Reconfigurable approximate accelerators for signal processing on resource constrained systems

– from algorithms to real-time implementation

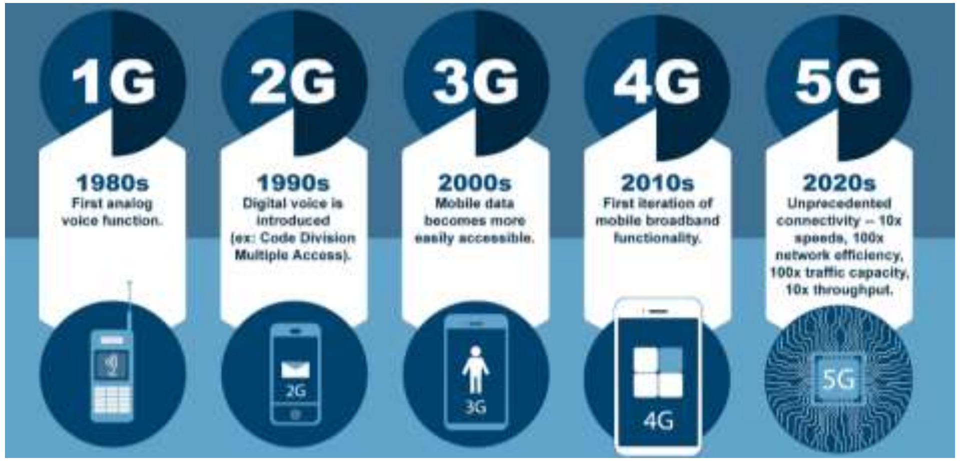
Yun Wu
Heriot-Watt University
30.11.2022

Signal Processing & Approximate Computing

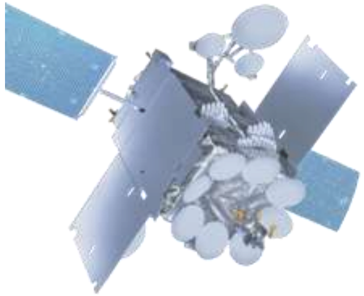
Insufficient Resources

Constrained Resources

Insufficient Bandwidth

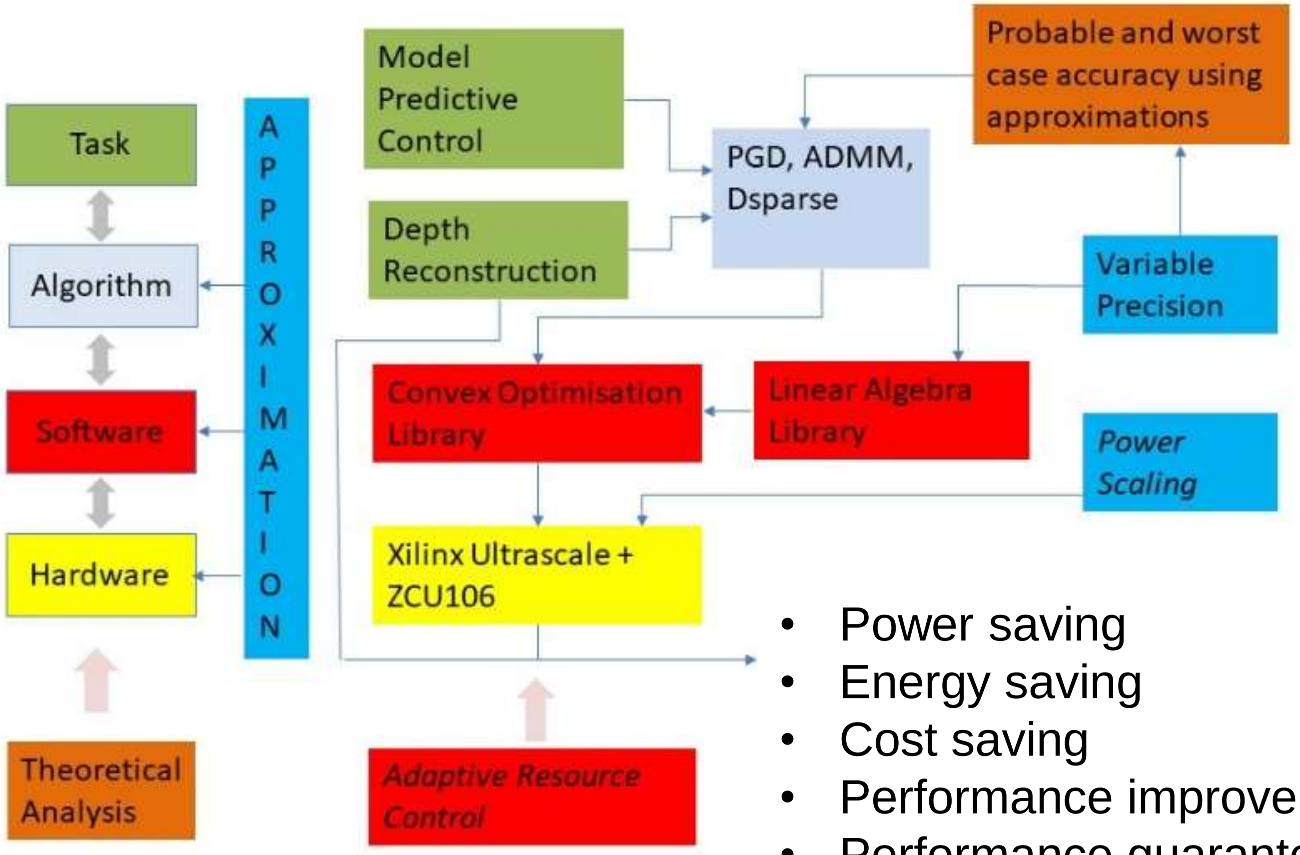


Constrained Bandwidth



Integrated Approximation Framework

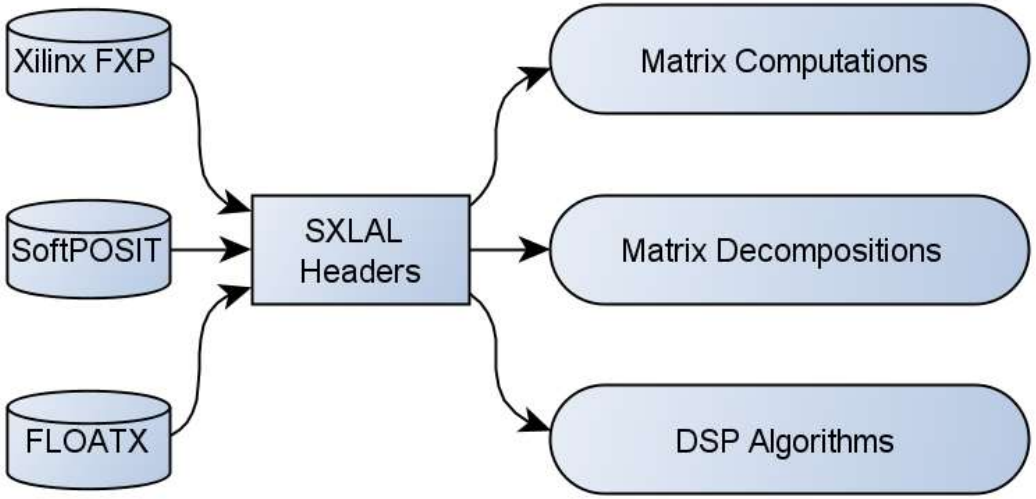
- A framework for analysing approximate errors towards efficient systems in terms of both power and space.
- Determine design choices for both low-level hardware, software, and high-level algorithmic approximations.
- Controllable approximations (variable precision, power scaling) have been deployed at several levels of the computational stack (left).
- The kernel of these approximations has been convex optimization
- Approximate libraries have been constructed to perform these tasks.



- Power saving
- Energy saving
- Cost saving
- Performance improvement
- Performance guarantee

Approximate Linear Algebra Library

- Header-only library without a compiled component
- Various computational precisions support for different arithmetic types
- Synthesizable through Xilinx High Lever Synthesis (HLS)



Approximate Arithmetic

- IEEE 754 floating point arithmetic

$$sign \cdot mantissa \cdot 2^{exponent}$$

sign	exponent	mantissa
1	e bits	m bits

- Q format fixed point arithmetic

$$sign \cdot \left(2^{integer} + 2^{-fraction}\right)$$

sign	integer	fraction
1	i bits	f bits

- Type III Unum – Posit

$$sign \cdot \left(2^{2^{p}}\right)^{regime} \cdot 2^{exponent} \cdot \left(1 + \frac{fraction}{2^{c}}\right)$$

sign	regime	exponent	fraction
1	p bits	e bits	c bits

Approximate Linear Algebra

Matrix types:

- ***Real – general real entries***
- Complex – general complex entries
- SPD – symmetric positive definite (real)
- HPD – Hermitian positive definite (complex)
- SY – symmetric (real)
- HE – Hermitian (complex)
- BND – band

Matrix Operations:

- ***BA – Basic Arithmetic (add, sub, mul, div, inv, etc.)***
- ***TF – triangular factorizations (LU, Cholesky)***
- ***OF – orthogonal factorizations (QR, QL, generalized factorizations)***
- EVP – eigenvalue problems
- SVD – singular value decomposition
- GEVP – generalized EVP
- GSVD – generalized SVD

	Real	Complex	SPD	HPD	SY	HE	BND	**BA**	**TF**	**OF**	EVP	SVD	GEVP	GSVD
SXLAL	**Yes**	No	No	No	No	No	No	**Yes**	**Yes**	**Yes**	No	No	No	No

Approximate Accelerators Generator

C++ kernel → Evaluation → Xilinx HLS → IP verification

C++ kernel
- Kernel (.cpp)
- Approximate Linear Algebra Library

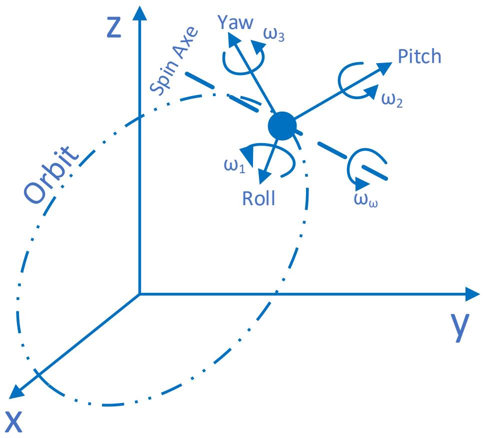
Simulation
- Mex Compilation
- Approximation Evaluation

Synthesis
- High Level Synthesis
- VHDL Code Generation

Evaluation
- Kernel (.vhd)

Performance
Cost
Power

Approximate Model Predictive Control



Spacecraft Attitude Control

- LASSO Problem formula:

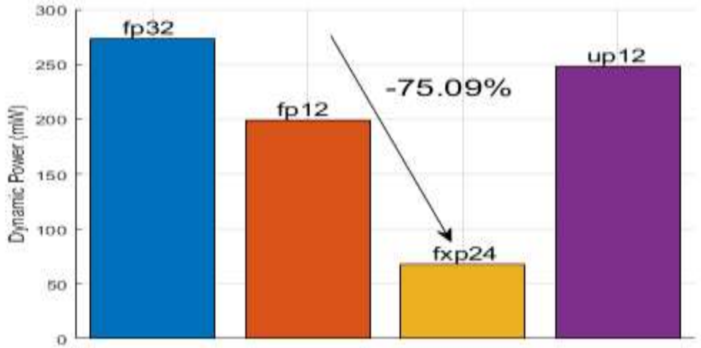
$$\arg\min_{u}\left\{\underbrace{\frac{1}{2}\|H\cdot u-b\|_2^2}_{g(u)}+\underbrace{\lambda\cdot\|u\|_1}_{h(u)}\right\}$$

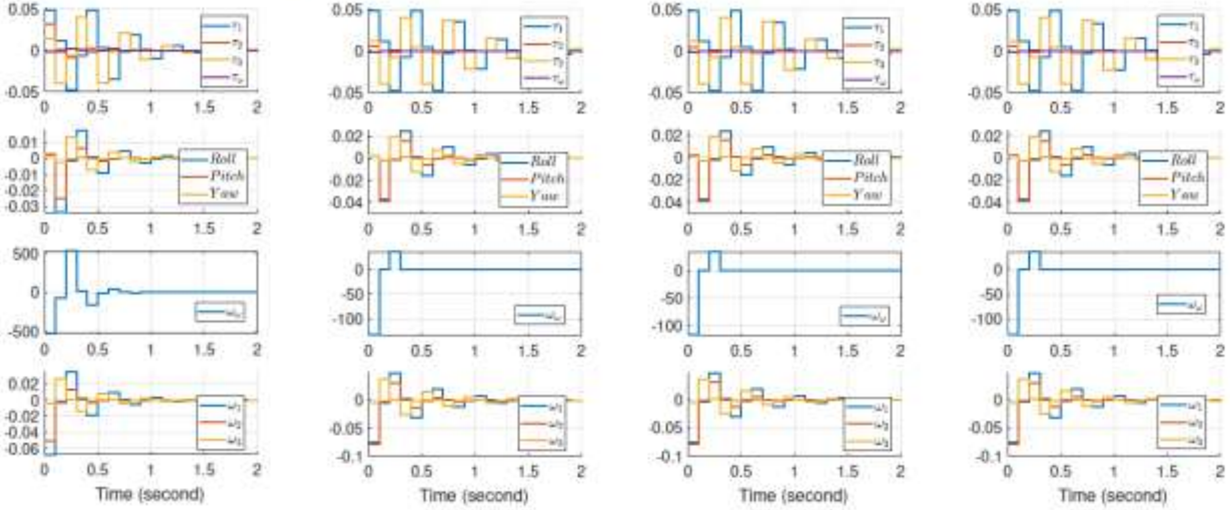
- Proximal operator:

$$u^{k+1}=prox_{\alpha h}\left(u^k-\alpha\cdot\nabla g(u^k)\right)$$

proximal gradient descent (PGD)

- ↓ 63.44% logic reduction





FP-32 FP-12 FXP-24 UP-12

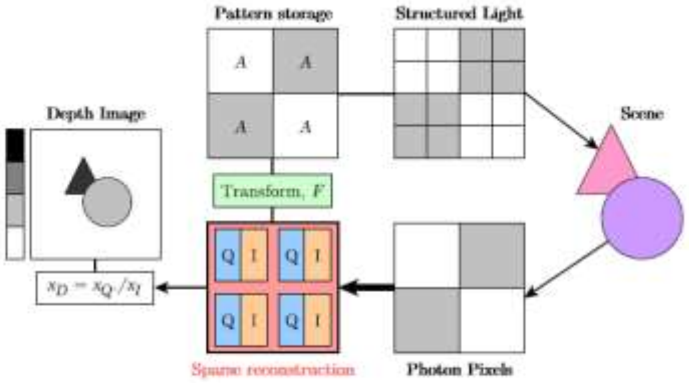
Parallel Compressed 3D Depth Reconstruction

Lidar depth reconstruction requires low power DSP for better signal-to-noise ratios

LASSO optimization problem in depth reconstruction:

$$\begin{cases} \arg,\min_{x_Q}\left(\frac{1}{2}\|y_Q - A\cdot x_Q\|_2^2 + \lambda\|F\cdot x_Q\|_1\right), \\ \arg,\min_{x_I}\left(\frac{1}{2}\|y_I - A\cdot x_I\|_2^2 + \lambda\|F\cdot x_I\|_1\right) \end{cases}$$

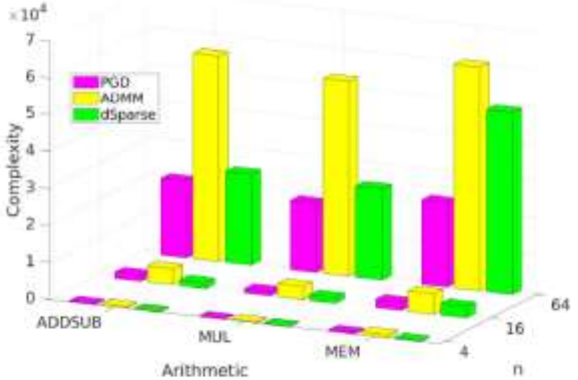
where y_Q, y_I is measurement, x_Q, x_I is the time-of flight (ToF)-sum and intensity (photon count), and F is the linear transformation function.



Parallel Depth Reconstruction System Diagram

Depth is reconstructed by solving the twins optimization problem and dividing the ToF-sum solution with intensity solution, within sub-divided parallel block of depth image:

$$\begin{cases} \tilde{x}_Q & = F^{-1}(x_Q) \\ \tilde{x}_I & = F^{-1}(x_I) \end{cases} \qquad x_D = \begin{cases} \tilde{x}_Q/\tilde{x}_I, & (for\ \ell ADMM\ and\ \ell PGD) \\ x_Q/x_I & (for\ dSparse) \end{cases}$$



Parallel Block Computational Complexity
(e.g., n=4x4=16)

Reduced Precision Convex Optimization Solver

ADMM - Alternating Direction Method of Multipliers

Input: $A, A^T y, L^{-1}, U^{-1}, \lambda_{ADMM}, y$
Output: x

$Initialization: const\{\alpha, \rho, \kappa = \lambda_{ADMM}/\rho\}, zeros\{z, q, u\}$

for $k = 0$ to k_{max} **do**
- $q^{k+1} = A^T \cdot y + \rho(z^k - u^k)$
- $x^{k+1} = q^{k+1}/\rho - 1/\rho^2 \cdot A^T \cdot (U^{-1} \cdot (L^{-1} \cdot (A \cdot q^{k+1})))$
- $\hat{x}^{k+1} = \alpha \cdot x^{k+1} + (1 - \alpha) \cdot z^k$
- $xu^{k+1} = \hat{x}^{k+1} + u^k$
- $z_1^{k+1} = \max\{0, xu^{k+1} - \kappa\}; z_2^{k+1} = \max\{0, -xu^{k+1} - \kappa\}$
- $z^{k+1} = z_1^{k+1} - z_2^{k+1}$
- $u^{k+1} = u^k + (\hat{x}^{k+1} - z^{k+1})$

end for
$x = z^{k_{max}}$
return x

PGD - Proximal Gradient Descent

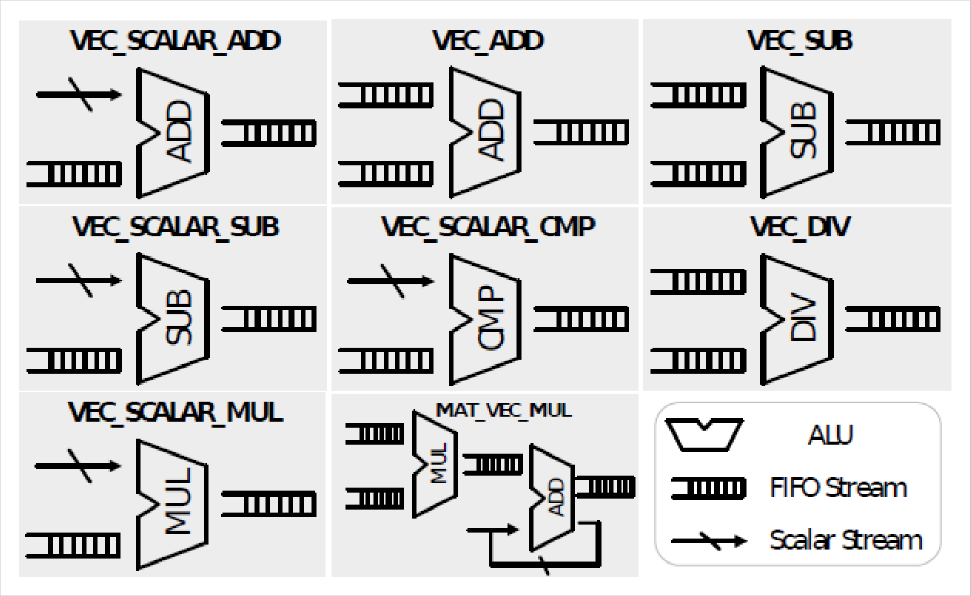
Input: W, V
Output: x

$Initialization: const\{\lambda^0_{PGD}, \beta^0, \gamma, k_{max}\}, zeros\{u^0\}$

for $k = 0$ to k_{max} **do**
- $g_x = Wx^k - V$
- $x^{k+1} = \mathbf{prox}_{\lambda^k_{PGD}}(x^k - \lambda^k_{PGD} \cdot g_x)$
- $\lambda^{k+1}_{PGD} = \lambda^k_{PGD} \cdot \beta^k$
- $\beta^{k+1} = min(\gamma \cdot \beta^k, 1)$

end for

Approximate matrix/vector arithmetic modules, including vector addition, multiplication, division, comparison and matrix to vector multiplication

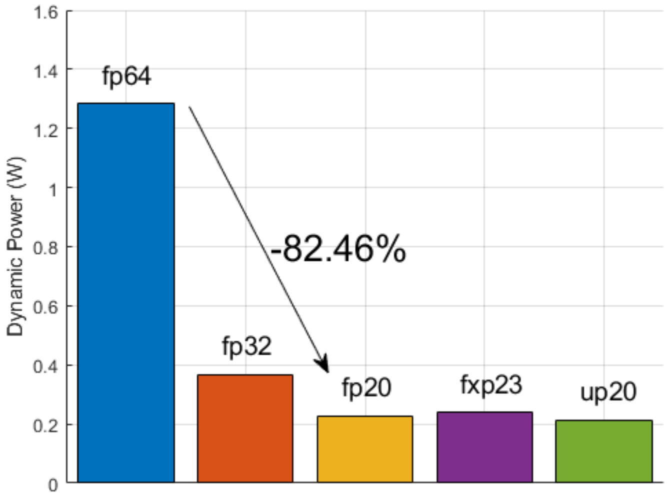


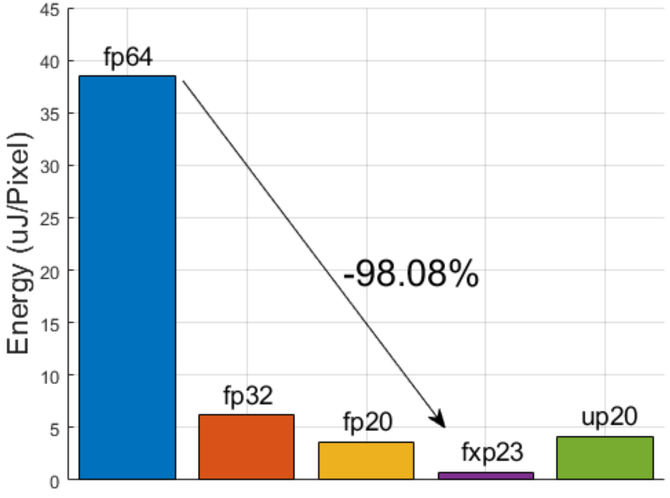
ADMM - Alternating Direction Method of Multipliers

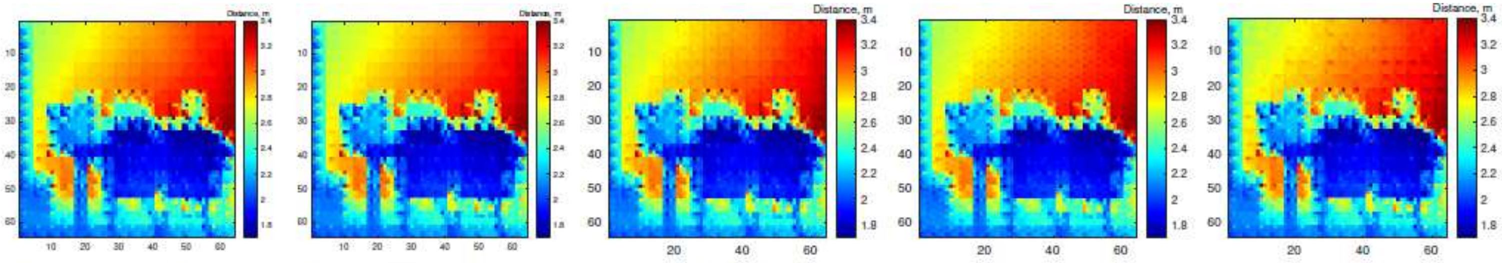
Compare to fp64:

- ↓ 79.65% logic reduction
- $over\ 6 \times$ throughput

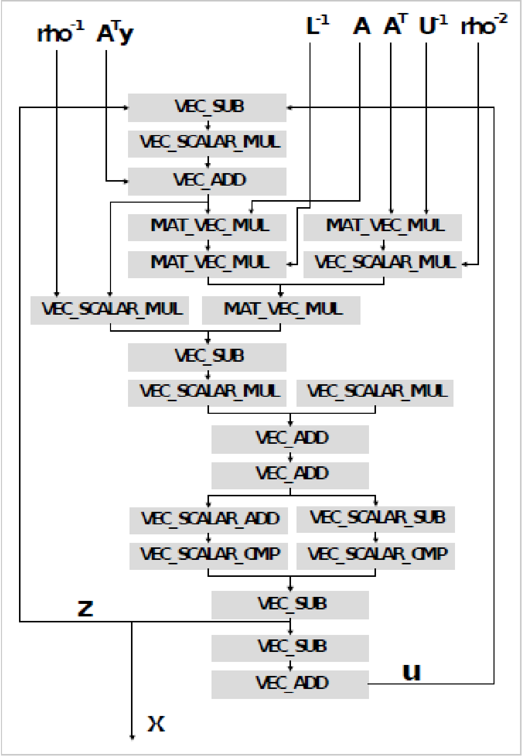
fp – floating point
fxp – fixed point
up – unum posit

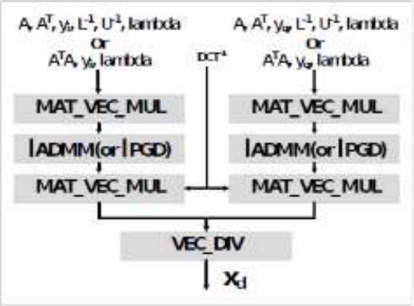






(a) ℓADMM (FP64), PSNR=28.69 dB, SSIM=0.75
(b) ℓADMM (FP32), PSNR=28.69 dB, SSIM=0.75
(c) ℓADMM (FP20), PSNR=28.69 dB, SSIM=0.74
(d) ℓADMM (FXP23), PSNR=28.72 dB, SSIM=0.72
(e) ℓADMM (UP20), PSNR=30.57 dB, SSIM=0.70



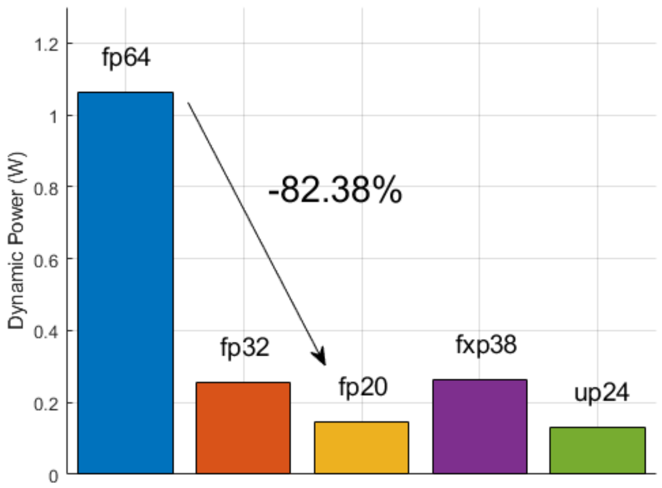


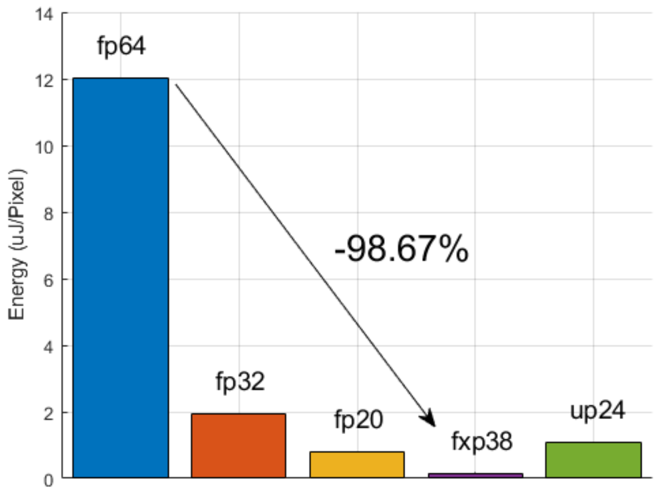
PGD - Proximal Gradient Descent

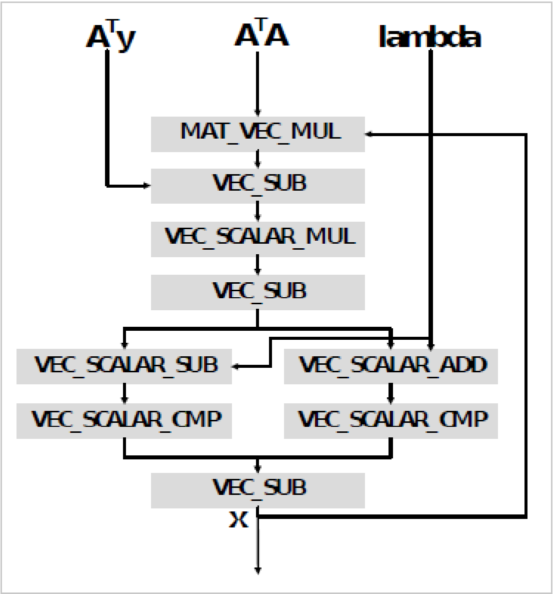
Compare to fp64:

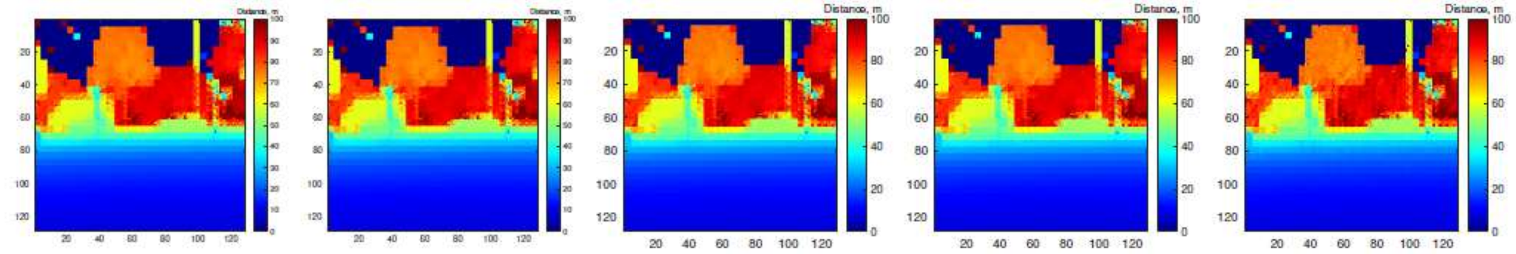
- ↓ 79.7% logic reduction
- $over$ 18 × throughput

fp – floating point
fxp – fixed point
up – unum posit

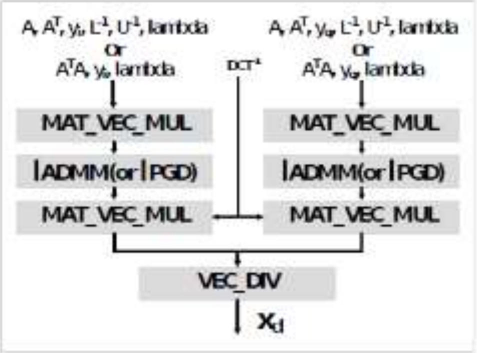








(f) ℓPGD (FP64), PSNR=0.35 dB, SSIM=0.50
(g) ℓPGD (FP32), PSNR=0.35 dB, SSIM=0.50
(h) ℓPGD (FP20), PSNR=0.35 dB, SSIM=0.49
(i) ℓPGD (FXP38), PSNR=0.35 dB, SSIM=0.50
(j) ℓPGD (UP24), PSNR=9.40 dB, SSIM=0.47

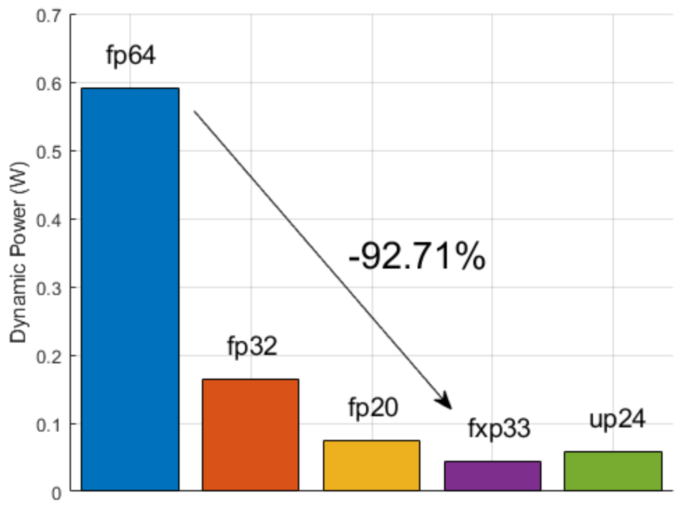


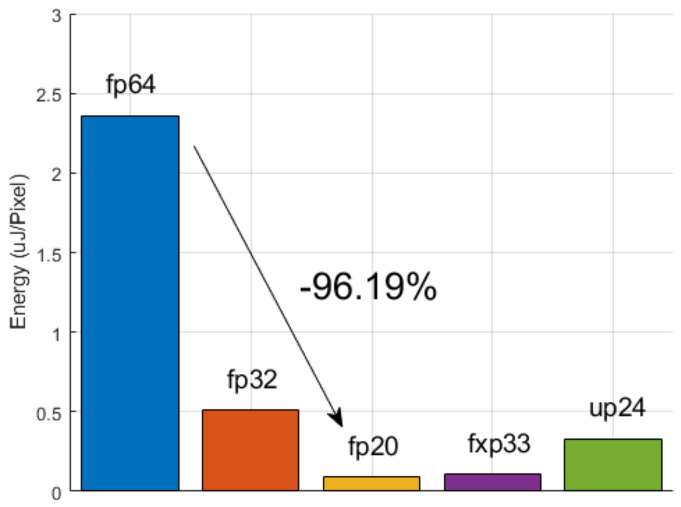
*d*Sparse - Discrete Least-Square

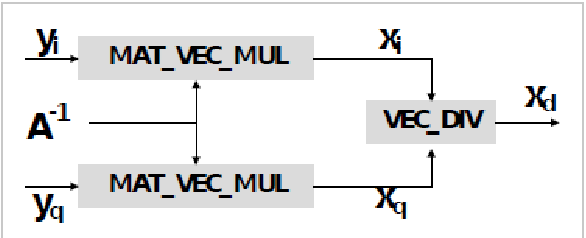
Compare to fp64:

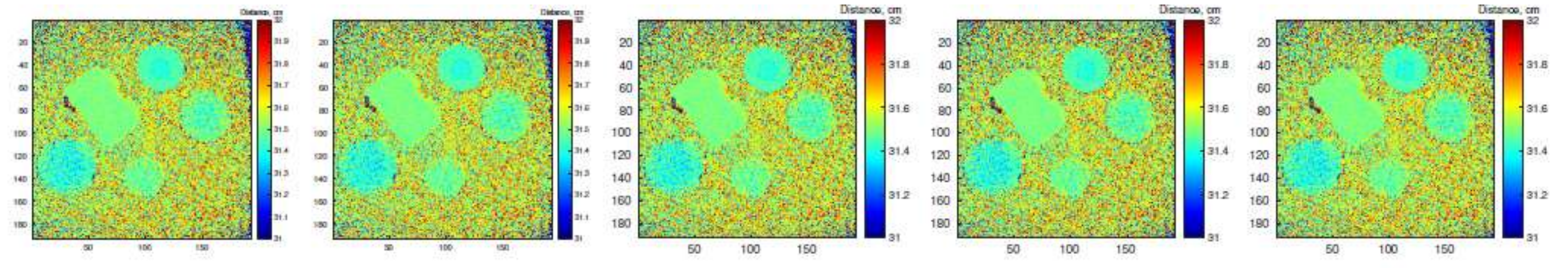
- ↓ 94.47% logic reduction
- $over\ 4 \times$ throughput

fp – floating point
fxp – fixed point
up – unum posit









(k) *d*Sparse (FP64), PSNR=Inf dB, SSIM=1
(l) *d*Sparse (FP32), PSNR=35.01 dB, SSIM=0.99
(m) *d*Sparse (FP20), PSNR=38.69 dB, SSIM=0.99
(n) *d*Sparse (FXP33), PSNR=44.24 dB, SSIM=0.99
(o) *d*Sparse (UP24), PSNR=37.21dB, SSIM=0.98

Real-Time Performance

data ratio $= (p/n) \cdot (\text{bits}/64)$ Throughput $= 16/\text{Latency}$ Energy $= (\text{Latency} \cdot \text{Dynamic Power})/16$

			Resources				Precision and Performance					
	Arithmetic	Fig.	LUT	DUP (DUP48E2)	BRAM (RAMB36)	Dynamic Power (W)	Bit Width (bits)	Data Ratio	Frequency (MHz)	Latency (ms)	Throughput Pixel/s	Energy (μJ/Pixel)
ℓADMM $p=8$ $n=16$	FP64	5(a),6(a),7(a),8(a)	10164	53	20	1.283	(1,11,52)	50.00%	390	0.48	3.33e4	38.46
	FP32	5(b),6(b),7(b),8(b)	4711	22	0	0.365	(1,8,23)	25.00%	444	0.27	5.92e4	6.17
	FP24	7(c)	3960	5	0	0.301	(1,6,17)	18.75%	475	0.25	6.4e4	4.71
	FP22	8(c)	3691	5	0	0.274	(1,6,15)	17.18%	430	0.28	5.71e4	4.8
	FP20	5(c)	3136	5	0	0.230	(1,6,13)	15.62%	472	0.25	6.4e4	3.59
	FP18	6(c)	2913	5	0	0.225	(1,6,11)	14.06%	546	0.22	7.27e4	3.09
	FXP38	6(d)	3363	18	0	0.238	(1,27,10)	29.68%	396	0.08	2e5	1.19
	FXP36	7(d)	3193	18	0	0.225	(1,25,10)	28.12%	373	0.09	1.77e5	1.27
	FXP34	8(d)	3019	18	0	0.222	(1,26,7)	28.12%	383	0.09	1.77e5	1.25
	FXP23	5(d)	2068	10	0	0.148	(1,14,8)	17.96%	433	0.08	2e5	0.74
	UP30	7(e)	17544	31	0	0.213	(1,2,1,26)	23.43%	377	0.31	5.16e4	4.13
	UP28	8(e)	17087	31	0	0.213	(1,2,1,24)	21.87%	359	0.32	5e4	4.27
	UP24	6(e)	17279	31	0	0.211	(1,2,1,20)	18.75%	356	0.33	5e4	4.23
	UP20	5(e)	17287	31	0	0.208	(1,2,1,16)	$\boxed{15.62\%}$	361	0.32	5e4	4.16
PGD $p=8$ $n=16$	FP64	5(f),6(f),7(f),8(f)	10147	33	13	1.065	(1,52,11)	50.00%	404	0.18	8.88e4	12.01
	FP32	5(g),6(g),7(g),8(g)	3454	17	0	0.256	(1,52,11)	25.00%	424	0.12	1.33e5	1.92
	FP22	8(h)	2590	5	0	0.179	((1,6,15)	**17.18%**	506	0.10	1.6e5	1.11
	FP20	5(h),7(h)	2216	5	0	0.159	(1,6,13)	**15.62%**	502	0.10	1.6e5	0.99
	FP18	6(h)	2060	5	0	0.145	(1,6,11)	$\boxed{14.06\%}$	534	0.09	1.77e5	0.81
	FXP38	6(i)	3326	36	0	0.263	(1,27,10)	29.68%	381	**0.01**	**1.6e6**	0.16
	FXP36	7(i)	3156	36	0	0.255	(1,25,10)	28.12%	396	**0.01**	**1.6e6**	0.16
	FXP34	8(i)	3050	32	0	0.237	(1,26,7)	26.56%	409	**0.01**	**1.6e6**	0.15
	FXP22	5(i)	2420	9	0	0.148	(1,14,7)	**17.18%**	405	**0.01**	**1.6e6**	$\boxed{0.09}$
	UP26	7(j),8(j)	11882	23	0	0.139	(1,2,1,22)	20.31%	382	0.13	1.23e5	1.13
	UP24	6(j)	11789	23	0	0.132	(1,2,1,20)	18.75%	382	0.13	1.23e5	1.07
	UP22	5(j)	11810	23	0	0.132	(1,2,1,18)	17.18%	386	0.13	1.23e5	1.07
dSparse $p=48$ $n=16$	FP64	5(k),6(k),7(k),8(k)	5174	22	15	0.590	(1,52,11)	300.00%	426	0.08	2e5	2.36
	FP32	5(l),6(l),7(l),8(l)	1499	10	8	0.165	(1,52,11)	150.00%	528	0.05	3.2e5	0.51
	FP20	6(m),7(m),8(m)	746	4	2.5	0.074	(1,6,13)	93.75%	516	0.02	8e5	**0.09**
	FP18	5(m)	688	4	2	0.071	(1,6,11)	84.37%	510	0.02	8e5	**0.09**
	FXP35	6(n)	442	8	4	**0.043**	(1,21,13)	164.06%	320	0.04	4e5	0.11
	FXP33	7(n),8(n)	422	8	4	**0.042**	(1,19,13)	154.68%	320	0.04	4e5	0.11
	FXP24	5(n)	286	4	4	$\boxed{0.028}$	(1,10,13)	112.50%	448	0.03	5.33e5	0.11
	UP24	6(o),7(o)	2843	11	8	0.059	(1,2,1,20)	112.50%	374	0.09	1.77e5	0.33
	UP22	8(o)	2836	11	8	0.060	(1,2,1,18)	103.12%	377	0.09	1.77e5	0.34
	UP18	5(o)	2793	11	8	**0.058**	(1,2,1,14)	84.37%	382	0.09	1.77e5	0.33

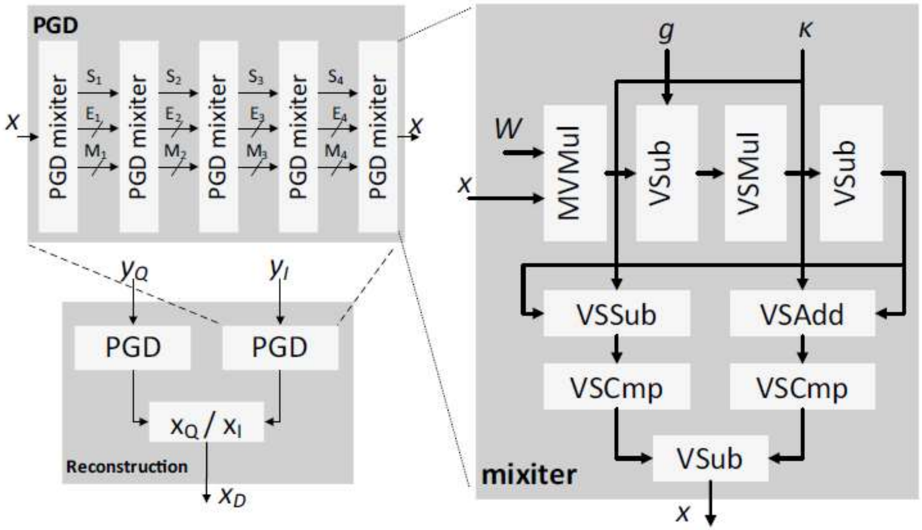
Mixed Precision 3D Depth Reconstruction

A mixed precision strategy:

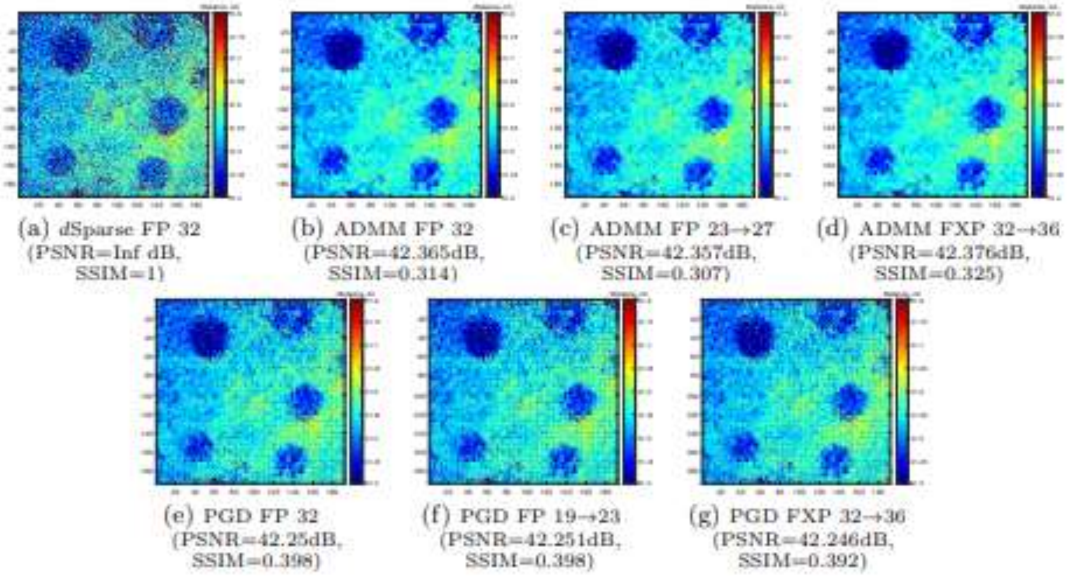
- Diverse precision across iterations
- Diverse precision for different solvers
- Automated flow from mixed precision kernel to accelerator prototype

Achieves even greater gains:

- ↓55% hardware cost reduction, and
- ↓78% power reduction compared to full precision
- ↓10~20% compared to constantly reduced precision
- ~5x throughput



$l1$ solver iterations of Mixed precision



Automatic Approximation

PID is widely used in controlled, autonomous systems

PID requires compact and low power controllers

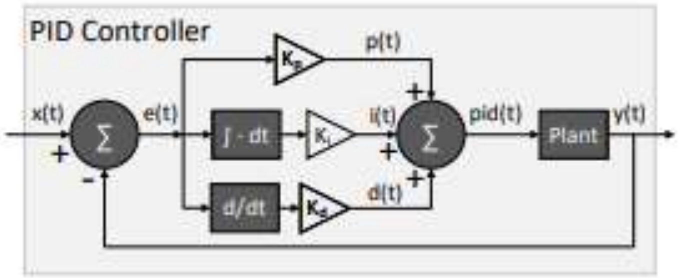
Affine Arithmetic:

$$\hat{x} = x_0 + \sum_{i=1}^{n} x_i \cdot \epsilon_i$$

in [-1,1]

Floating point: $\hat{v}_{fp} = (-1)^S \times \hat{M} \times 2^{(2^{(b\hat{w}_e-1)}-1)-\hat{E}}$

Fixed point: $\hat{v}_{fxp} = (-1)^S \times (2^{\hat{I}} + 2^{-\hat{F}})$

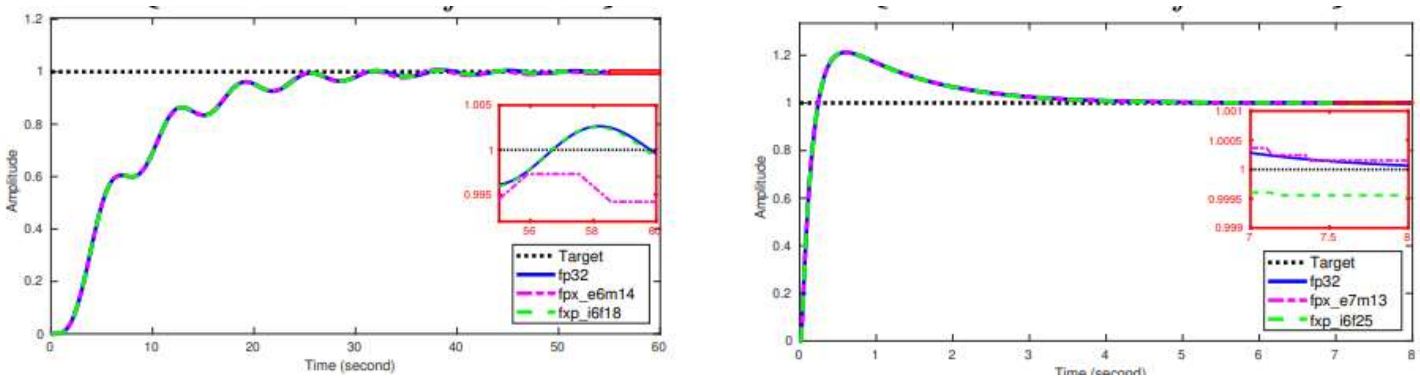


Goal: derive the precision automatically

Achievements:

- area reduction by 62%
- power reduction by 27%

Same control performance w.r.t. standard computational precision



Power Scaling Approximation

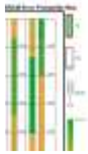
Reducing power consumption while maintaining performance considering fine-grained granularity approximation vulnerability.

$$P_{total} = P_{dynamic} + P_{static},$$

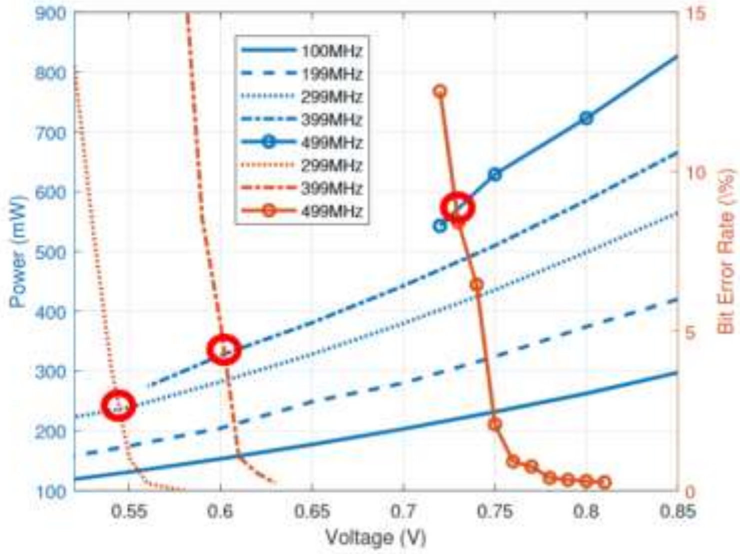
$$P_{dynamic} = \alpha \cdot C \cdot F \cdot V^2,$$

$$P_{static} = \sum I_{leakage} \cdot V,$$

- Reducing voltage and overclocking decrease power use in silicon devices without affecting the outcome
- However, this is within limits: beyond these there are significant errors in both computation and storage
- The physical locations of these errors are very hard to predict, dependent on the hardware device and routing layout



Undervolting & Overclocking:
Up to 60% power savings



Achievements

- We presented an approximate linear algebra library, which can be utilized for real-time implementation of approximate reconfigurable accelerator

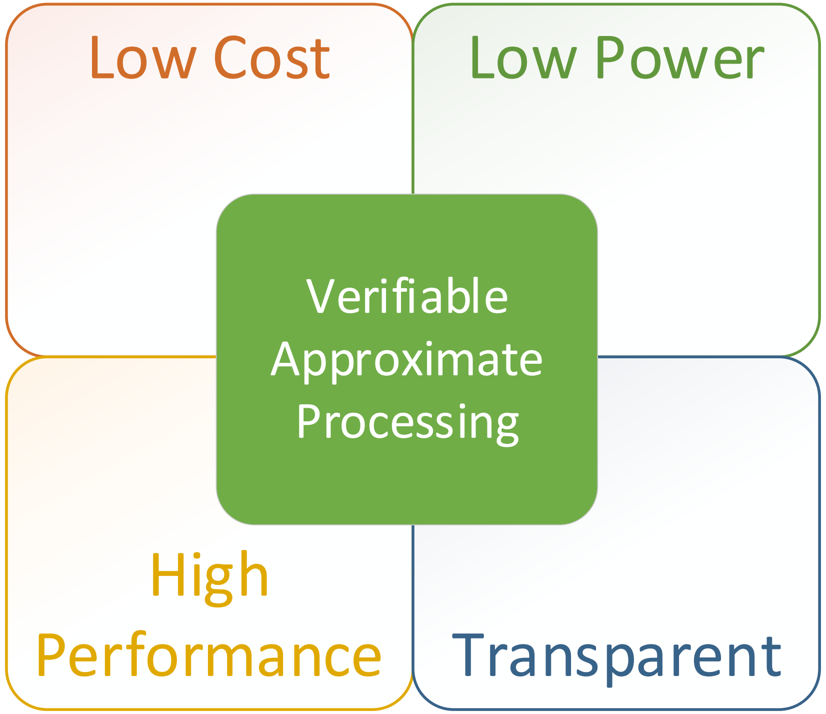
- We achieved real-time processing & low cost/power/energy, e.g. reducing latency from 0.48 ms to 0.01 ms, and significant power savings, by as much as 95%, in the best cases, and additional savings by reducing further the overall data bandwidth in the compressed sensing cases.

- It shows potential ways of mitigating the computational complexity of compressed sensing techniques, while benefiting from low power, eye-safe sparse illumination and lowering the data bandwidth throughout the computation stack.

- We explored the automated approximation through functionality self-validation approach and native hardware approximation through power scaling.

- This demonstrates a pathway to dedicated hardware logic design for resource constrained devices.

On-Going and Future Directions



- **Top to bottom stack analysis**: incorporate errors that occur from hardware/software approximations into the analyses of algorithm to obtain performance guarantees.
- **Theoretical Performance Analysis**: Error modeling, theoretical analysis, and verification of typical convex optimization algorithms to different applications.
- **Power Scaling**: approximate the computation by tuning clock frequency and voltage of an FPGA (Zynq Ultrascale). Considerable analysis is required to assess whether the actual errors incurred match those predicted by the theory.
- **Dynamic scheduling/allocation**: demonstrate approximation by reduced precision and power scaling in a dynamic scheduling framework. Unlike single objective optimisation, one may select an attentive multi-objective policy.
- **Extend Approximate Linear Algebra library**: this would allow new algorithms to be rapidly prototyped in a tool bench.