PiotrLuszczek

Half Precision Benchmarking for HPC

S7676

Major Floating Point Formats from IEEE 754 (2008)

Precision	Width	Exponent bits	Mantissa bits	Epsilon	Max
Quadruple	128	15	112	$O(10^{-34})$	1.2×10^{4932}
Extended	80	15	64	$O(10^{-19})$	
Double	64	11	52	$O(10^{-16})$	1.8×10^{308}
Single	32	8	23	$O(10^{-7})$	3.4×10^{38}
Half*	16	5	10	$O(10^{-3})$	65504
*Only storage format is specified					

Programming Data Types: C/C++ and Fortran

- long double
  - 80 or 128 bits
- double
  - 64 bits
- float
  - 32 bits
- __half (~~short float~~)
  - 16 bits
- __half2 (cuda_fp16.h)
  - 2x16 bits

- real*16
  - 128 bits
- real*8
  - 64 bits
- real*4
  - 32 bits
- real*2
  - 16 bits

FP16 Hardware (Current and Future)

- AMD
  - MI5, MI8, MI25
- ARM
  - NEON VFP FP16 in V8.2-A
- Intel
  - Xeon CPUs (vectorized conversions)
- NVIDIA
  - Pascal: P100, TX1, TX2, ...
  - Volta: Tensor Core
- Supercomputers
  - TSUBAME 3.0
  - Tokyo Tech
  - ...
- Cloud
  - Google with P100 (coming soon)
  - Azure: Pascal debut in 2017

Applications Using FP16

- Machine Learning
  - Deep Neural Networks
  - Visualization and image processing (OpenVZ)
- Linear Algebra
  - Eigen
  - University of Tennessee libraries and projects
- Molecular dynamics
  - Gromacs

Iterative Refinement

- In exact arithmetic
  - $x_1 \leftarrow x_0 + A^{-1}(b - Ax_0)$
- In finite precision A^{-1} is not available due to
  - Round-off error
  - Lower-precision LU factors
- In practice, Richardson Iteration is often used
  - $x_{k+1} \leftarrow x_k + \widetilde{A}^{-1}(b - Ax_k)$
  - Convergence depends on the spectrum
    - Textbook result wrt. $I - \widetilde{A}^{-1}A$

Classic Iterative Refinement Implementation

- Linear system Ax=b may be solved through LU factorization
  - L, U, P ← lu_factor(A)
  - y ← L \ Pb
  - x ← U \ y
  - r ← b - Ax (use higher precision to accumulate)
  - z ← U \ L \ P * r
  - x_{final} ← x+z (use higher precision)
- All operations performed in the same floating-point precision

Mixed-Precision Iterative Refinement

- Linear system in 64-bit precision Ax=b may be solved through LU factorization in 16-bit precision:
  - L,U, P ← lu(A) (n^3) (16 bits)
  - y ← L \ Pb (n^2) (16 bits)
  - x ← U \ y (n^2) (16 bits)
  - r ← b - Ax (n^2) (64 bits)
  - z ← P L \ U \ r (n^2) (16 bits)
  - x_{final} ← x+z (n) (64 bits)
- Requirement:
  - Matrix A must be well conditioned in 16 bits:
    - $\kappa(A) < 10^5$

Early Error Analysis Results

- Standard backward stability

$$(A+E)x=b \quad where \quad E\leq\phi(n)\epsilon\|A\|$$

- Need to generalize to two machine precision: 16 and 64

$$\lim_{k\to\infty}\|x_0-x_k\|=ForwardError(\epsilon_{16,}\epsilon_{64})$$

$$\lim_{k\to\infty}\frac{\|b-Ax_k\|}{\|A\|\cdot\|x_k\|}=BackwardError(\epsilon_{16,}\epsilon_{64})$$

- Details: see paper and tech report
- Summary: it works if matrix cooperates

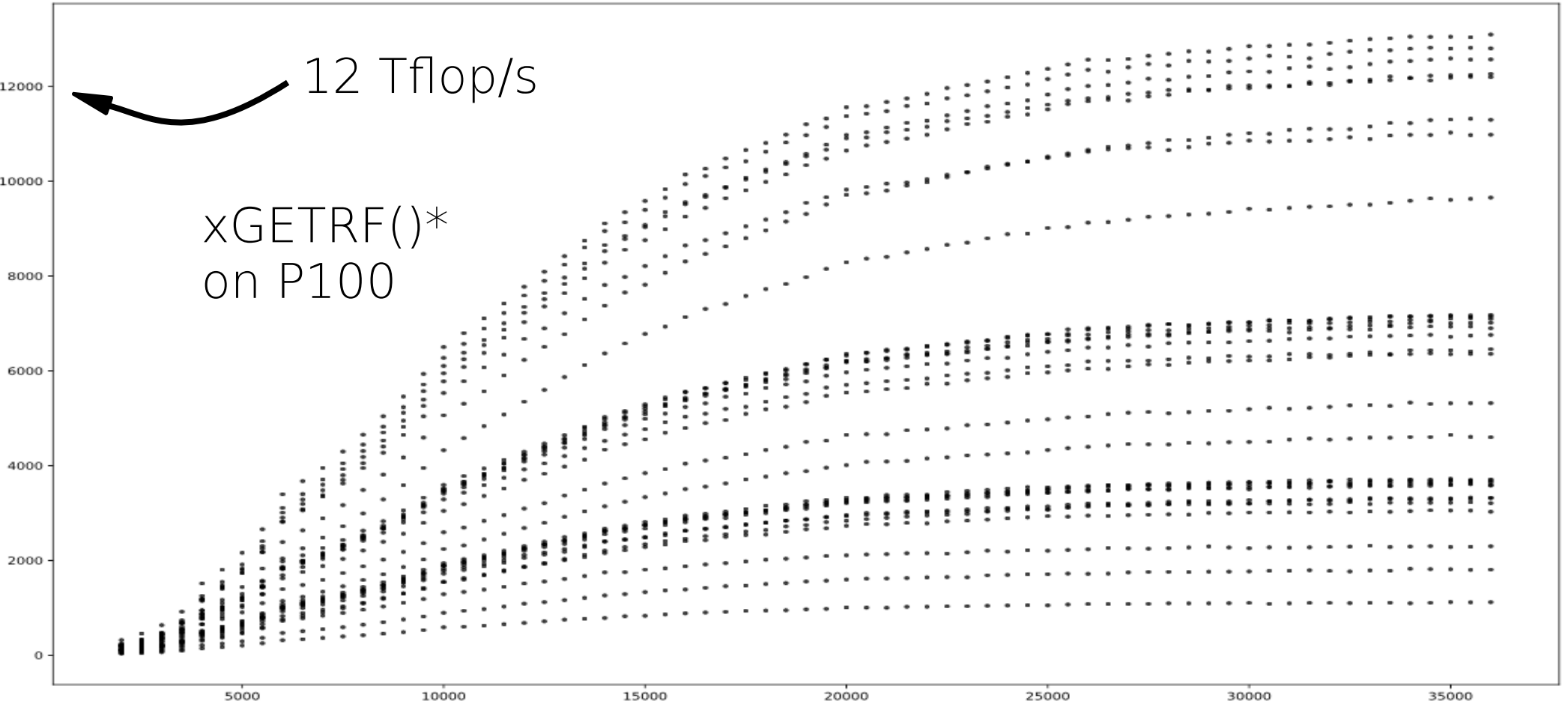
Hardware/Software Support: Assembly and Intrinsics

- x86
  - CVTSH_SS, CVTSS_SH
  - emmintrin.h
    - _cvtss_sh(), _cvtsh_ss()
  - f16cintrin.h → x86intrin.h
    - _mm_cvtph_ps(), _mm_cvtps_ph(), _mm256_cvtph_ps(), _mm256_cvtps_ph()
- PTX
  - cvt.f16.*
  - fma.f16x2
- ARM
  - vld1_f16, vst1_f16, vcvt_f16_f32

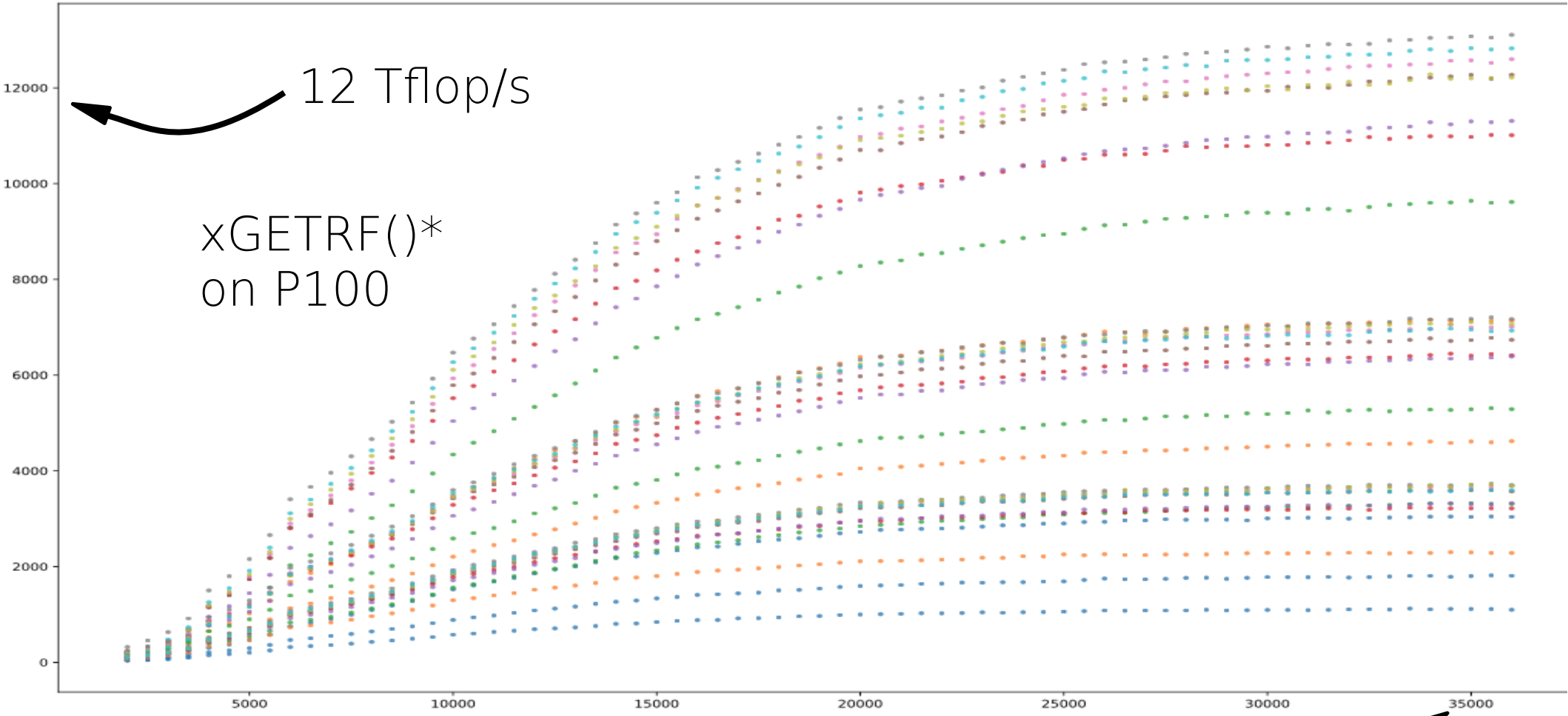
In High-Level Programming Environments

- Julia
  - A = zeros(Float16, N, N); b = zeros(Float16, N,N);
  - A[:,:] = randn(N, N); b[:,:] = randn(N,1);
  - x = A \ b; # works OK
- Python
  - numpy.float16
  - linalg.solve(randn(N,N,float16), randn(N,1,float16))
  - TypeError: array type float16 is unsupported in linalg
- MATLAB
  - Must use MEX files

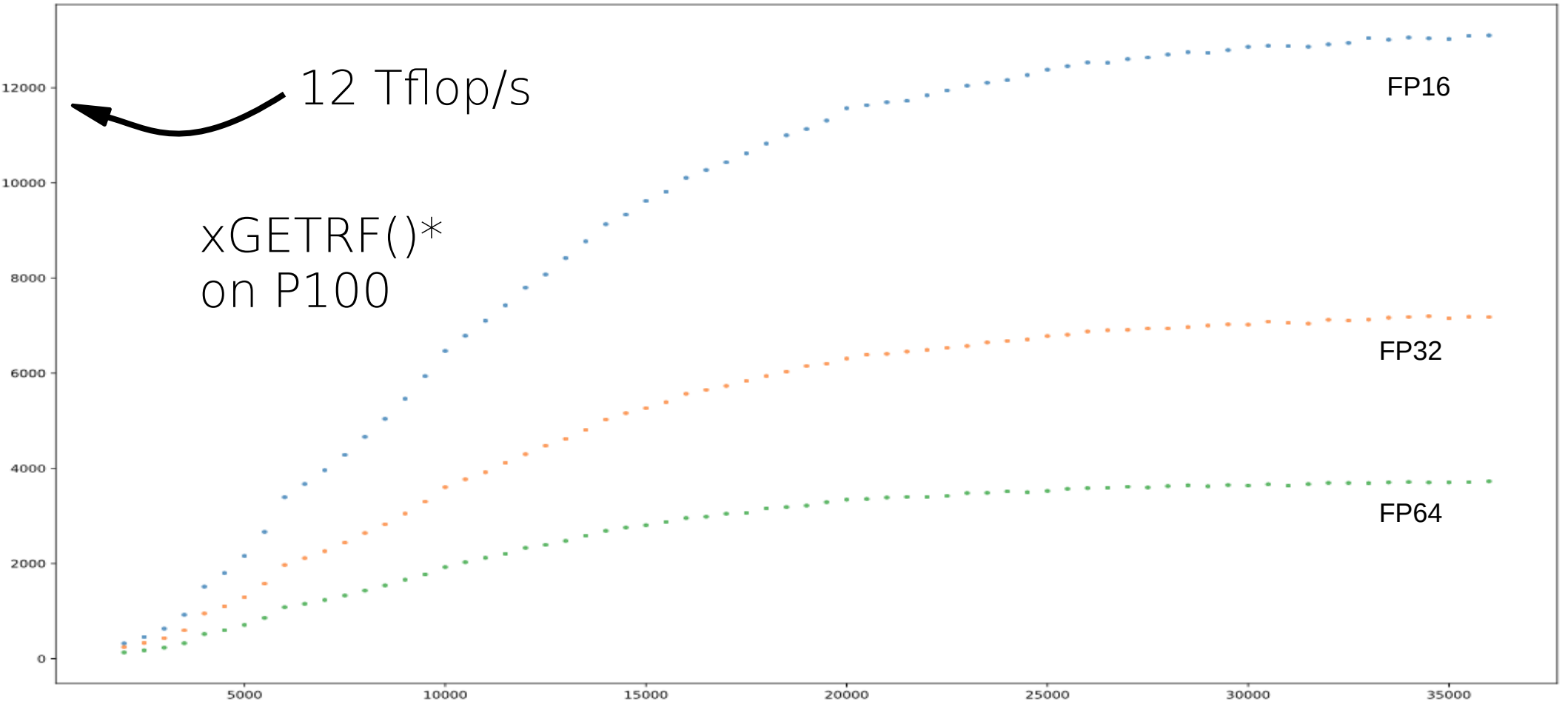
Autotuning with FP16, FP32, and FP64



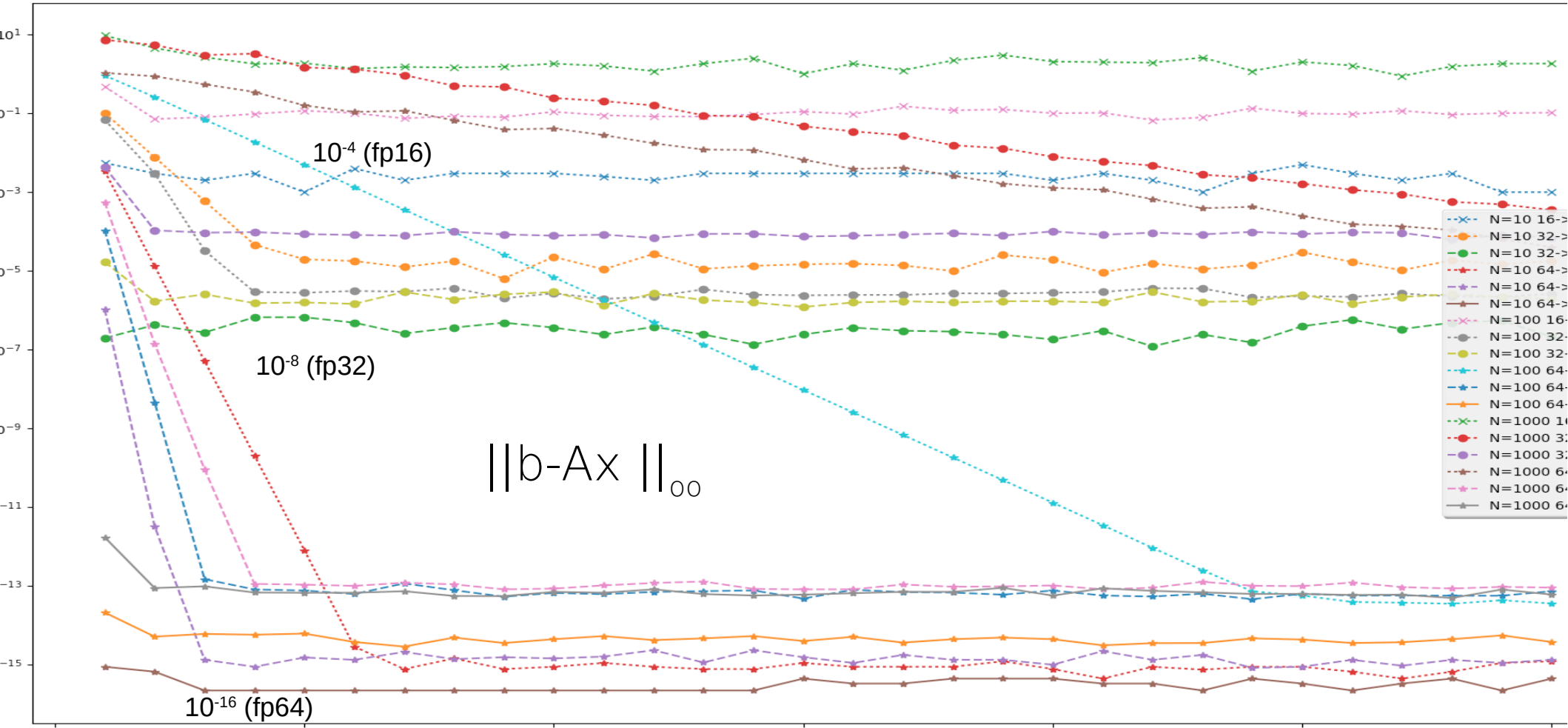
Autotuning with FP16,FP32,FP64 (color)



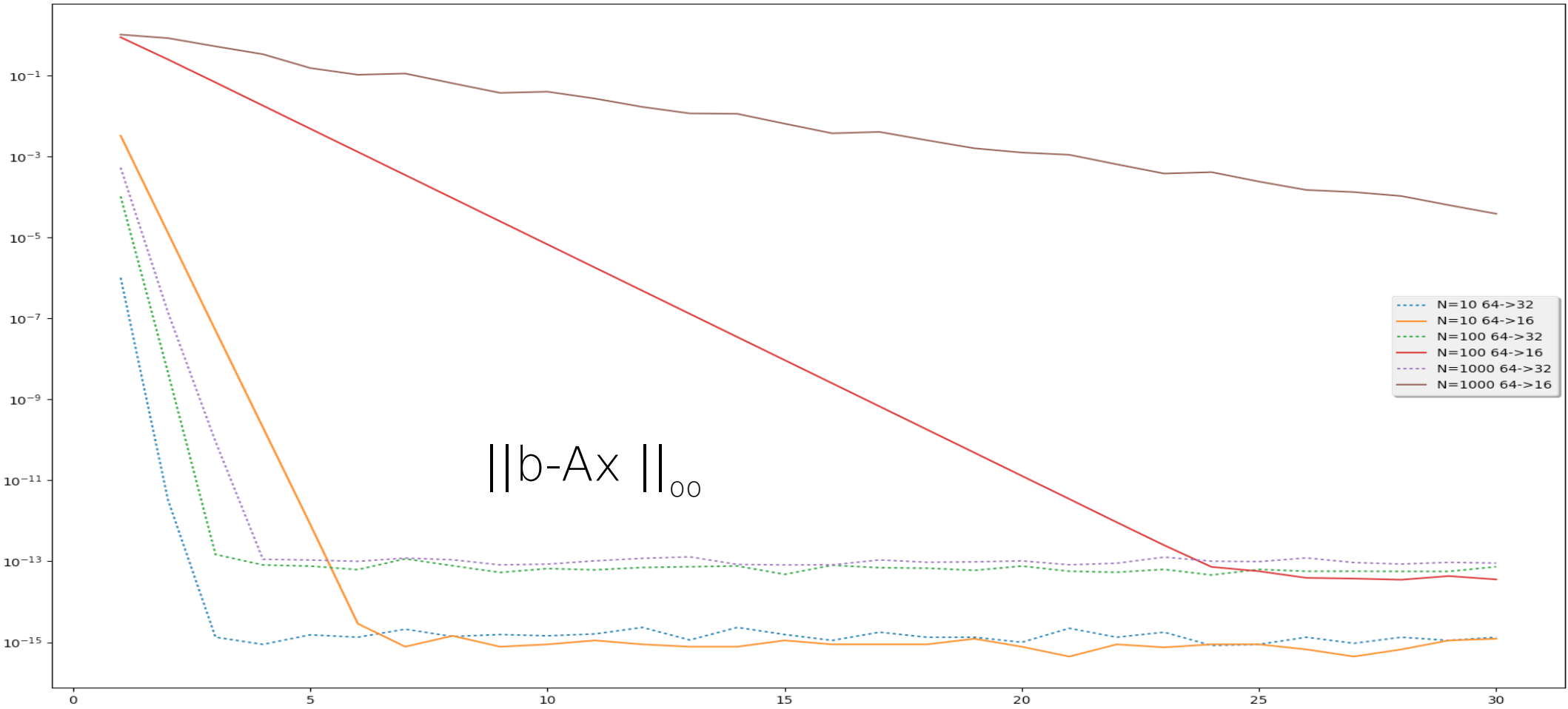
Best Performers for FP16, FP32, FP64



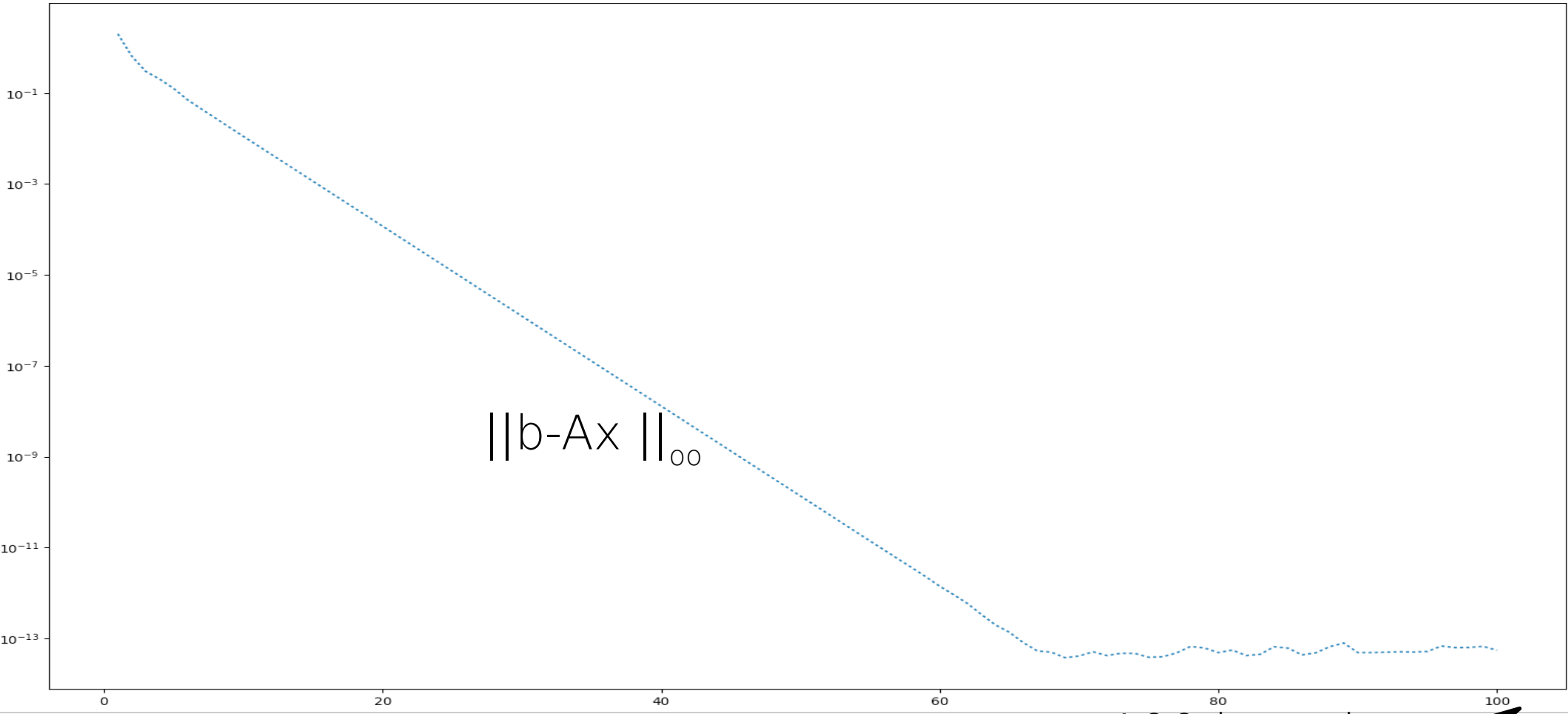
Convergence Results: All Precisions



Example Convergence: FP64 to FP32 / FP16



Example of Slow Convergence: FP64 → FP16



Future Work

- Test on new Hardware
  - IBM/NVIDIA Minsky
  - ARM/Cavium
  - Tegra/Jetson
- New algorithm approaches
  - New iterative schemes
  - New precision tweaks to increase accuracy
- Verification
  - Up-casting
  - Down-casting
  - Convergence
- Performance
  - Improve 16-bit kernels
  - Use 32-bit kernels on non-supporting hardware