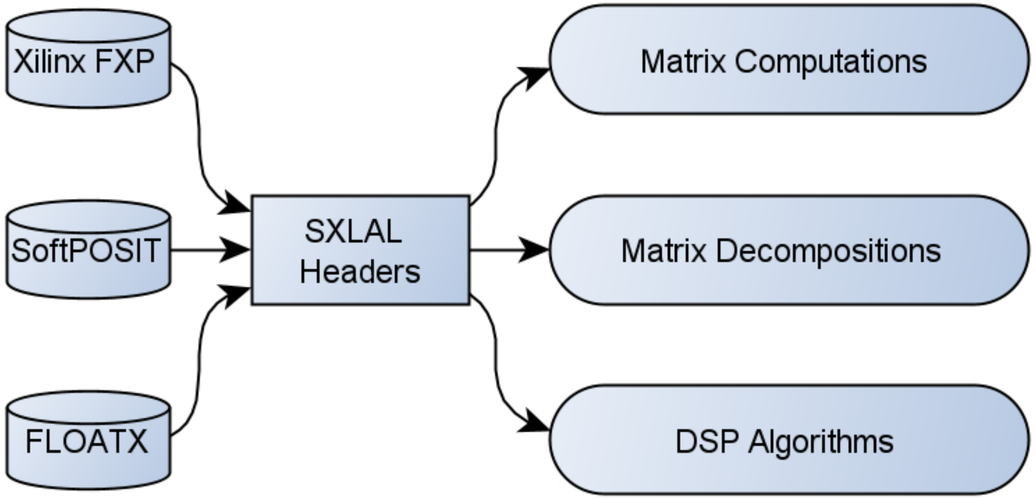
Synthesizable Approximate Linear Algebra Library (SXLAL)

Yun Wu

08.11.2022

Features

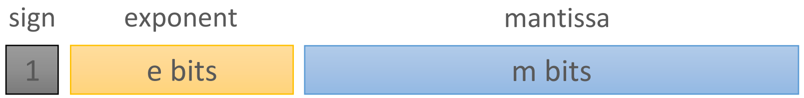
- Header-only library without a compiled component
- Various computational precisions support for different arithmetic types
- Synthesizable through Xilinx High Lever Synthesis (HLS)



Supporting Arithmetic

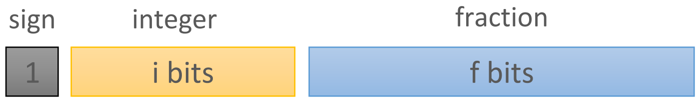
- IEEE 754 floating point arithmetic

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



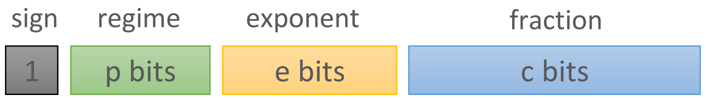
- Q format fixed point arithmetic

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



- Type III Unum – Posit

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



Supporting Linear Algebra

Matrix types:

- Real – general (nonsymmetric) real
- Complex – general (nonsymmetric) complex
- 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

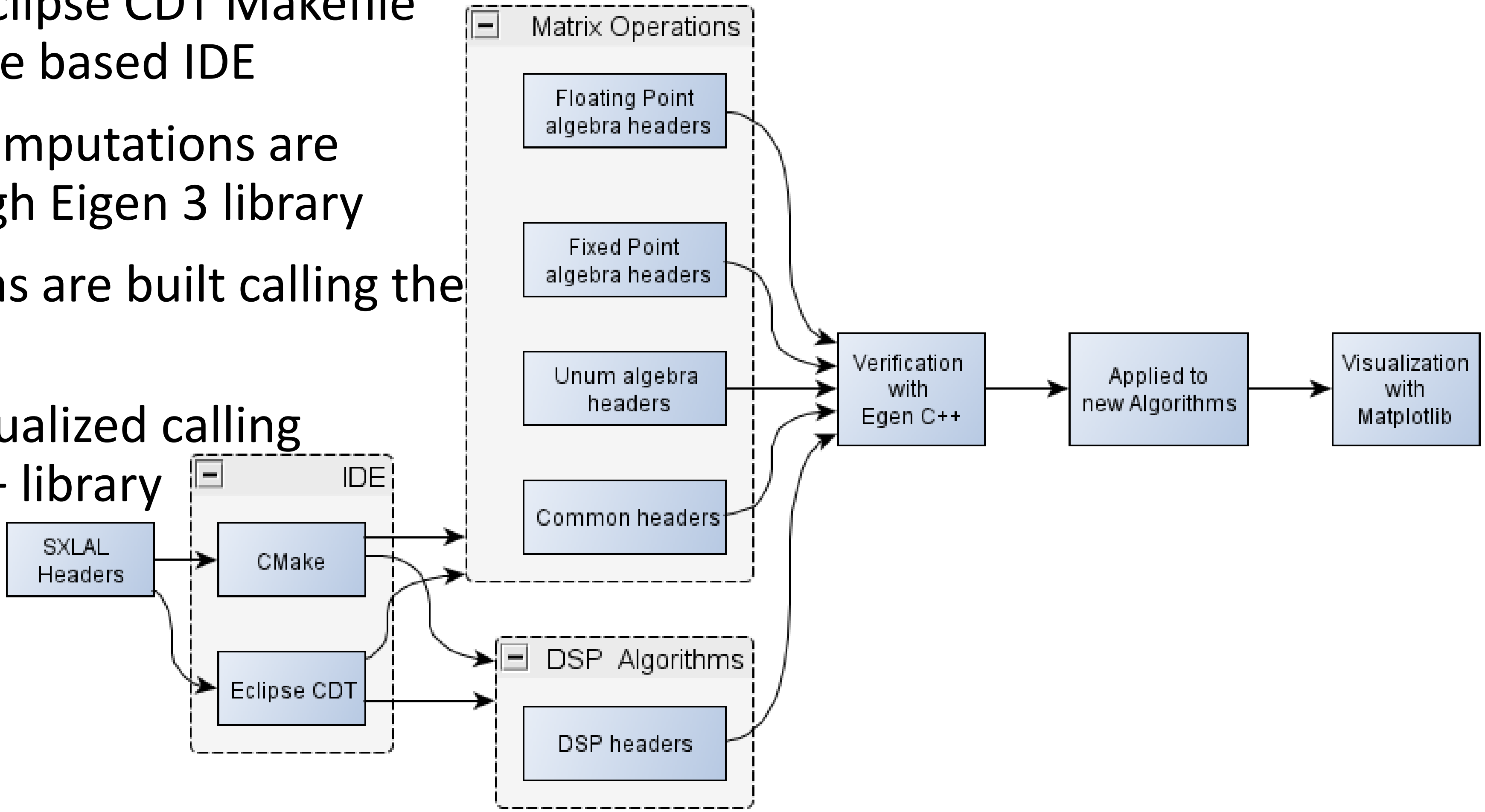
	Real	Complex	SPD	HPD	SY	HE	BND	BA	TF	OF	EVP	SVD	GEVP	GSVD
Approximate Linear Algebra Library (XLALib)	Yes	No	No	No	No	No	No	Yes	Yes	Yes	No	No	No	No

Prerequisites

- OS: Linux distribution based on Debian (e.g. Ubuntu 2020.4)
- IDE: Eclipse CDT or CMake
- System libraries: python python-numpy python-matplotlib libboost-all-dev libeigen3-dev fftw-dev
- External libraries: FLOATX, Xilinx HLS Fixed Point, SoftPOSIT, Matplotlib C++, Eigen C++

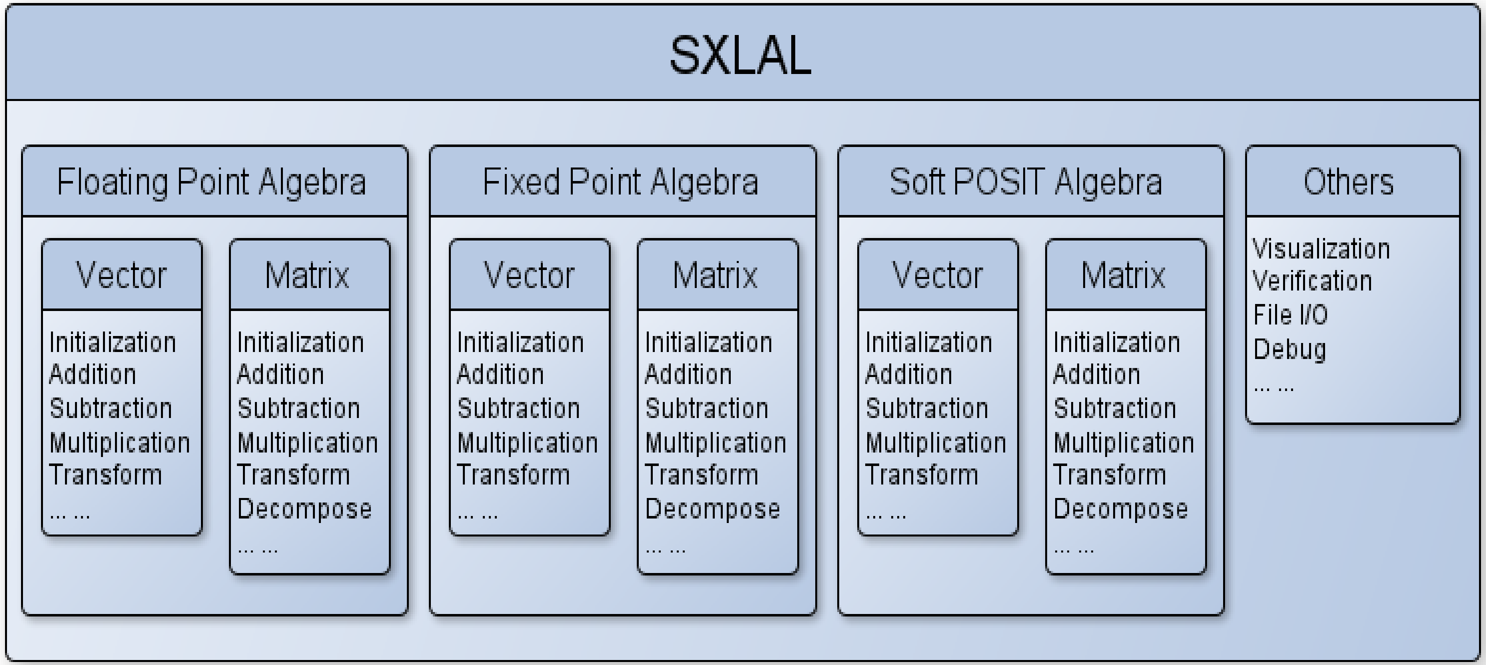
Development & Deployment

- Using either Eclipse CDT Makefile or other CMake based IDE
- The algebra computations are verified through Eigen 3 library
- New algorithms are built calling the header library
- Results are visualized calling Matplotlib C++ library



Algebra Classes

- Separate header classes for different arithmetic types
- Algebra functions within each header class
- Helper classes assisting visualization, verification, debug and File I/O, etc.



Algebra Functions

- E.g. Matrix Multiplication

```
float A[4][4], B[4][4], C[4][4];
MAT_MUL <float, 4, 4, 4> (A, B, C);
```

Function call flow:

- Use the algebra function name
- Define arithmetic type and data dimension
- The computation follow the input sequence
- Last argument is the output

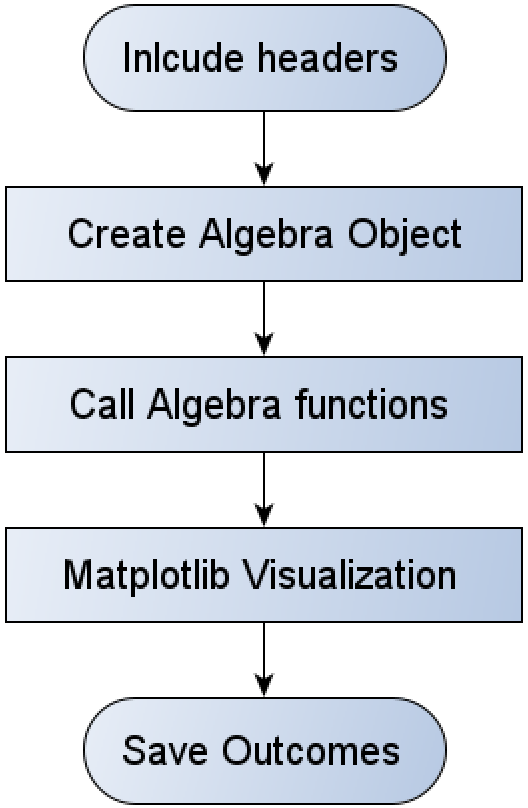
function name   function parameters   function arguments

MAT_MUL <T, M, N, P> (A, B, C)

arithmetic type   matrix dimensions   C = A x B, T A[M][N], T B[N][P], T C[M][P]

Library Usage

- Use the header-only library as normal C++ header
- Define specific objects using different arithmetic algebra classes
- Use the algebra functions in the defined object with specific arithmetic type and precision
- Use Matplotlib to visualize the results (Optional)
- Save the results using File I/O classes (Optional)



Example: Proximal Gradient Descent (1)

- The header files of algebra operation using different arithmetic:

```
#include "fpt_algebra.hpp“   // Headers for custom floating point linear algebra
#include "xfxpt_algebra.hpp“   // Headers for fixed point linear algebra
#include "softposit_algebra.hpp“   // Headers for unum linear algebra
```

Include header-only library

- Define the data structure

```
#define ROW 64 // matrix row number
#define COL 64 // matrix column number
#define DIAG (COL<ROW ? COL : ROW) // diagonal matrix size
```

Define data dimension

- Create the object to call algebra functions

```
Float_Point_Algebra Float_Point_Algebra_obj; // Algebra object using floating point arithmetic
Fixed_Point_Algebra Fixed_Point_Algebra_obj; // Algebra object using fixed point arithmetic
SoftPosit_Algebra SoftPosit _Algebra_obj; // Algebra object using unum arithmetic
```

Create algebra objects

Example: Proximal Gradient Descent (2)

- Each iteration calling the algebra functions from the header library

```
// Save previous point
Float_Point_Algebra_obj.VEC_EQ<float, DIAG>(x_k_vec, x_k_plus1_vec);

// Compute gradient
float tmp_mul_vec1[DIAG];
Float_Point_Algebra_obj.MAT_VEC_MUL<float, DIAG, DIAG>(
                Amatrix_c, x_k_vec,     tmp_mul_vec1 );
Float_Point_Algebra_obj.VEC_ADD<float, DIAG>( tmp_mul_vec1, bvector_c,
                                                grad_g );

// new decent point
float new_point[DIAG];
float tmp_mul_vec2[DIAG];
Float_Point_Algebra_obj.VEC_SCALAR_MUL<float, DIAG>( grad_g, 1/L_c,
                                                         tmp_mul_vec2 );
Float_Point_Algebra_obj.VEC_SUB<float, DIAG>( x_k_vec, tmp_mul_vec2,
                                                new_point );
```

Calling algebra functions

```
// Proximal projection
float rndnoise[DIAG];
float a_ones_vec[DIAG];
float minus_a_ones_vec[DIAG];
Float_Point_Algebra_obj.ONES_VEC<float, DIAG>( a_ones_vec );
Float_Point_Algebra_obj.ONES_VEC<float, DIAG>( minus_a_ones_vec );
float tmp_min[DIAG];
float tmp_max[DIAG];
float tmp_mul_vec3[DIAG];
Float_Point_Algebra_obj.RND_VEC<float, DIAG>( rndnoise );
Float_Point_Algebra_obj.VEC_SCALAR_MIN<float, DIAG>(new_point, BOX_CONST, tmp_min);
Float_Point_Algebra_obj.VEC_SCALAR_MAX<float, DIAG>(tmp_min, -BOX_CONST, tmp_max);
Float_Point_Algebra_obj.VEC_SCALAR_MUL<float, DIAG>( rndnoise, error_std,
                                                         tmp_mul_vec3);
Float_Point_Algebra_obj.VEC_ADD<float, DIAG>( tmp_max, tmp_mul_vec3,
                                                 x_k_vec);

// check early termination constraint
float norm1, norm2;
float tmp_sub_vec[DIAG];
Float_Point_Algebra_obj.VEC_SUB<float, DIAG>( x_k_vec, x_k_plus1_vec,
                                                 tmp_sub_vec);
Float_Point_Algebra_obj.VEC_NORM<float, DIAG>(tmp_sub_vec, norm1);
Float_Point_Algebra_obj.VEC_NORM<float, DIAG>(x_k_vec, norm2);
if( norm1<= EPS_STOP ){
        break;
}
```

Example: Proximal Gradient Descent (3)

- Include matplotlibcpp library

```
#include "matplotlibcpp.hpp"
```

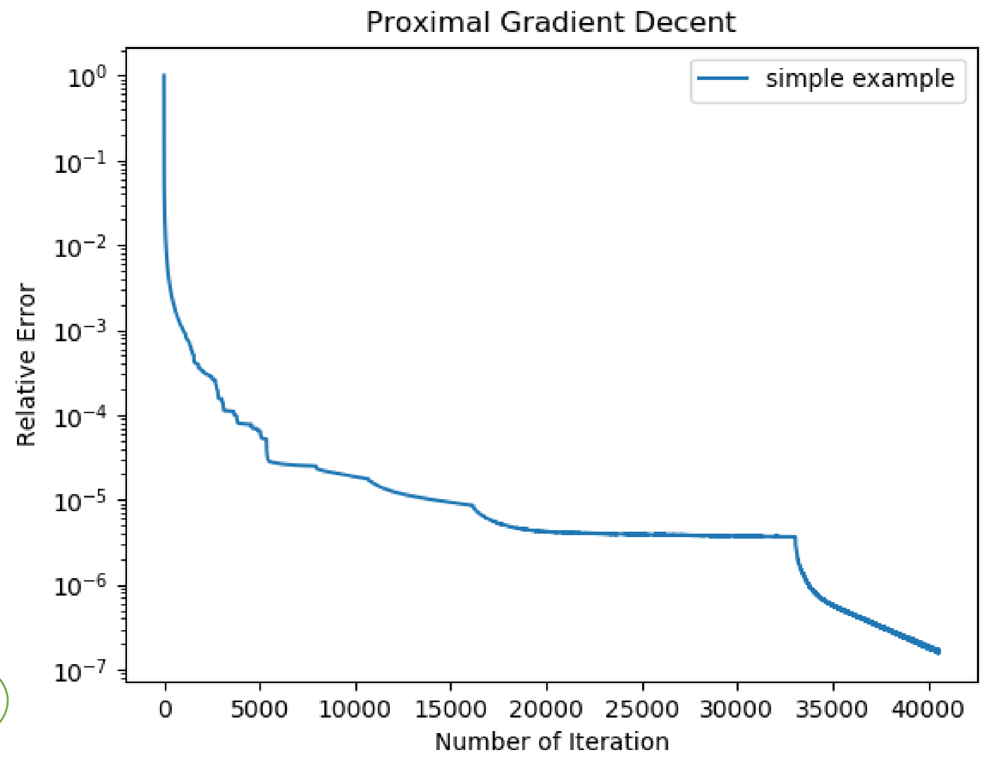
- Define matplotlibcpp namespace

```
namespace plt = matplotlibcpp;
```

- Plot using matplotlib functions

```
#ifdef PLOT_FIGURE
        // Matplotlib plotting
        plt::named_semilogy( "simple example", x, y );
        plt::title("Proximal Gradient Decent");
        plt::xlabel("Number of Iteration");
        plt::ylabel("Relative Error");
        plt::legend();
        plt::save(figurename);
#ifdef SHOW_FIGURE
        plt::show();
#endif// endif SHOW_FIGURE
#endif// endif PLOT_FIGURE
```

Calling Matplotlib to draw figure



Plot figure using random input and floating point arithmetic

Example: Proximal Gradient Descent (4)

- Custom floating point, e.g.

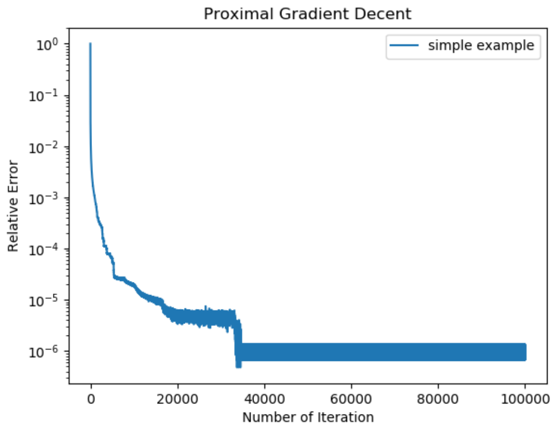
```
using fpx = flx::floatx<5, 10>;
fpx A[4][4], B[4][4], C[4][4];
MAT_MUL  <fpt, 4, 4, 4>  (A, B, C);
```

- Custom fixed point, e.g.

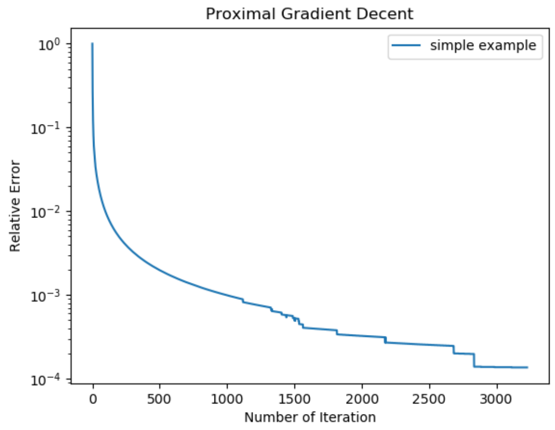
```
typedef ap_fixed<32,20> fxp;
fxp A[4][4], B[4][4], C[4][4];
MAT_MUL  <fxp, 4, 4, 4>  (A, B, C);
```

- Custom Unum soft POSIT, e.g.

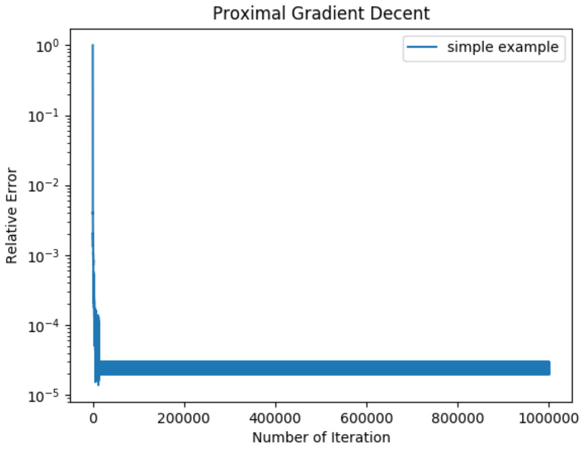
```
posit_2_t A[4][4], B[4][4], C[4][4];
MAT_MUL <posit_2_t, 4, 4, 4, 16> (A, B, C);
```



Custom floating point 16 bits



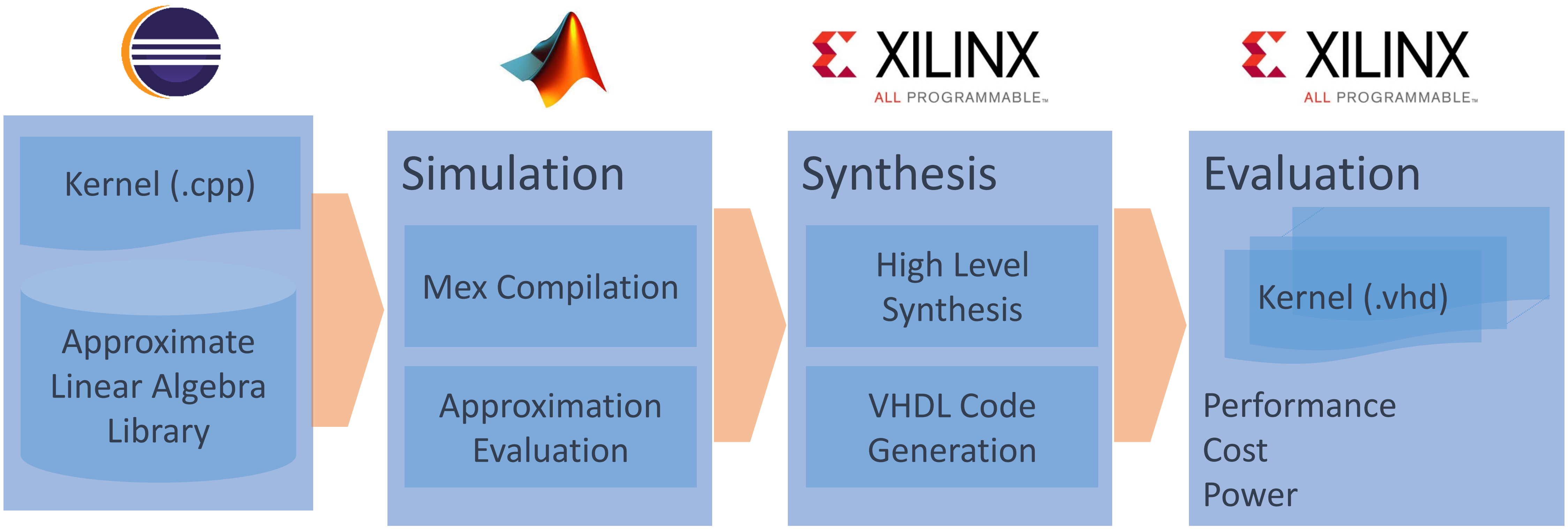
Custom fixed point 28 bits



Custom Unum soft POSIT 16 bits

Hardware Prototyping

- C++ kernel → MEX verification → Xilinx HLS → HDL IP verification



Applied Applications

- Convex Optimization Solvers:
  - Proximal Gradient Descent
  - Alternating Direction Method of Multipliers
- Applications:
  - Lidar Depth reconstruction
  - Model Predictive Control (MPC)
  - Proportional–Integral–Derivative (PID) controller

Key Publications

1. Efficient Reconfigurable Mixed Precision ℓ1 Solver for Compressive Depth Reconstruction. Journal of Signal Processing Systems 94, 1083–1099 (2022).
2. Energy Efficient Approximate 3D Image Reconstruction. IEEE Transactions on Emerging Topics in Computing (Early Access)

Future Developing

- EVP – eigenvalue problems
- SVD – singular value decomposition
- General DSP algorithms
- Complex value linear algebra
- Tensor operations