Linear Algebra Benchmarks on C66x

Oct 2017

C66x Core Advances Summary

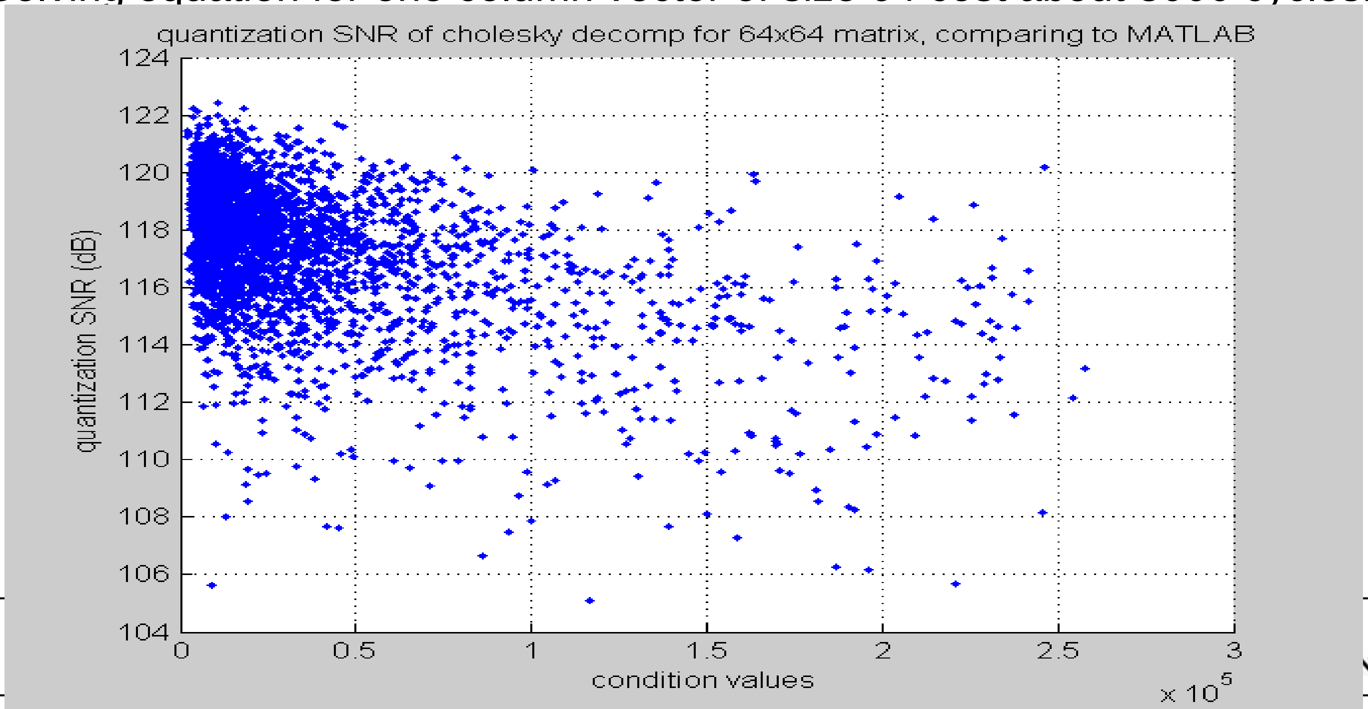
- ISA- 100% backward compatible
  - Code running on C674x can be compiled and run directly on C66x
- Optimized for complex arithmetic and linear algebra (matrix processing)
- Improved vector processing capability (SIMD)
- 4 single precision floating-point multiplies per cycle
- Fully pipelined double precision floating point multiplies
- Reduce latency of double precision multiply from 10 to 4
- Improved 32-bit fixed-point operations, such as 8 32x32 fixed point multiplies: CMPY32R1, QSMPY32R1, etc
- Improved 32-bit fixed-point operations, such as 32 16x16 fixed point multiplies: CMATMPY, CMATMPYR1, CCMATMPY, CCMATMPYR1, etc.

C66x Floating-point Capabilities Summary

- Same single-precision operations per cycle as C64x+ 16-bit integer operations per cycle.
- Capability of C66x single-precision floating-point multiplication and subtraction/addition is:
  - Same capability of 16-bit integer operation in C64x+ core:
  - Same as 32-bit integer operations in C66x core
  - 4 times capability of 32-bit integer operation in C64x+ core
- Multiplication:
  - 8 single-precision multiplication per cycle: CMPYSP and QMPYSP can calculate 4 pairs of single–precision multiples per .M unit per cycle.
- Addition/subtraction:
  - 8 single-precision addition/subtraction per cycle: DADDSP and DSUBSP can add/sub 2 floats and they can be executed on both .L and .S units.
- Conversion between floating-point and fixed-point:
  - 8 single-precision to integer conversion, or 8 integer to single-precision conversion per cycle: DSPINT, DSPINT, DINTHSP and DSPINTH converts 2 floats to 2 integers per cycle and it can be executed on both .L and .S units.
- Division:
  - C66x also offers single cycle single-precision 1/x and 1/sqrt(x) calculation.

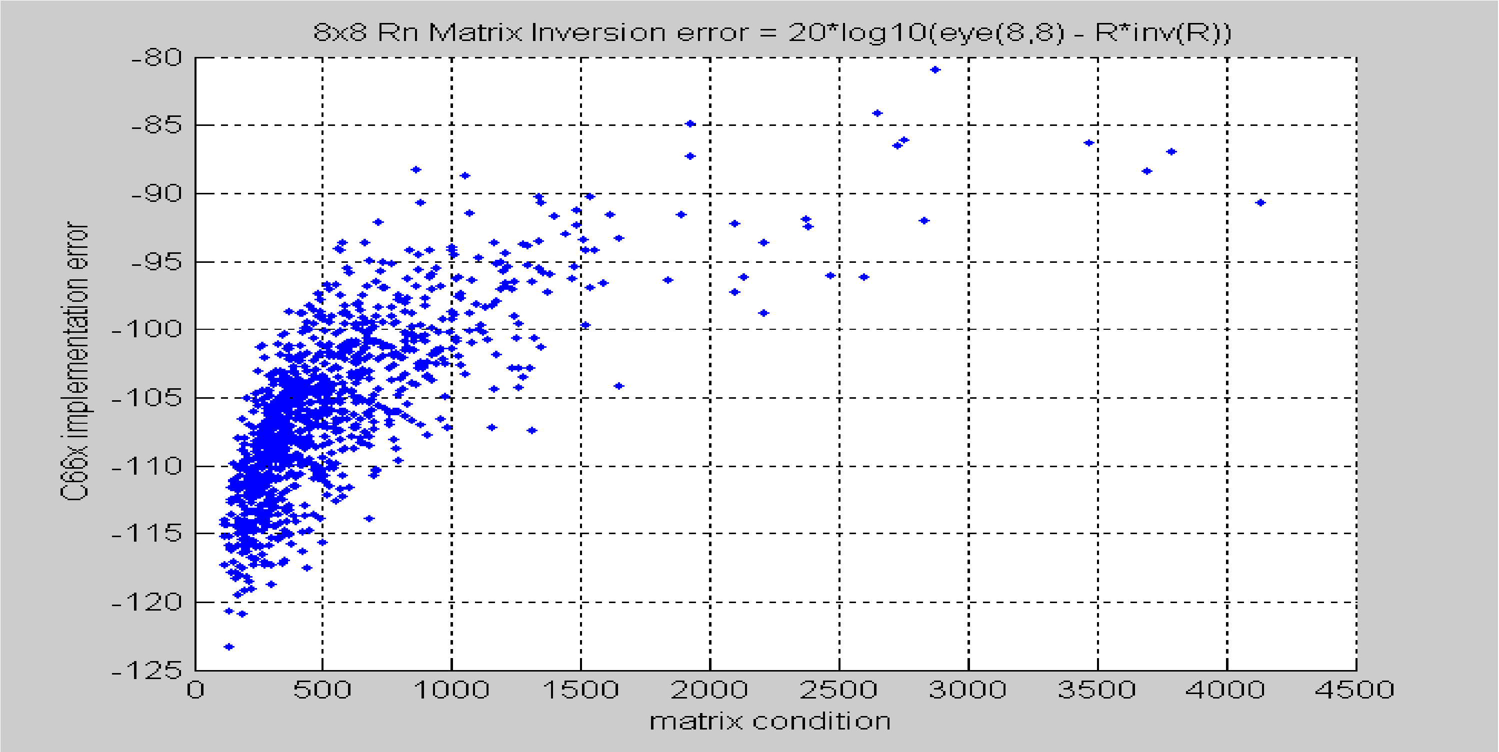
Cholesky Decomposition

- 32-bit fixed-point and single precision floating-point implementation take about the same cycles:
  - 64x64 matrix: 257k cycles
  - 8x8 matrix: 2300 cycles
- Inversion of upper/lower triangular matrix inversion can be done with upper/lower triangular solver, which takes about 250k cycles to get the full inversion of the 64x64 size matrix.
  - There might be other faster way, but we have not explored (mainly because matrix inversion is mostly in context of solving linear equations)
  - Solving equation for one column vector of size 64 cost about 8000 cycles.



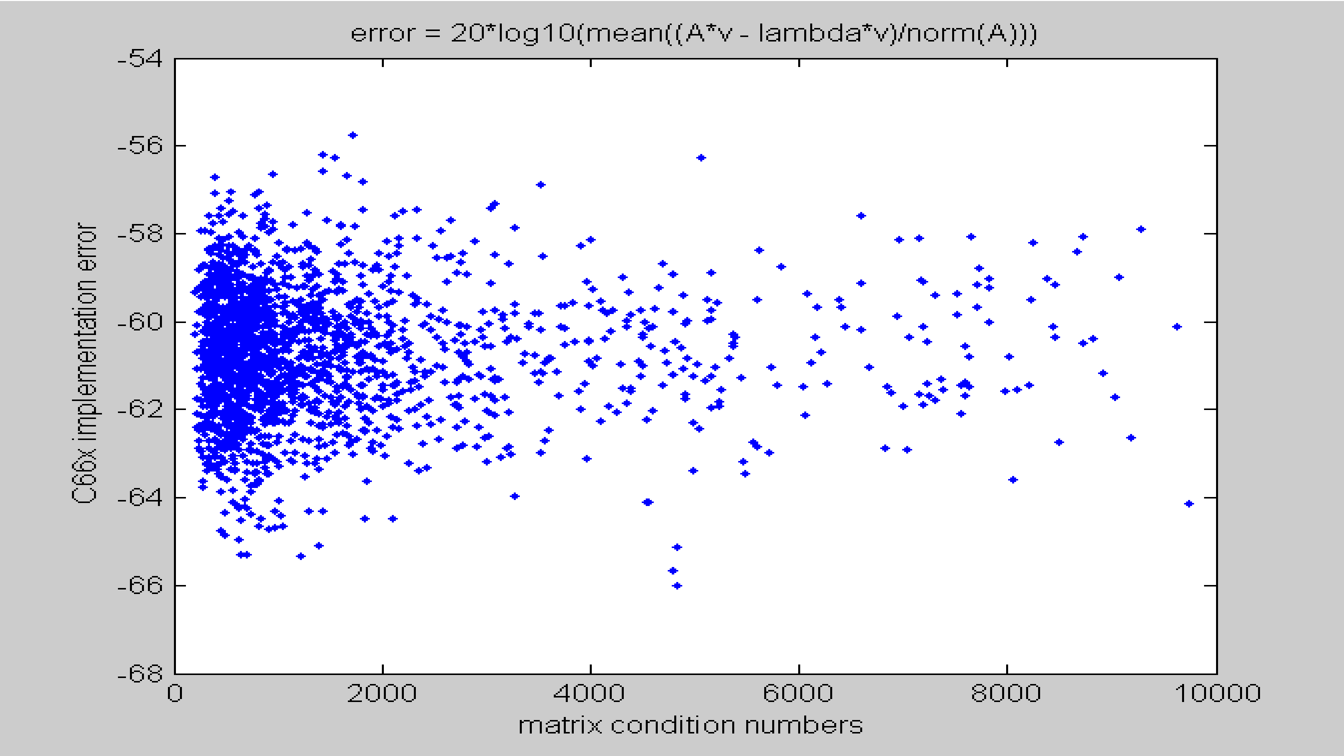
Close Form Matrix Inversion

- Assuming is positive semi-definite complex matrix
- For size up to 8x8, floating-point implementation
  - 3x3: 80 cycles for single inversion, 20 cycles/matrix for multiple inversions (with extra memory needed for intermediate results)
  - 4x4: 180 cycles for single inversion, 45 cycles/matrix for multiple inversions (with extra memory needed for intermediate results)
  - 8x8: 1500 cycles for single inversion, 750 cycles/matrix for multiple inversions (with extra memory needed for intermediate results)



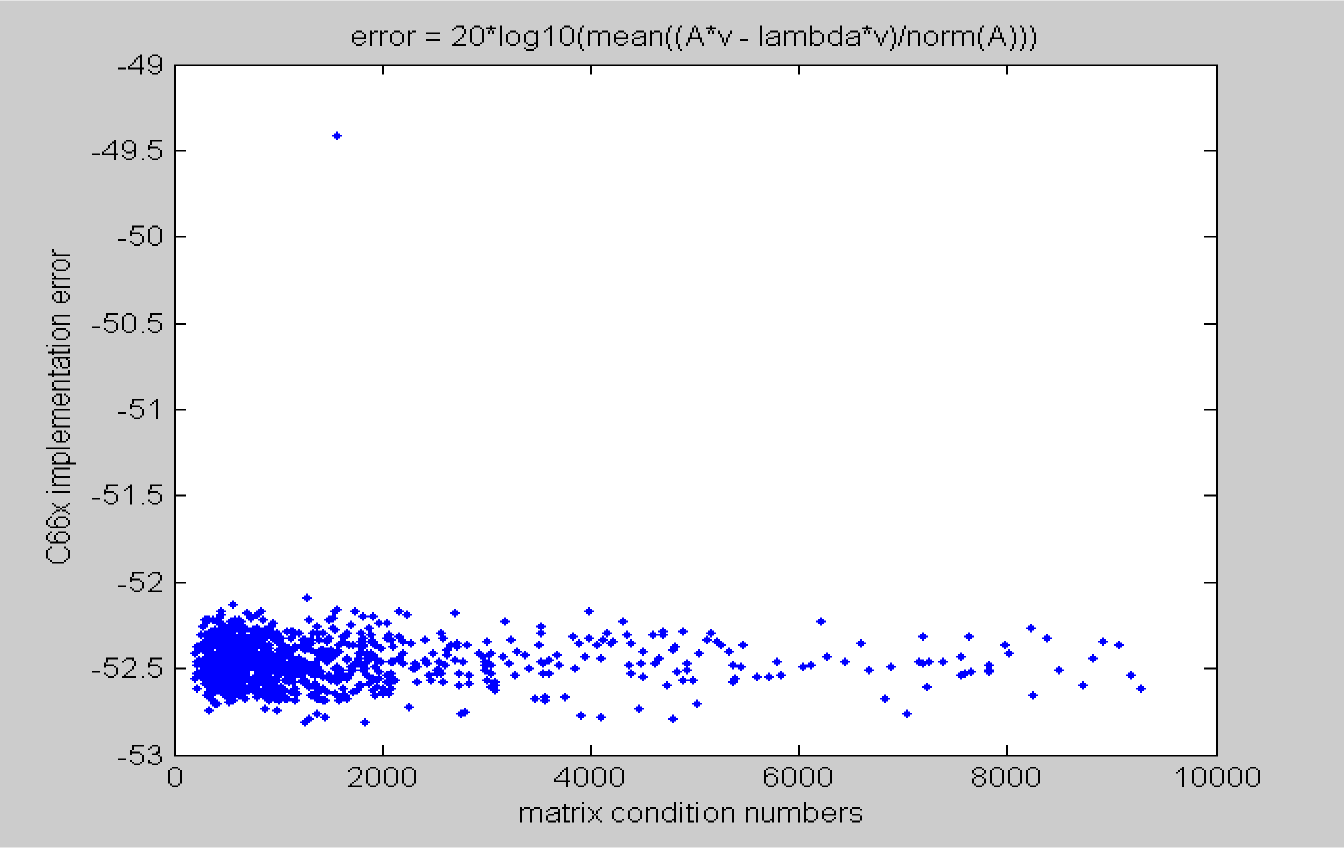
EVD: Mantan's Method

- Find the primary eigenvalue and corresponding eigenvector
- Using floating-point implementation
- Cycles:
  - 64x64 matrix: 119k cycles
  - 8x8 matrix: 3600 cycles



EVD: Jacobi Method

- Full EVD: Find all eigenvalues and corresponding eigenvectors
- Using floating-point implementation
- Cycles:
  - 64x64 matrix: 9 million cycles
  - 8x8 matrix: 16k cycles



Matrix Multiplication

- For 16-bit fixed-point complex matrix multiplication, C66x has an instruction CMATMPYR1 than can calculate two [2x2] x [2x1] complex matrix by vector multiplication in 1 cycle. Any even size matrix multiplication or matrix by vector multiplication can be broken in to smaller block to take advantage of this instruction.
  - Example of [8x128]x[128x8] 16-bit fixed-point complex matrix multiplication takes ~1500 cycles (with loop overhead).
- For single-precision floating point complex matrix multiplication, C66x has an instruction CMPYSP that can calculate two complex floating point multiplication in 1 cycles.
  - Example of [8x128]x[128x8] single precision floating-point complex matrix multiplication takes ~4500 cycles (with loop overhead).