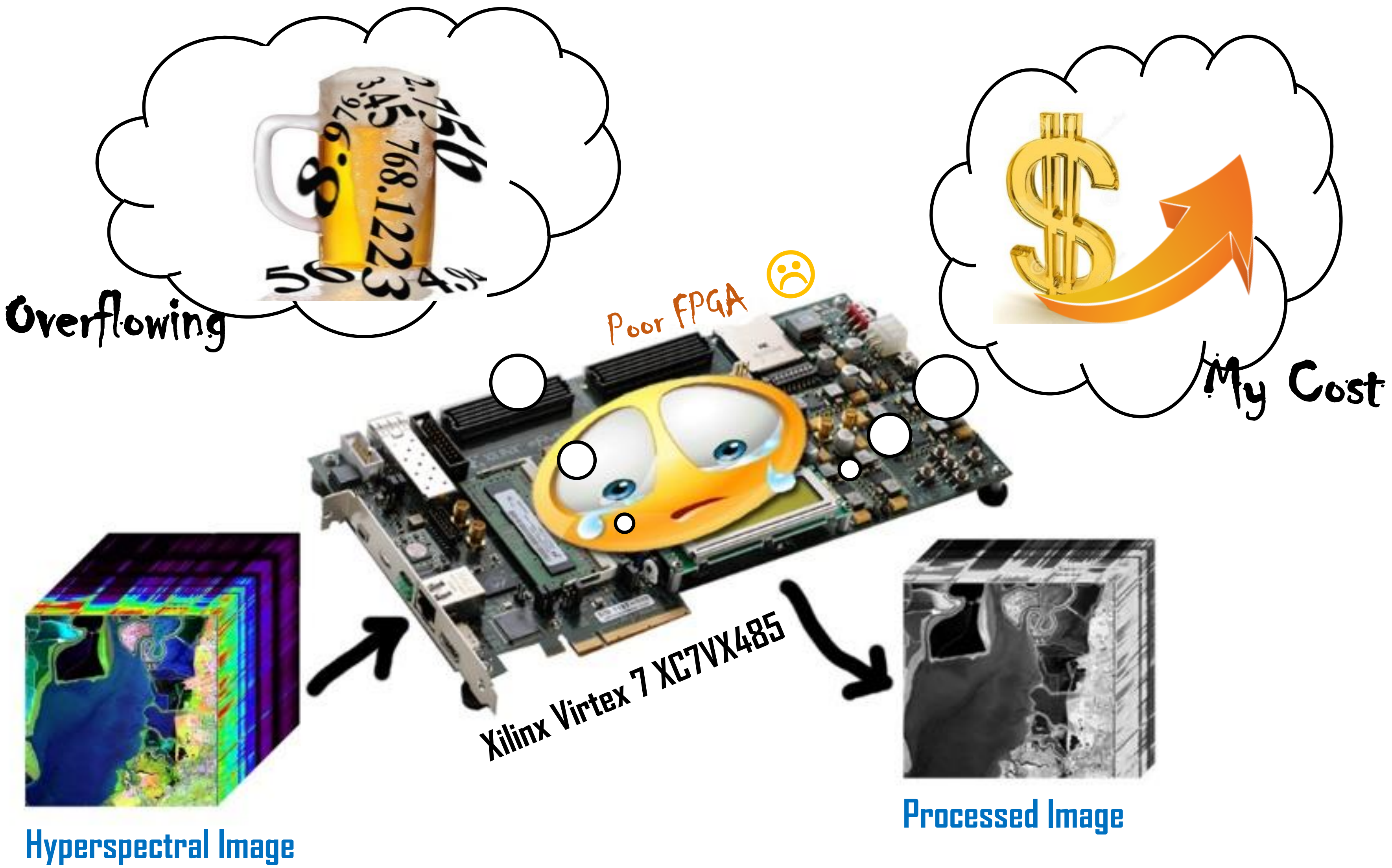
HyspIRI symposium-2015
NASA Goddard Spaceflight Center

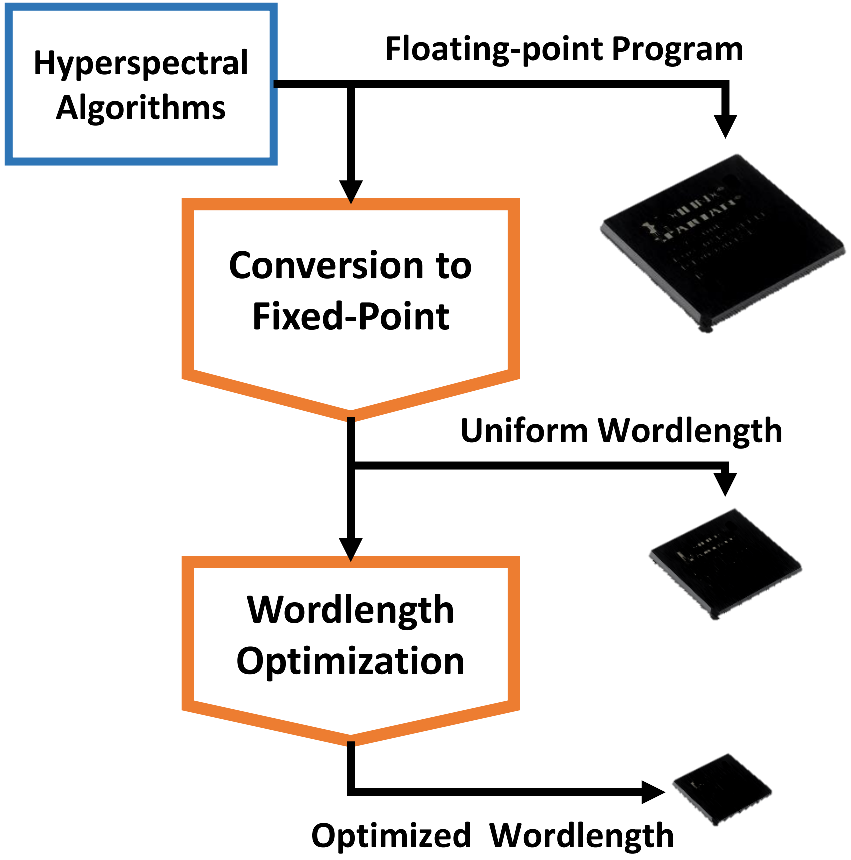
An Overflow-Free, Fixed-point based Singular Value Decomposition Algorithm for Dimensionality Reduction of Hyperspectral Images

Bibek Kabi, Anand S. Sahadevan, Ramanarayan Mohanty Aurobinda Routray, Bhabani. S. Das
Anmol Mohanty (Me!)

Indian Institute of Technology, Kharagpur


Overflowing
Poor FPGA
My Cost
Xilinx Virtex 7 XC7VX485
Hyperspectral Image
Processed Image

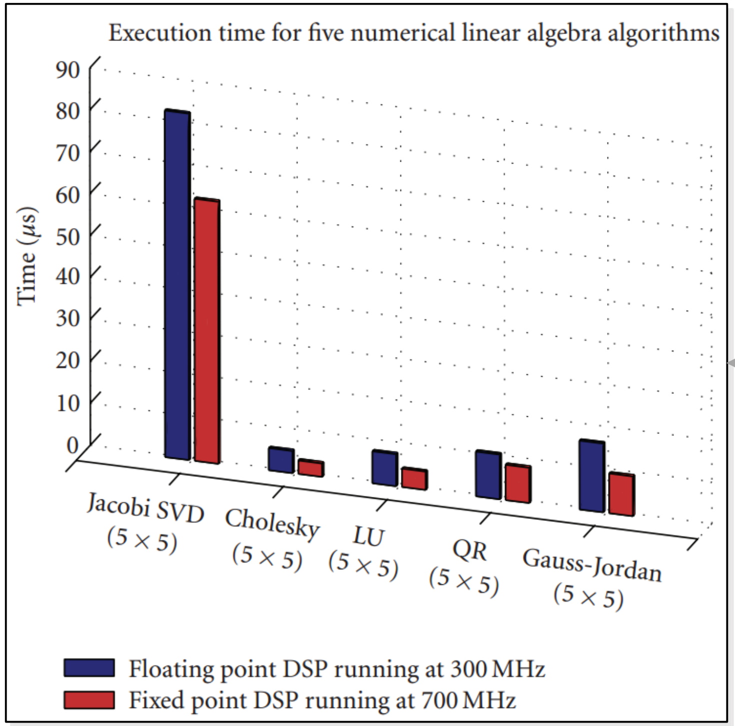
Motivation

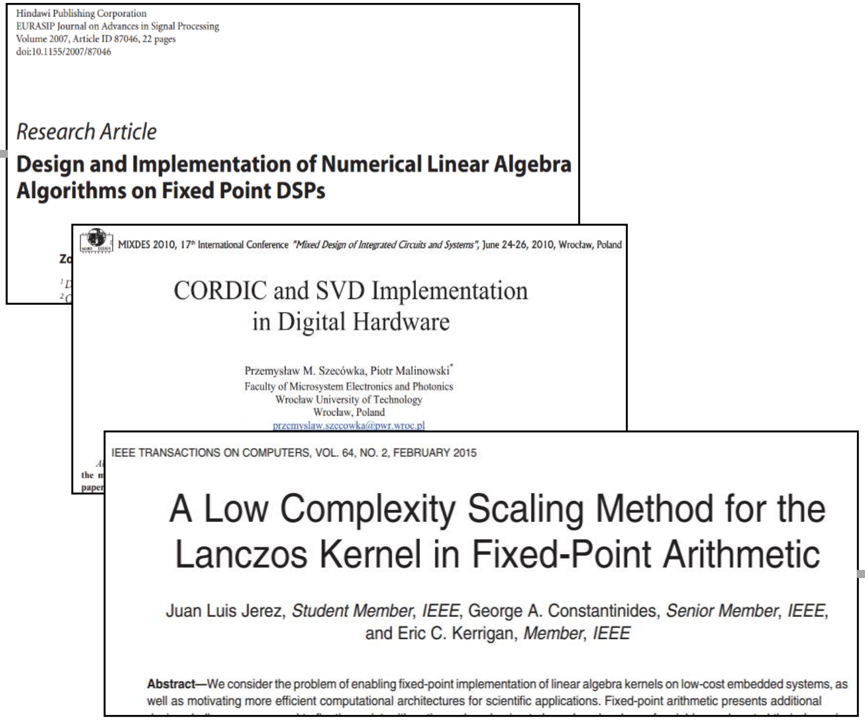


Hardware	Price	Power Consumption
Floating-Point Processor		
Fixed-Point Processor		
Fixed- Point ASIC		

FPGA: *Field Programmable Gate Array*
ASIC: *Application Specific Integrated Circuit*

Previous Works on Linear Algebra based on fixed point





Hindawi Publishing Corporation
EURASIP Journal on Advances in Signal Processing
Volume 2007, Article ID 87046, 22 pages
doi:10.1155/2007/87046

Research Article

Design and Implementation of Numerical Linear Algebra Algorithms on Fixed Point DSPs

MIXDES 2010, 17th International Conference *"Mixed Design of Integrated Circuits and Systems"*, June 24-26, 2010, Wrocław, Poland

CORDIC and SVD Implementation in Digital Hardware

Przemysław M. Szecówka, Piotr Malinowski*
Faculty of Microsystem Electronics and Photonics
Wrocław University of Technology
Wrocław, Poland
przemyslaw.szecowka@pwr.wroc.pl

IEEE TRANSACTIONS ON COMPUTERS, VOL. 64, NO. 2, FEBRUARY 2015

A Low Complexity Scaling Method for the Lanczos Kernel in Fixed-Point Arithmetic

Juan Luis Jerez, *Student Member, IEEE*, George A. Constantinides, *Senior Member, IEEE*, and Eric C. Kerrigan, *Member, IEEE*

Abstract—We consider the problem of enabling fixed-point implementation of linear algebra kernels on low-cost embedded systems, as well as motivating more efficient computational architectures for scientific applications. Fixed-point arithmetic presents additional

	Registers	LUTs	Latency (delay)
double	1046	911	14
float	557	477	11
FX53	53	53	1
FX24	24	24	1

∼ 20x resource savings
∼ 10x latency savings

Jerez et al., 2015

*SVD – Singular Value Decomposition

DATASET USED

Hyperspectral Images for Validation

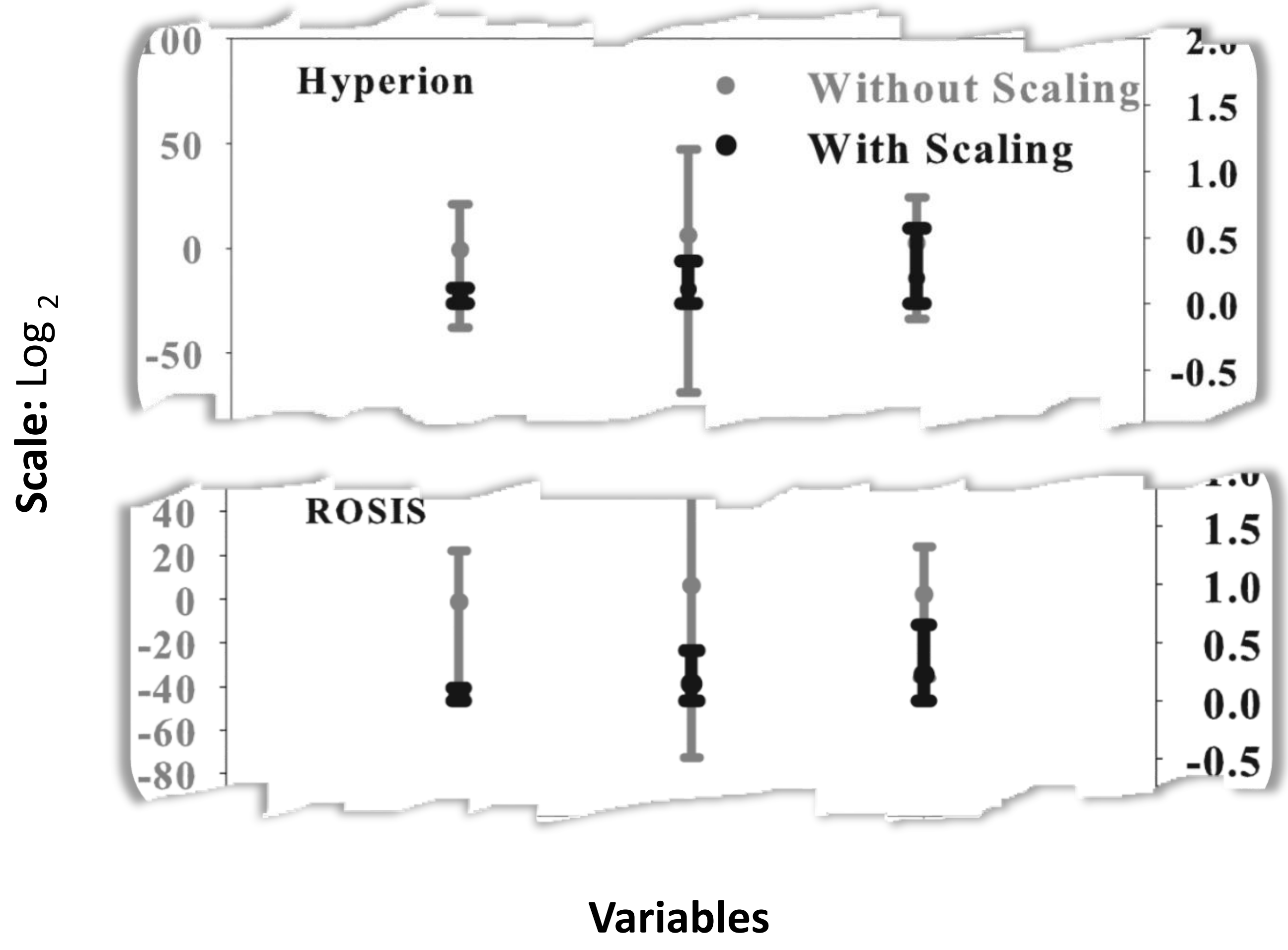
- **Hyperion (Space-borne):** Hyperion image contains the Chilika Lake site, India.
- **ROSIS (Air-borne):** ROSIS contains Pavia, University site, Italy.

Hyperion

ROSIS

Problems in Fixed point (due to Overflow)

Diverse & Large Range



Overflow

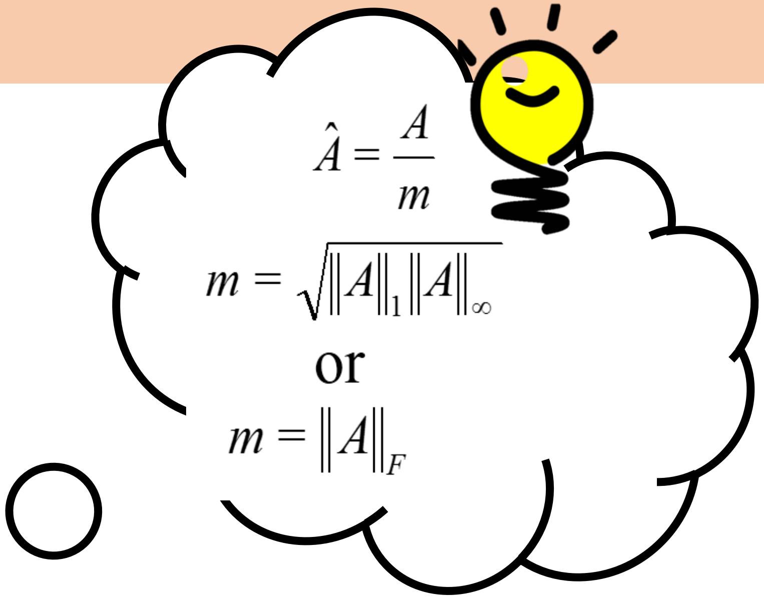
Data	SQNR
Hyperion	3.73
ROSIS	10.01

MSE		
PCs	**Hyperion**	**ROSIS**
PC1	2.0628e+05	6.7682e+04
PC2	1.0199e+03	6.5052e+03
PC3	1.5305e+04	4.6489e+03
PC4	262.3714	1.3140e+03
PC5		3.2786e+03

IWLs - Integer Wordlengths   SQNR - Signal-to-quantization-noise-ratio   PCs - Principal Components   MSE - Mean Squared Error

Proposal - Scaling Method!

If each element of a matrix is divided by the **square root of the product of its one-norm and infinity-norm** or **Frobenius norm** then all the variables generated during the computation of SVD will have **tight analytical ranges**



1.7395e+05	1.5038e+05	1.2673e+05
1.5038e+05	1.5677e+05	1.4146e+05
1.2673e+05	1.4146e+05	1.4673e+05
1.1601e+05	1.2567e+05	1.3744e+05
1.1131e+05	1.1879e+05	1.2754e+05

Scaling

0.0056	0.0048	0.0041
0.0048	0.0050	0.0045
0.0041	0.0045	0.0047
0.0037	0.0040	0.0044
0.0036	0.0038	0.0041

$$m = \sqrt{\|A\|_1 \|A\|_\infty}$$

0.0084	0.0073	0.0061
0.0073	0.0076	0.0069
0.0061	0.0069	0.0071
0.0056	0.0061	0.0067
0.0054	0.0058	0.0062

$$m = \|A\|_F$$

PROOF

Derivation in brief

Proof: Using vector and matrix norm properties, the ranges of the variables can be derived. We start by bounding the elements of the input matrix as

$$\max_{xy} |\hat{A}_{xy}| \leq \|\hat{A}\|_2 \leq 1$$

Given the scaling factor as $m = \sqrt{\|A\|_1 \|A\|_\infty}$, the Hestenes SVD algorithm applied to $\hat{A}$ has the following bounds for the variables for all i, j, x and y:

- $[\hat{A}]_{xy} \in [-1, 1]$
- $t \in [-1, 1]$
- $cs \in [0, 1]$
- $sn \in [-1, 1]$
- $[U]_{xy} \in [-1, 1]$
- $[V]_{xy} \in [-1, 1]$
- $a \in [0, r]$
- $b \in [0, r]$
- $c \in [-r, r]$
- $[\sigma_i]_x \in [0, 1]$,

where i, j denotes the iteration number and $[]_x$ and $[]_{xy}$ denote the x^{th} component of a vector and xy^{th} component of a matrix respectively.

Given the scaling factor as $m = \|A\|_F$, the Hestenes SVD algorithm applied to $\hat{A}$ has the following bounds for the variables for all i, j, x and y:

- $[\hat{A}]_{xy} \in [-1, 1]$
- $t \in [-1, 1]$
- $cs \in [0, 1]$
- $sn \in [-1, 1]$
- $[U]_{xy} \in [-1, 1]$
- $[V]_{xy} \in [-1, 1]$
- $a \in [0, 1]$
- $b \in [0, 1]$
- $c \in [-1, 1]$
- $[\sigma_i]_x \in [0, 1]$,

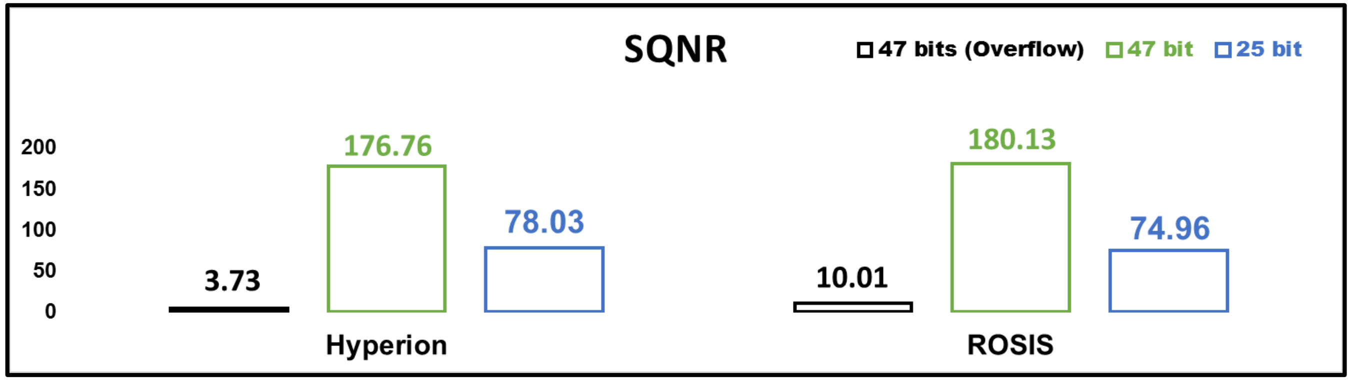
where i, j denotes the iteration number and $[]_x$ and $[]_{xy}$ denote the x^{th} component of a vector and xy^{th} component of a matrix respectively.

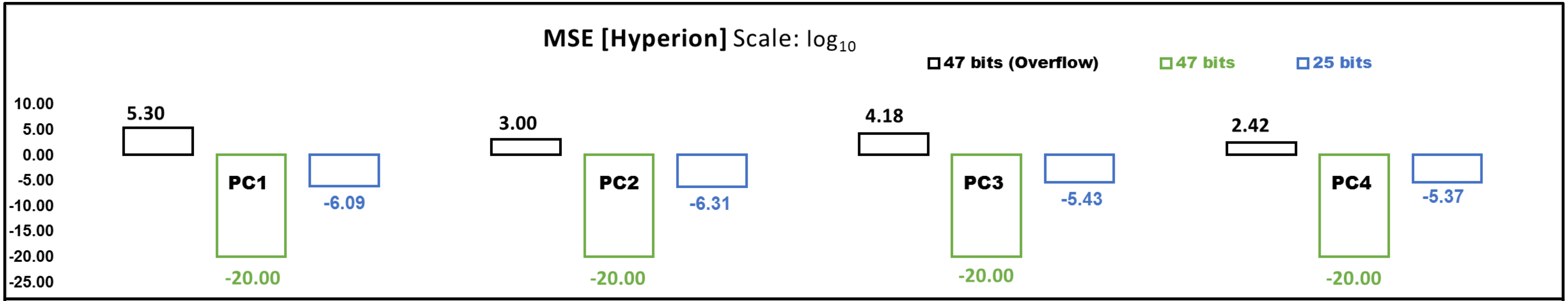
U is the left singular vector matrix, which is orthogonal and each column of *U* has unity norm. Hence all elements of are in the range [-1,1] following (*).

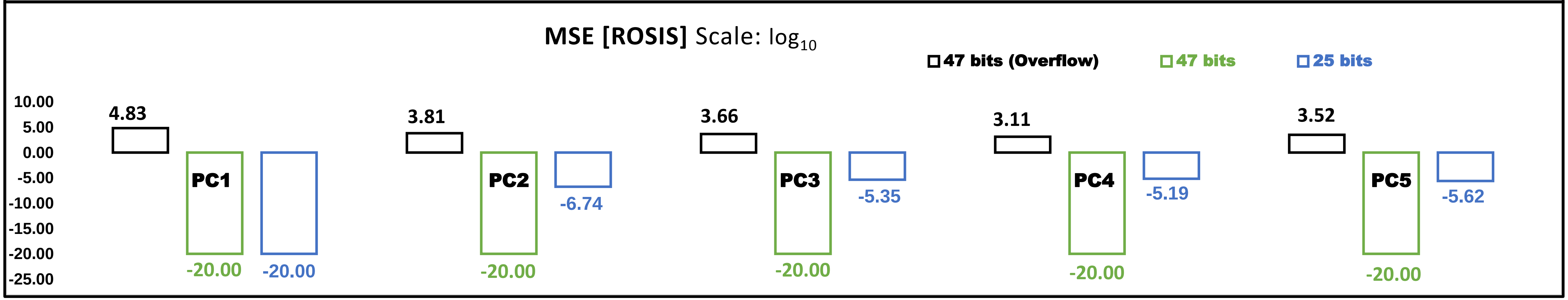
$$\|U(:,i)\|_\infty \leq \|U(:,i)\|_2 = 1 \qquad (*)$$

Similar is the case for right singular vectors *V*.

Results and Evaluation





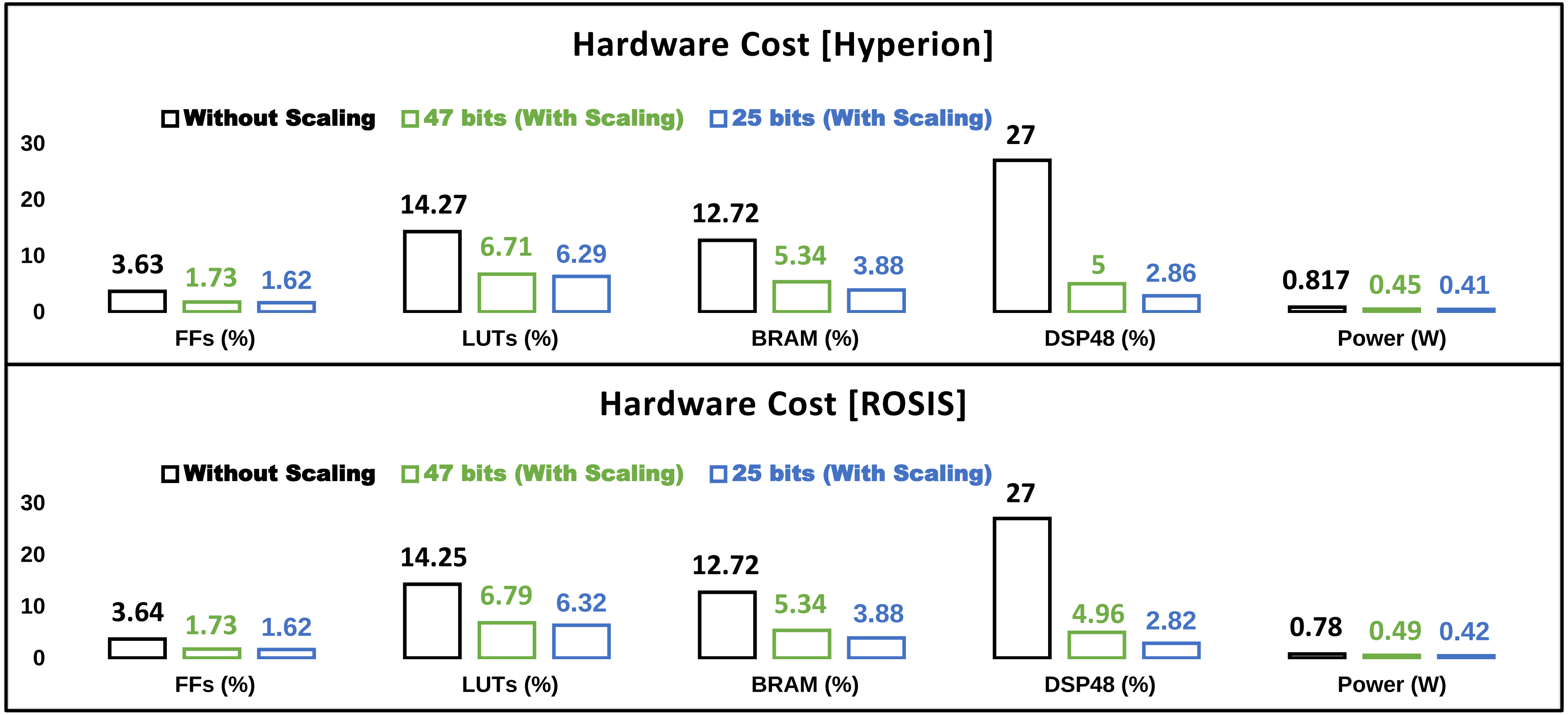


SQNR and MSE (scaling vs without scaling) with double precision floating-point as the reference

Reduced Hardware Cost

High-level synthesis [HLS] of fixed-point SVD algorithm on **Xilinx Virtex 7 XC7VX485** FPGA

Fixed-point code for HLS is implemented using SystemC



Percentage reduction in hardware cost after scaling		
53%–55% in FF (flip flop) **53%–56%** in LUT (look-up-table)	**59%–69%** in BRAM **82%–89%** in DSP48	**38%–50%** in On-Chip power

Backup slides

Validated

SQNR and MSE in fixed-point arithmetic (scaling vs without scaling) with double precision floating-point result as the reference.

Without Scaling (Overflow , 47 bits)

SQNR	47 bit
Hyperion	3.73
ROSIS	10.01

With Scaling (Overflow-free)

SQNR	47 bit	40 bit	35 bit	32 bit	25 bit
Hyperion	176.76	132.15	106.44	84.45	78.03
ROSIS	180.13	135.65	134.79	120.60	74.96

Without Scaling (Overflow, 47 bits)

MSE	Hyperion	ROSIS
PC1	2.0e+05	6.7e+04
PC2	1.0e+03	6.5e+03
PC3	1.5e+04	4.6e+03
PC4	262.3714	1.3e+03
PC5	NIL	3.3e+03

With Scaling (Overflow-free)

MSE	Hyperion				ROSIS				
	PC1	PC2	PC3	PC4	PC1	PC2	PC3	PC4	PC5
25 bit	8.1e-7	4.9e-7	3.7e-6	4.3e-6	0	1.8e-7	4.5e-6	6.5e-6	2.4e-6
32 bit	4e-9	1.3e-7	2.1e-6	1.8e-6	0	4.6e-7	3e-6	5.8e-6	2.3e-6
35 bit	0	0	8.5e-7	1.9e-7	0	0	0	0	0
40 bit	0	0	9.5e-11	2.2e-9	0	0	0	0	0
47 bit	0	0	0	0	0	0	0	0	0

Reduced Cost & On-chip Power Consumption

Without Scaling

COST	Utilization (%)	
	Hyperion	ROSIS
FF	3.63	3.64
LUTs	14.27	14.25
BRAM	12.72	12.72
DSP48	27.00	27.00
On-Chip Power	Consumption (W)	
	Hyperion	ROSIS
Power	0.817	0.783

With Scaling

COST	Utilization (%)									
	Hyperion					ROSIS				
	25 bit	32 bit	35 bit	40 bit	47 bit	25 bit	32 bit	35 bit	40 bit	47 bit
FF	1.62	1.63	1.67	1.71	1.73	1.62	1.63	1.67	1.71	1.73
LUT	6.29	6.41	6.51	6.56	6.71	6.32	6.48	6.53	6.62	6.79
BRAM	3.88	4.17	4.66	5.15	5.34	3.88	4.17	4.66	5.15	5.34
DSP48	2.86	3.29	5	5	5	2.82	3.25	4.96	4.96	4.96
On-Chip Power	Consumption (W)									
Power	0.41	0.43	0.43	0.45	0.45	0.42	0.44	0.44	0.46	0.49

With Scaling

COST	Reduction (%)									
	Hyperion					ROSIS				
	25 bit	32 bit	35 bit	40 bit	47 bit	25 bit	32 bit	35 bit	40 bit	47 bit
FF	55.37	55.09	53.99	52.89	52.34	55.49	55.21	54.12	53.02	52.47
LUT	55.92	55.08	54.37	54.02	52.97	55.64	54.52	54.17	53.54	52.35
BRAM	69.49	67.21	63.36	59.51	58.01	69.49	67.21	63.36	59.51	58.01
DSP48	89.4	87.81	81.48	81.48	81.48	89.55	87.96	81.62	81.62	81.62
Power	49.44	47.73	47.36	44.79	44.55	46.36	43.67	42.91	41.63	37.42

One-sided Jacobi SVD algorithm (Hestenes)

```
V = I;
for l = 1 to n do
  for i = 1 to n do
    for j = i + 1 to n do
/* compute (a c; c b) ≡ the (i, j) submatrix of A^T A */
      a = A(:, i)^T A(:, i);
      b = A(:, j)^T A(:, j);
      c = A(:, i)^T A(:, j);
/* compute the Jacobi rotation which diagonalizes (a c; c b) */
      ζ = (b − a)/(2c);
      t = sign(ζ)/(|ζ| + sqrt(1 + ζ^2));
      cs = 1/sqrt(1 + t^2);
      sn = cs · t;
/* update columns i and j of A */
      for k = 1 to n do
        tmp = A(k, i);
        A(k, i) = cs · tmp − sn · A(k, j);
        A(k, j) = sn · tmp + cs · A(k, j);
      end for
/* update the matrix V of right singular vectors */
      for k = 1 to n do
        tmp = V(k, i);
        V(k, i) = cs · tmp − sn · V(k, j);
        V(k, j) = sn · tmp + cs · V(k, j);
      end for
    end for
  end for
end for
/* singular values are computed from the norms of the columns of the final A */
for i = 1 to n do
  σ_i = ||A(:, i)||_2;
end for
/* the left singular vectors U are computed from the normalized columns of the final A */
for i = 1 to n do
  U(:, i) = A(:, i)/σ_i;
end for
```