          y_dist = (ant_y[pulse] - y_mat[pulse]) * (ant_y[pulse] - y_mat[pulse])
9          z_dist = (ant_z[pulse] - z_mat[pulse]) * (ant_z[pulse] - z_mat[pulse])
10         dR = sqrt(x_dist + y_dist + z_dist) - r0[pulse]
11
12         v = pi * 4 * C * min_f[pulse] * dR
13         phase_corr.re = cos(v)
14         phase_corr.im = sin(v)
15
16         interp_index = (int) ((dR + interp_const) / maxWr_nfft);
17
18         t = (dR - r_vec[interp_index]) * r_vec_inv_const;
19         interp_result.re = (1.0 - t) * rc[pulse][interp_index].re + t * rc[pulse][interp_index + 1].re;
20         interp_result.im = (1.0 - t) * rc[pulse][interp_index].im + t * rc[pulse][interp_index + 1].im;
21
22         image[pixel].re += (interp_result.re * phCorr.re - interp_result.im * phCorr.im);
23         image[pixel].im += (interp_result.re * phCorr.im + interp_result.im * phCorr.re);
..
```

Range compression

Distance calculation (norm)

Phase correction

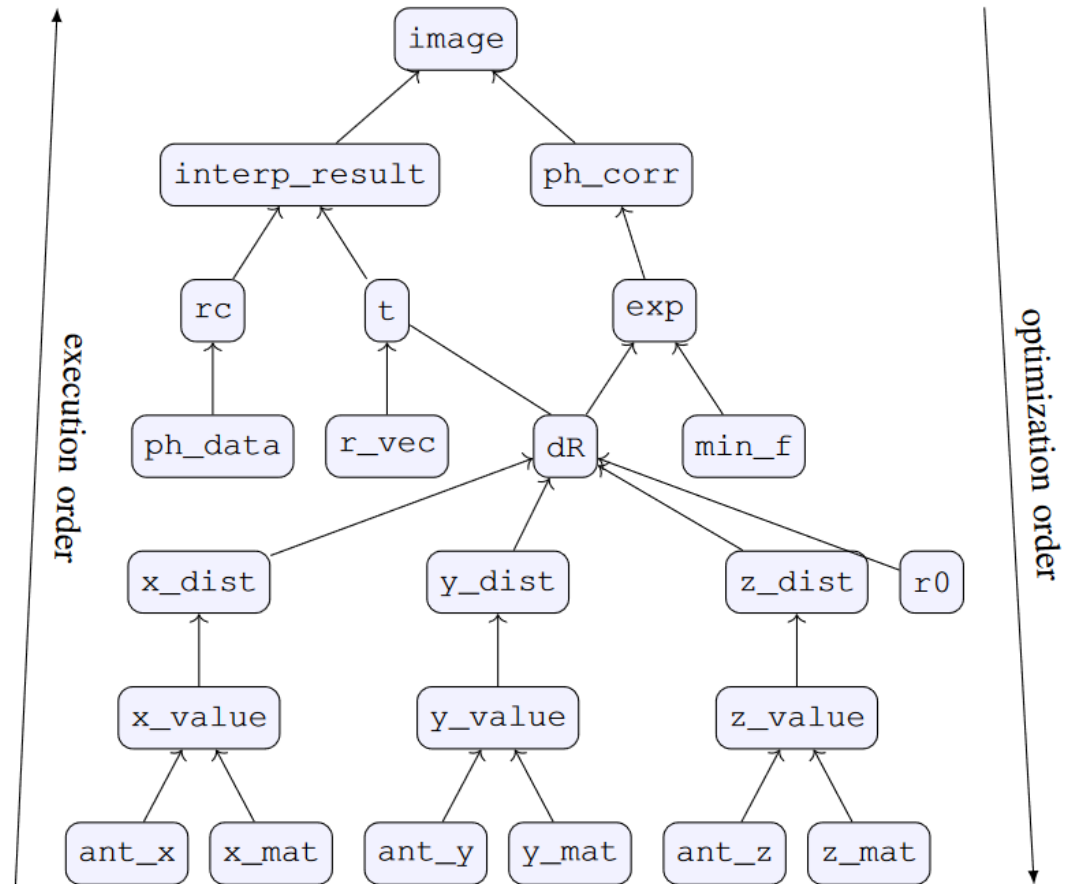
Interpolation

Accumulation

Pixel calculation

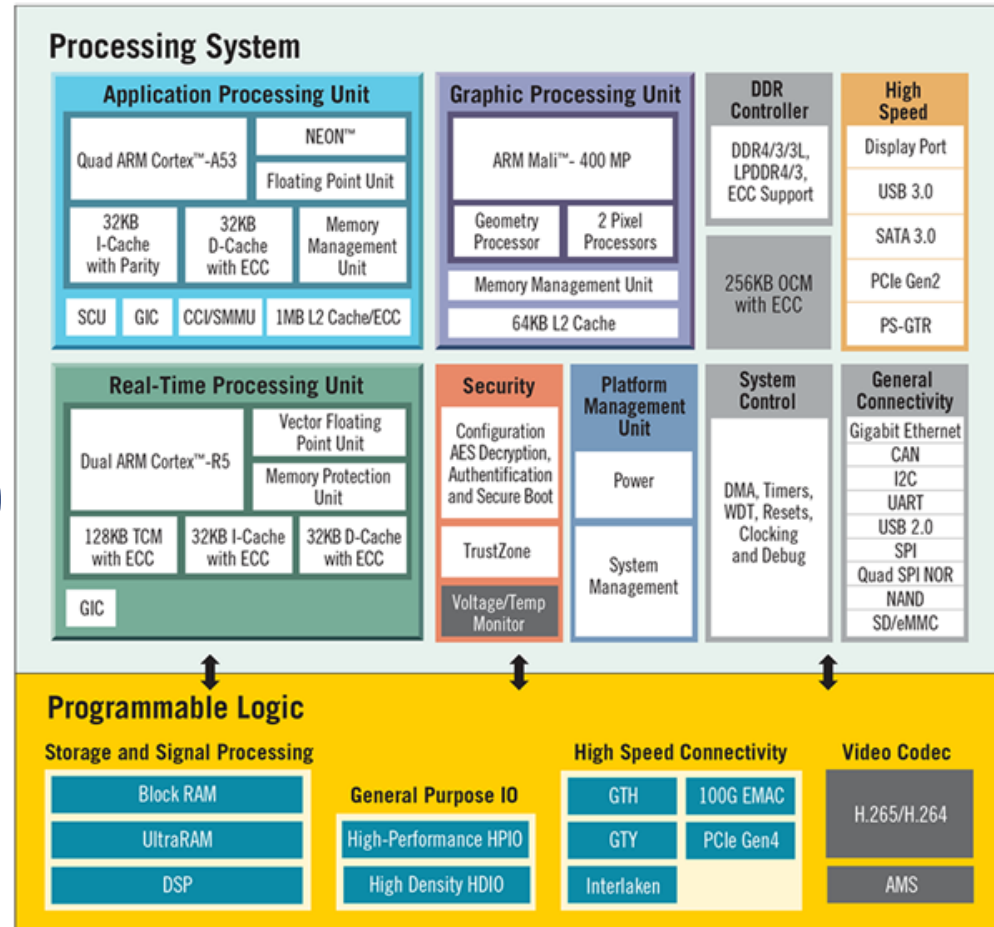
Backprojection Algorithm Explained (III)

- The Backprojection algorithm is one of the best considering image quality.
- It has a computational complexity of $O(N^3)$ compared to the complexity of most SAR algorithms, $O(N^2 \log_2 N)$.



Zynq UltraScale+ SoC-FPGA

- Quad-core Arm Cortex-A53
- Dual-core Arm Cortex-R5F
- 256KB of on-chip memory
- Reconfigurable fabric:
 - 504 Logic Cells
 - 1728 DSP Slices
 - 96 blocks URAM (27Mb)
 - 312 blocks of BRAM (11Mb)



Note: Illustration not drawn to scale.

HW/SW System Overview

Software

Antenna positions (ant_x, ant_y, ant_z) →
R0: start frequency of each pulse →
Pulses (after IFFT) →
Minimum frequency (min_f) →
Position of each pixel (x_mat, y_mat, z_mat) →

Final image ←

Floating-point

Fixed
-point

AXI Full

Hardware

Pulse contribution
calculation

Pulse contribution
calculation

⋮

⋮

Pulse contribution
calculation

Pulse contribution
calculation

Image ←

Fixed-point

Optimization Methodology

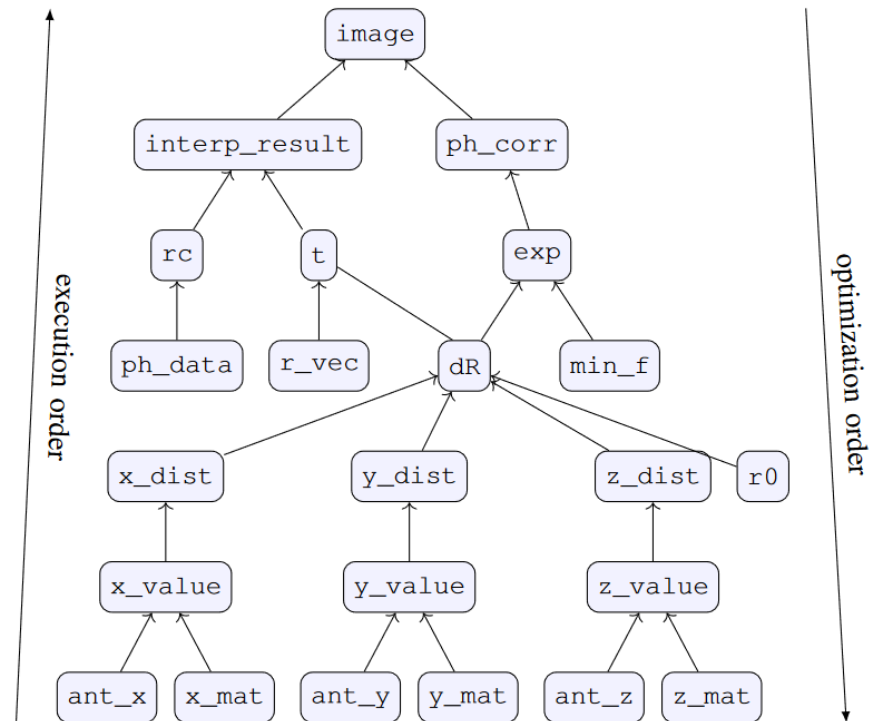
- Wordlength optimization
 - One size does not fit all!
 - Variables are related to different physical quantities
 - Some have more impact than others on the final image quality
- Data transfers and Internal Memories
 - The algorithm has 3 nested loops which promote different data transfers and internal storage requirements
 - Minimization of memory accesses via packing multiple values per address
- Parallel Processing
 - The contributions of each pulse are independent so they can be computed in parallel
 - Parallelization level depends on the occupation of the reconfigurable fabric

Wordlength Optimization (I)

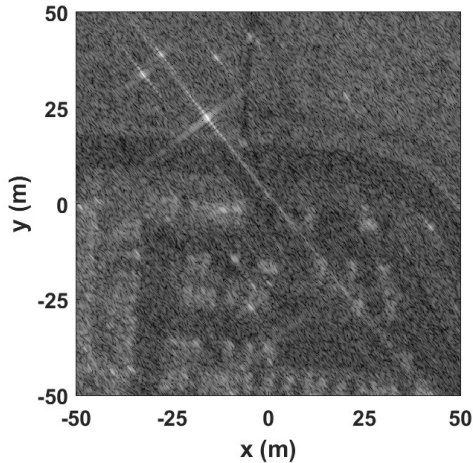
- Test the quality of the output
 - All wordlengths on all variables
 - Bottom-up approach
 - Floating-point and fixed-point

dR:

Truncated Bits	SSIM
0	1.0
⋮	⋮
18	1.0
19	0.9999999689526684
⋮	⋮
34	0.9992833850701763
35	0.9983809785534972
36	0.9957459498533803
37	0.9882183486161709
38	0.9674969080173169
⋮	⋮
51	0.5776500976891125
52	0.5768455548968846

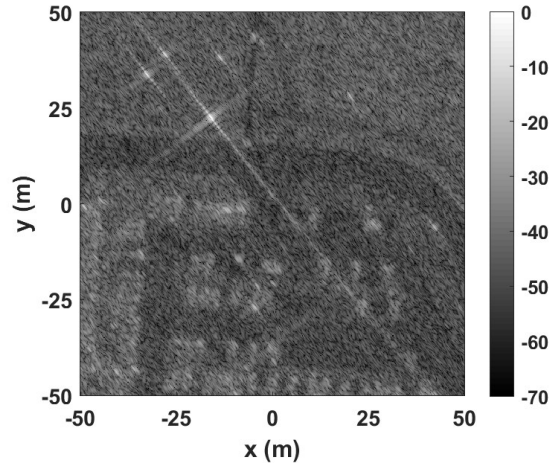


Wordlength Optimization (II)



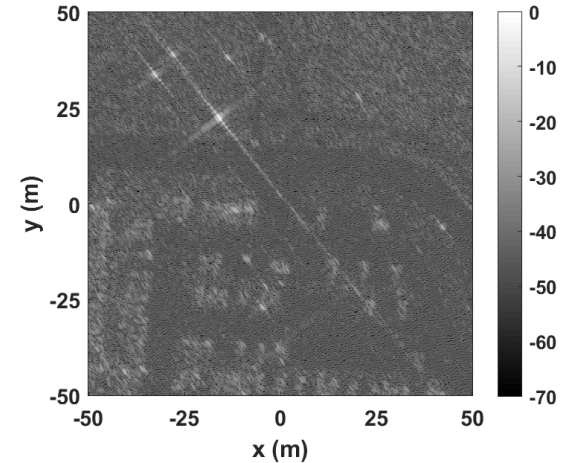
Fixed-point Q1.45 accumulator.

Output SSIM = 1.00



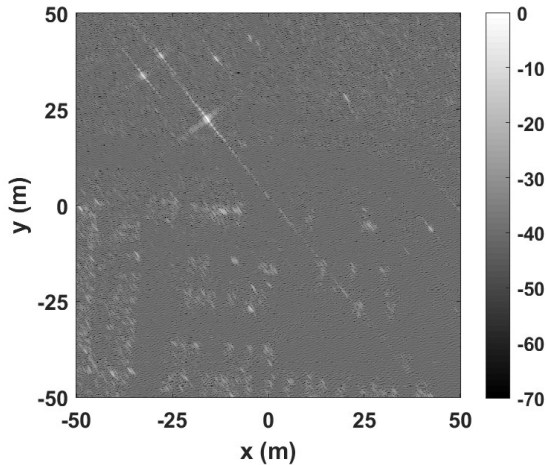
Fixed-point Q1.26 accumulator.

Output SSIM = 0.99



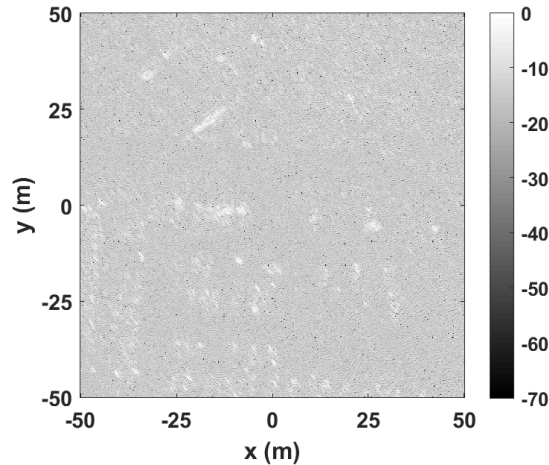
Fixed-point Q1.22 accumulator.

Output SSIM = 0.72



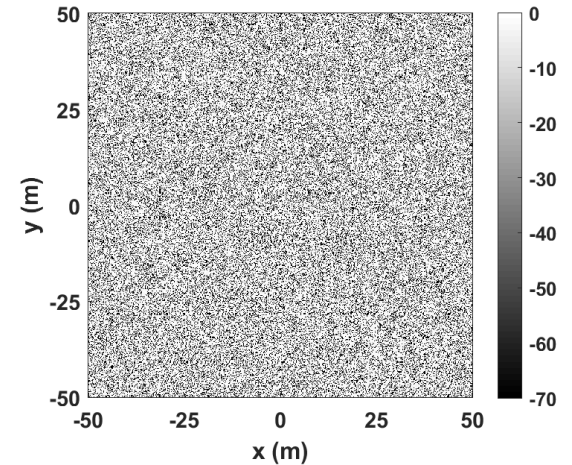
Fixed-point Q1.21 accumulator.

Output SSIM = 0.62



Fixed-point Q1.6 accumulator.

Output SSIM = 0.60



Fixed-point Q1.1 accumulator.

Output SSIM = 0.54


Wordlength Optimization (III)

Variable	SSIM = 1.0	SSIM \geq 0.99
ant_x	11	10
ant_y	11	10
ant_z	11	11
dR	32	12
image	45	26
interp_result	41	23
interp_const	14	6
maxWr	25	4
maxWr_nfft	37	16
nfft_inv	12	12
phCorr	25	4
phData	34	15
r0	10	10
rc	41	24
r_vec	26	7
t	19	2
x_dist	14	0
y_dist	19	0
z_dist	16	0
x_mat	25	5
y_mat	23	5
z_mat	0	0

Data Transfers From/To the Accelerator (I)

- The algorithm has 3 loops to compute the contribution of each pulse for all pixels on the image:

```
for (p = 0; p < N_PULSES; ++p)
{
    for (x = 0; x < NX; x++)
    {
        for (y = 0; y < NY; y++)
        {
            (...)
            // contribution of each pulse for each pixel;
            image[iy][ix] = accum;
        }
    }
}
```



MEMORY OCCUPIED BY THE INPUT DATA AND FINAL IMAGE WHEN THE P-LOOP IS THE OUTER LOOP.

p	Dimensions	Memory [KB]
min_f	117 * 35bits	0.50
r0	117 * 25bits	0.36
ant_x	117 * 25bits	0.36
ant_y	117 * 25bits	0.36
ant_z	117 * 25bits	0.36
rc	117 * 4096 * 32bits * 2 (complex values)	3744.00
r_vec	4096 * 33bits	16.50
image	501 * 501 * 46bits * 2 (complex values)	2818.86
Total		6581.3

```
for (y = 0; y < NY; y++)
{
    for (x = 0; x < NX; x++)
    {
        accum = 0;
        for (p = 0; p < N_PULSES; ++p)
        {
            (...)
            // contribution of all pulses for each pixel;
        }
        image[iy][ix] = accum;
    }
}
```

MEMORY OCCUPIED BY THE INPUT DATA AND FINAL IMAGE WHEN THE X-LOOP AND Y-LOOP ARE THE OUTER LOOPS.

x, y	Dimensions	Memory [KB]
x_mat	501 * 501 * 32bits	980.47
y_mat	501 * 501 * 32bits	980.47
z_mat	501 * 501 * 1bit	30.64
image	501 * 501 * 46bits * 2 (complex values)	2818.86
rc[p]	4096 * 32bits * 2 (complex values)	32.00
r_vec	4096 * 33bits	16.50
Total		4858.94

Data Flow and Memory Management

- Data loaded into the accelerator before the execution of the algorithm:
 - min_f, r0, ant_x, ant_y, ant_z and z_mat
- Range compressed (rc) pulses loaded before the two inner loops.
- x_mat is loaded into memory before the y-loop.
- The remaining variables are loaded as they are necessary.
- The image was divided into 12 blocks of 501x42 pixels to minimize the memory usage by the accelerator.

```
for (int b = 0; b < NR_BLOCKS; b++) {  
    init_block(image);  
    for (int p = 0; p < np_to_calc; p++) {  
        read_rc(p, data_rc, rc);  
        for (int x = 0; x < NX; x++) {  
            read_x_mat(x, data_x_mat, x_mat);  
            for (int y_block = 0; y_block < BLOCK_SIZE; y_block++) {
```

Inner Loop Processing

- It is independent so it can be parallelized
- SIN/COS:
 - ku variable is multiple of π
 - Input is between 0 and 2π
 - Avoid computing the remainder of 2π via a variable transformation on the SIN/COS function approximation

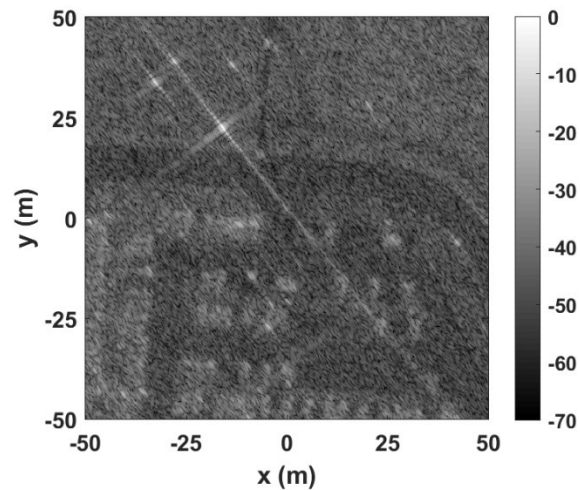
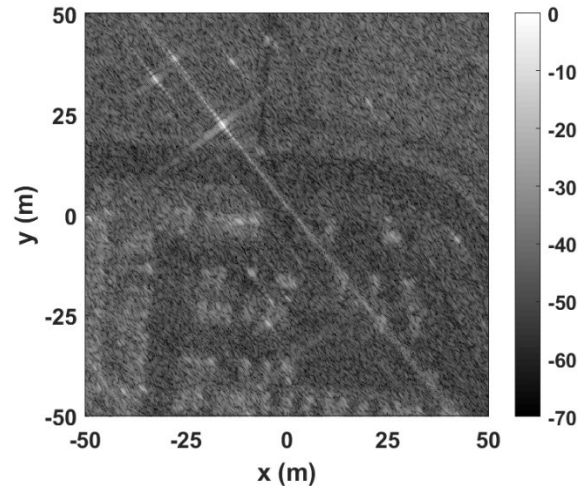
```
/* calculate the range R from the platform to this pixel */
const double xdiff = platpos[p].x - px;
const double ydiff = platpos[p].y - py;
const double zdiff = platpos[p].z - z0;
const double R = sqrt(xdiff*xdiff + ydiff*ydiff + zdiff*zdiff);
/* convert to a range bin index */
const double bin = (R-R0)*dR_inv;

complex sample, matched_filter, prod;
/* interpolation range is [bin_floor, bin_floor+1] */
const int bin_floor = (int) bin;
/* interpolation weight */
const float w = (float) (bin - (double) bin_floor);
/* linearly interpolate to obtain a sample at bin */
sample.re = (1.0f-w)*data[p][bin_floor].re + w*data[p][bin_floor+1].re;
sample.im = (1.0f-w)*data[p][bin_floor].im + w*data[p][bin_floor+1].im;

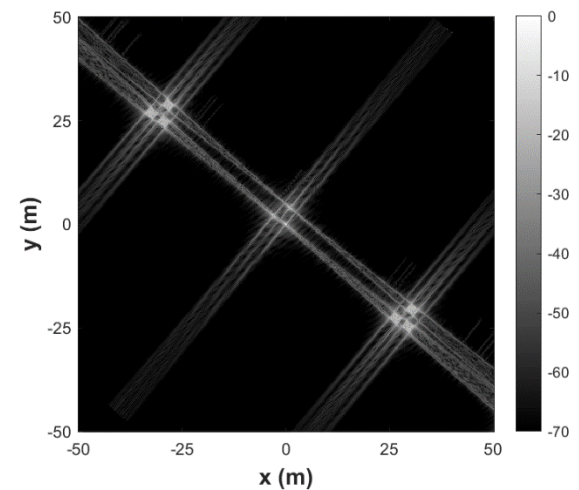
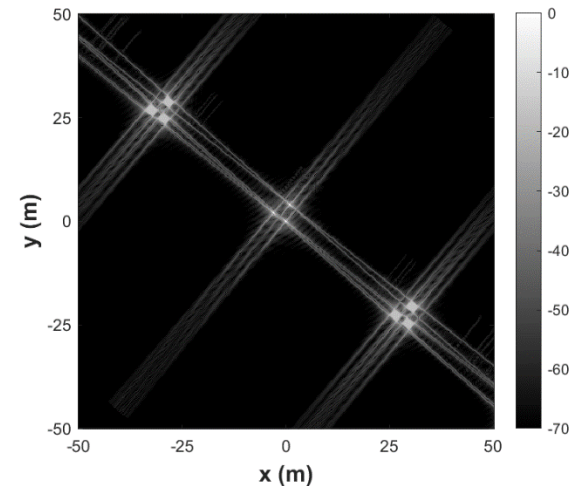
/* compute the complex exponential for the matched filter */
matched_filter.re = cos(2.0 * ku * R);
matched_filter.im = sin(2.0 * ku * R);
/* scale the interpolated sample by the matched filter */
prod = cmult(sample, matched_filter);
/* accumulate this pulse's contribution into the pixel */
accum.re += prod.re;
accum.im += prod.im;
```


Datasets and Performance Analysis

- Gotcha Dataset



- Synthetic Dataset



Results and Limitations

- Processing time:
 - SW-only: 27.72s
 - HW/SW: 11.42 s
 - Speedup: 2.43x
- Resources:
 - LUTs are in the highest occupation
 - Yet they are below 50% occupation
 - It is possible to further parallelize the algorithm

RESOURCES NECESSARY FOR THE ACCELERATOR IMPLEMENTATION

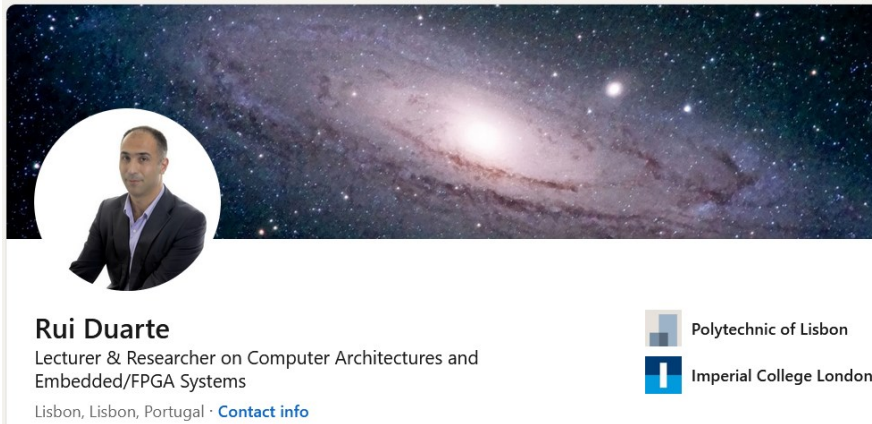
	BRAMs	URAMs	LUTs	DSPs
Total	41	36	106016	184
Device %	6%	37%	46%	10%

Conclusions & Future Work

- Proposed a novel HW/SW architecture using Zynq UltraScale+
- Performance improvement via custom wordlength optimization.
- Performance improve of 2.43x over the software-only implementation
- Future work will involve:
 - Introduction of a search algorithms over the wordlength optimization to reduce the design time.
 - Study of alternative architectures to improve the performance for real-time operation

End of Presentation

- Thank you for your interest on this work.
- Questions?



email: rpd@inesc-id.pt

- **Funding:**

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with references UIDB/50021/2020 and SFRH/BD/144133/2019, and has been also supported by project IPL/2022/eS2ST_ISEL through Instituto Politécnico de Lisboa.



TECH
NOLOGY
FROM
SEED