The Hazards and Challenges of Low-Precision Computation

Erin Carson

Charles University

SIAM Parallel Processing 2022

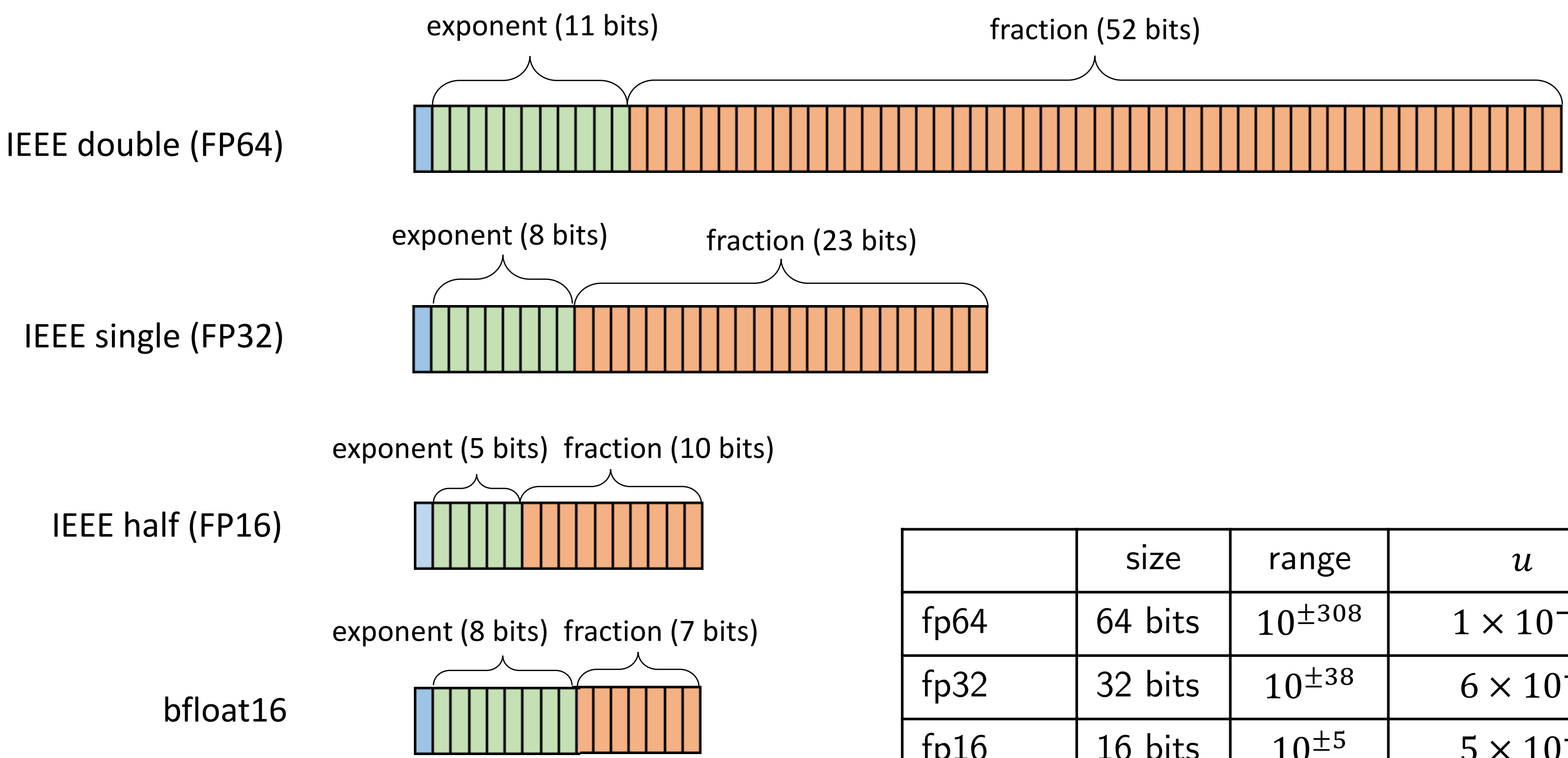
February 24, 2022

FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

We acknowledge funding from Charles Univ. PRIMUS project No. PRIMUS/19/SCI/11, Charles Univ. Research Program No. UNCE/SCI/023, and the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Admin.

Floating Point Formats

$$(-1)^{\mathrm{sign}} \times 2^{(\mathrm{exponent}-\mathrm{offset})} \times 1.\mathrm{fraction}$$



	size	range	u
fp64	64 bits	$10^{\pm 308}$	1×10^{-16}
fp32	32 bits	$10^{\pm 38}$	6×10^{-8}
fp16	16 bits	$10^{\pm 5}$	5×10^{-4}
bfloat16	16 bits	$10^{\pm 38}$	4×10^{-3}

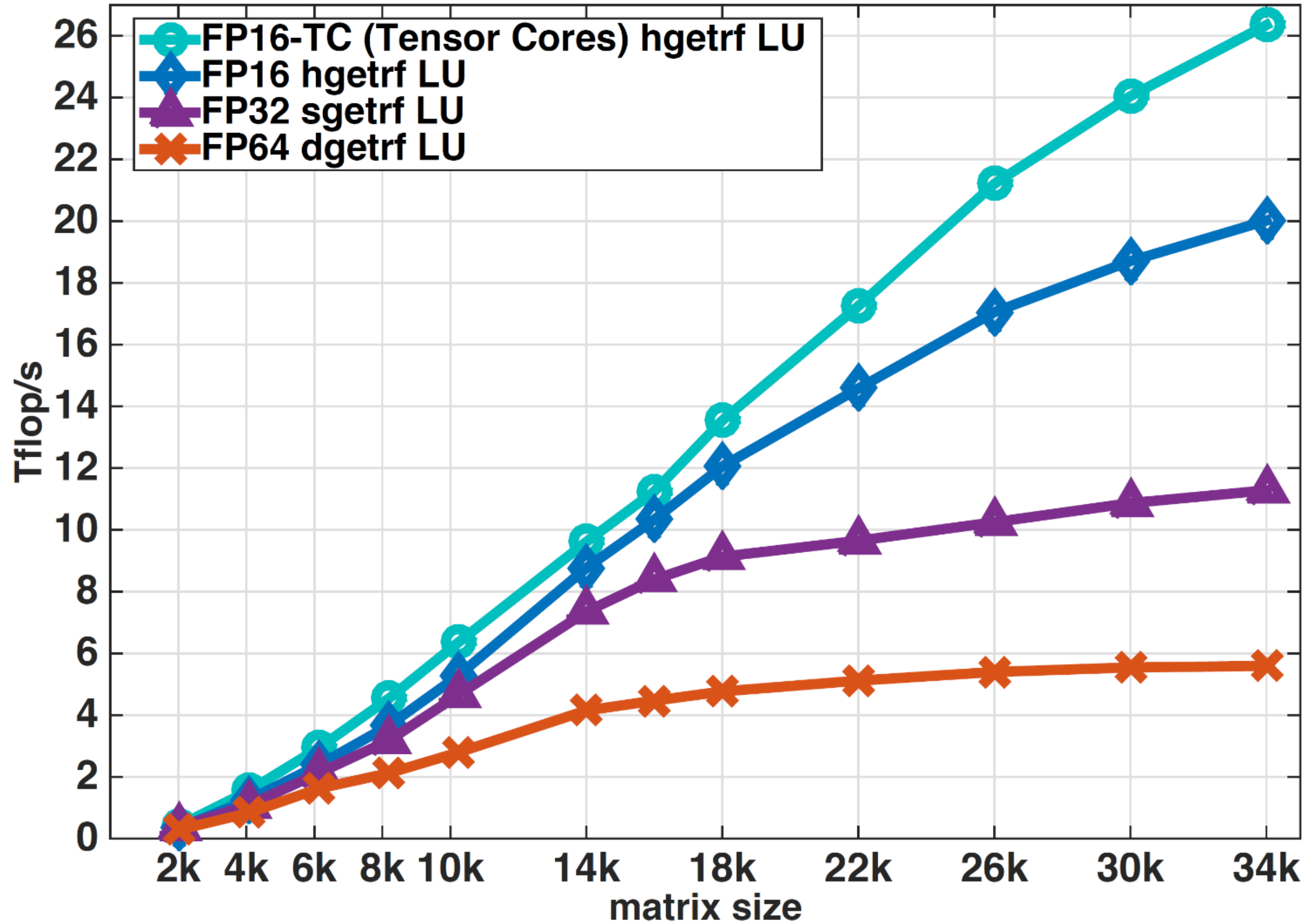
Hardware Support for Multiprecision Computation

Use of low precision in machine learning has driven emergence of low-precision capabilities in hardware:

- AMD Radeon Instinct MI25 GPU, 2017:
  - single: 12.3 TFLOPS, half: 24.6 TFLOPS
- NVIDIA Tesla P100, 2016: native ISA support for 16-bit FP arithmetic
- NVIDIA Tesla V100, 2017: tensor cores for half precision;

  4x4 matrix multiply in one clock cycle
  - double: 7 TFLOPS, half+tensor: 112 TFLOPS (**16x!**)
- NVIDIA A100, 2020: tensor cores with multiple supported precisions: FP16, FP64, Binary, INT4, INT8, bfloat16
- Intel AI processors (Nervana, Xeon)
- Google's Tensor processing unit (TPU): as low as 8-bit arithmetic, bfloat16
- Future exascale supercomputers: (~2021) Expected extensive support for reduced-precision arithmetic (32/16/8-bit)

Performance of LU factorization on an NVIDIA V100 GPU



[Haidar, Tomov, Dongarra, Higham, 2018]

Mixed Precision Capabilities on Supercomputers

From TOP500:

June 2021

	Accelerator/CP Family	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	NVIDIA Volta	97	19.4	626,503,420	1,049,977,600	11,875,056
2	NVIDIA Ampere	26	5.2	351,252,600	505,841,268	3,435,116
3	NVIDIA Pascal	9	1.8	57,876,640	85,807,525	1,141,300

Mixed Precision Capabilities on Supercomputers

From TOP500:

June 2021

	Accelerator/CP Family	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	NVIDIA Volta	97	19.4	626,503,420	1,049,977,600	11,875,056
2	NVIDIA Ampere	26	5.2	351,252,600	505,841,268	3,435,116
3	NVIDIA Pascal	9	1.8	57,876,640	85,807,525	1,141,300

June 2019

	Accelerator/CP Family	Count	System Share (%)	Rmax (GFlops)	Rpeak (GFlops)	Cores
1	NVIDIA Pascal	61	12.2	106,025,166	179,951,012	2,738,356
3	NVIDIA Volta	12	2.4	224,559,400	360,593,742	4,488,720

HPL-AI Benchmark

- Highlights confluence of HPC+AI workloads
  - Like HPL, solves dense Ax=b, results still to double precision accuracy
  - Achieves this via mixed-precision iterative refinement
    - may be implemented in a way that takes advantage of the current and upcoming devices for accelerating AI workloads

HPL-AI Benchmark

- Highlights confluence of HPC+AI workloads
  - Like HPL, solves dense Ax=b, results still to double precision accuracy
  - Achieves this via mixed-precision iterative refinement
    - may be implemented in a way that takes advantage of the current and upcoming devices for accelerating AI workloads

- HPL-AI Results (June 2021):
  1. Fugaku: 2 EXAFLOP/s (vs. 442 PETAFLOP/s on HPL; 4.5x)
  2. Summit: 1.15 EXAFLOP/s (vs. 149 PETAFLOP/s on HPL; 7.7x)

- More information: https://icl.bitbucket.io/hpl-ai/
- Reference implementation: https://bitbucket.org/icl/hpl-ai/src/

Mixed precision in NLA

- BLAS: cuBLAS, MAGMA, [Agullo et al. 2009], [Abdelfattah et al., 2019], [Haidar et al., 2018]
- Iterative refinement:
  - Long history: [Wilkinson, 1963], [Moler, 1967], [Stewart, 1973], ...
  - More recently: [Langou et al., 2006], [C., Higham, 2017], [C., Higham, 2018], [C., Higham, Pranesh, 2020], [Amestoy et al., 2021]
- Matrix factorizations: [Haidar et al., 2017], [Haidar et al., 2018], [Haidar et al., 2020], [Abdelfattah et al., 2020]
- Eigenvalue problems: [Dongarra, 1982], [Dongarra, 1983], [Tisseur, 2001], [Davies et al., 2001], [Petschow et al., 2014], [Alvermann et al., 2019]
- Sparse direct solvers: [Buttari et al., 2008]
- Orthogonalization: [Yamazaki et al., 2015]
- Multigrid: [Tamstorf et al., 2020], [Richter et al., 2014], [Sumiyoshi et al., 2014], [Ljungkvist, Kronbichler, 2017, 2019]
- (Preconditioned) Krylov subspace methods: [Emans, van der Meer, 2012], [Yamagishi, Matsumura, 2016], [C., Gergelits, Yamazaki, 2021], [Clark, 2019], [Anzt et al., 2019], [Clark et al., 2010], [Gratton et al., 2020], [Arioli, Duff, 2009], [Hogg, Scott, 2010]

For survey and references, see [Abdelfattah et al., IJHPC, 2021]

Challenges of low precision

- Do error bounds still apply?
  - Error bound with constant nu provides no information if $nu > 1$
  - One solution: probabilistic approach [Higham, Mary, 2019], [Higham, Mary, 2020]

Challenges of low precision

- Do error bounds still apply?
  - Error bound with constant nu provides no information if $nu > 1$
  - One solution: probabilistic approach [Higham, Mary, 2019], [Higham, Mary, 2020]

- Smaller range of representable numbers
  - Limited range of lower precision might cause overflow when rounding
  - Quantities rounded to lower precision may lose important numerical properties (e.g., positive definiteness)
  - One solution: scaling and shifting approach [Higham, Pranesh, 2019]

Challenges of low precision

- Do error bounds still apply?
  - Error bound with constant nu provides no information if $nu > 1$
  - One solution: probabilistic approach [Higham, Mary, 2019], [Higham, Mary, 2020]

- Smaller range of representable numbers
  - Limited range of lower precision might cause overflow when rounding
  - Quantities rounded to lower precision may lose important numerical properties (e.g., positive definiteness)
  - One solution: scaling and shifting approach [Higham, Pranesh, 2019]

- Larger unit roundoff
  - Lose something small when storing: $fl(x) = x(1+\delta), \ \ |\delta| \leq u$
  - Lose something small when computing: $fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \ \ |\delta| \leq u$

Challenges of low precision

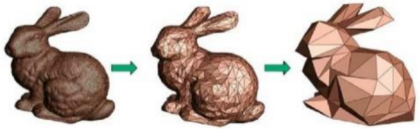
- Do error bounds still apply?
  - Error bound with constant nu provides no information if $nu > 1$
  - One solution: probabilistic approach [Higham, Mary, 2019], [Higham, Mary, 2020]
- Smaller range of representable numbers
  - Limited range of lower precision might cause overflow when rounding
  - Quantities rounded to lower precision may lose important numerical properties (e.g., positive definiteness)
  - One solution: scaling and shifting approach [Higham, Pranesh, 2019]
- Larger unit roundoff
  - Lose something small when storing: $fl(x) = x(1+\delta), \quad |\delta| \leq u$
  - Lose something small when computing: $fl(x \text{ op } y) = (x \text{ op } y)(1+\delta), \quad |\delta| \leq u$

Does it matter?

Inexact computations

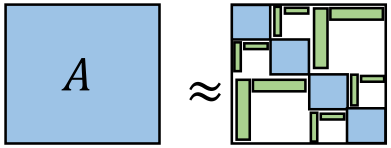
- In real computations we have many sources of inexactness
    - Imperfect data, measurement error
    - Modeling error, discretization error
    - Intentional approximation to improve performance
        - Reduced models, Low-rank representations, sparsification, randomization

Model Reduction

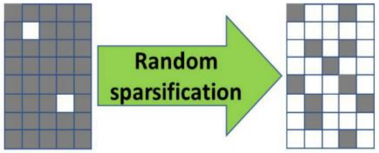


[Schilders, van der Vorst, Rommes, 2008]

Low-rank (hierarchical) approximation



Sparsification, Randomized algorithms

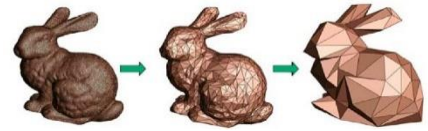


[Sinha, 2018]

Inexact computations

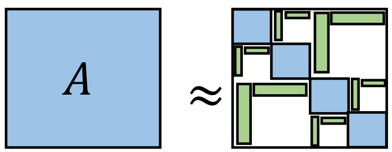
- In real computations we have many sources of inexactness
  - Imperfect data, measurement error
  - Modeling error, discretization error
  - Intentional approximation to improve performance
    - Reduced models, Low-rank representations, sparsification, randomization

Model Reduction



[Schilders, van der Vorst, Rommes, 2008]

Low-rank (hierarchical) approximation



Sparsification, Randomized algorithms

[Sinha, 2018]

- Given that we are already working with so much inexactness, does it matter if we use lower precision?
  - Analysis of accuracy in techniques that use intentional approximation ***almost always*** assume that roundoff error is small enough to be ignored
  - Is this true? Is it true even if we use low precision?

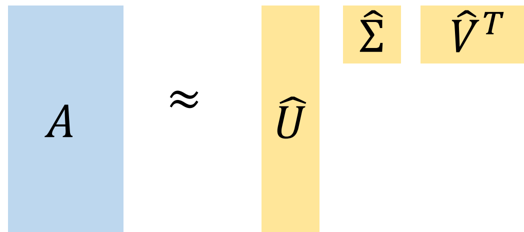
Example: Randomized Algorithms

- Given $m \times n$ A, want truncated SVD with parameter k

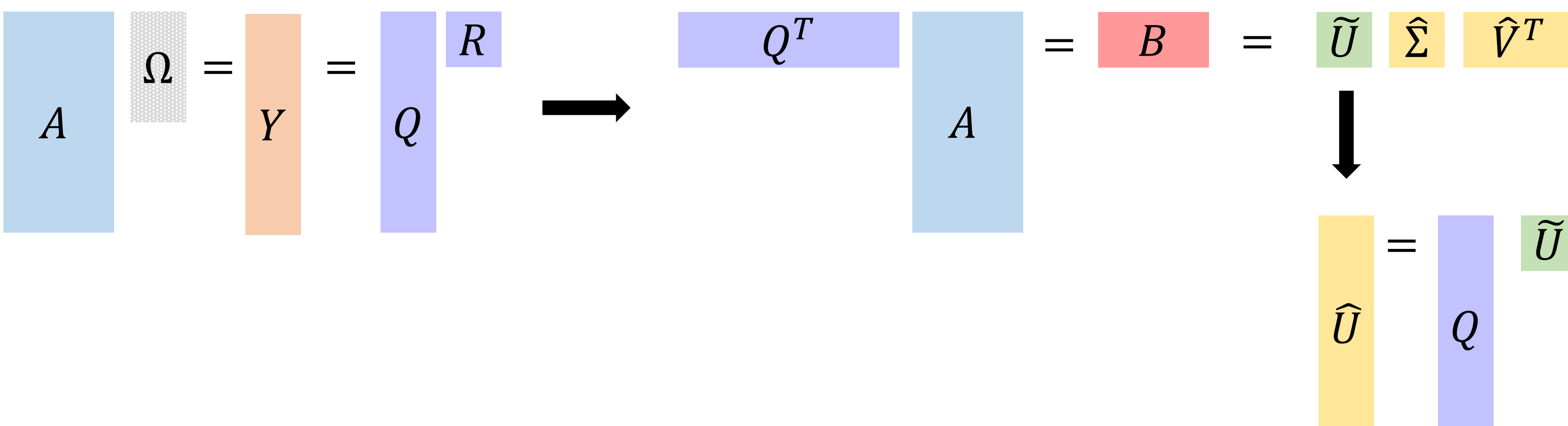
$$A \approx \hat{U} \hat{\Sigma} \hat{V}^T$$

Example: Randomized Algorithms

- Given $m \times n$ A, want truncated SVD with parameter k



- Randomized SVD:



Assuming exact arithmetic:
If Q satisfies $\|A - QQ^T A\| \leq \varepsilon$, then $\left\|A - \hat{U}\hat{\Sigma}\hat{V}^T\right\| \leq \varepsilon$

What happens in finite precision?

Let's try different types of randsvd matrices from the MATLAB gallery:

```
A = gallery('randsvd',[100,40],1e6,mode); k=15;
```

$[U, S, V] \quad = \mathrm{svd}(A)$: non-randomized SVD, exact arithmetic

$[\hat{U}, \hat{S}, \hat{V}] \quad = \mathrm{rsvd}(A)$: randomized SVD, exact arithmetic

$[\hat{U}_d, \hat{S}_d, \hat{V}_d] = \mathrm{rsvd}(A)$: randomized SVD, double precision

$[\hat{U}_h, \hat{S}_h, \hat{V}_h] = \mathrm{rsvd}(A)$: randomized SVD, half precision

What happens in finite precision?

Let's try different types of randsvd matrices from the MATLAB gallery:

```
A = gallery('randsvd',[100,40],1e6,mode); k=15;
```

$[U, S, V] = \text{svd}(A)$: non-randomized SVD, exact arithmetic

$[\hat{U}, \hat{S}, \hat{V}] = \text{rsvd}(A)$: randomized SVD, exact arithmetic

$[\hat{U}_d, \hat{S}_d, \hat{V}_d] = \text{rsvd}(A)$: randomized SVD, double precision

$[\hat{U}_h, \hat{S}_h, \hat{V}_h] = \text{rsvd}(A)$: randomized SVD, half precision

Mode 3: Geometrically distributed singular values

$\|A - USV^T\|_2$ = 4.92e-03

$\|A - \hat{U}\hat{S}\hat{V}^T\|_2$ = 4.92e-03

$\left\|A - \hat{U}_d\hat{S}_d{\hat{V}_d}^T\right\|_2$ = 4.92e-03

$\left\|A - \hat{U}_h\hat{S}_h{\hat{V}_h}^T\right\|_2$ = 4.92e-03

What happens in finite precision?

Let's try different types of randsvd matrices from the MATLAB gallery:

```
A = gallery('randsvd',[100,40],1e6,mode); k=15;
```

$[U, S, V] = \text{svd}(A)$: non-randomized SVD, exact arithmetic

$[\hat{U}, \hat{S}, \hat{V}] = \text{rsvd}(A)$: randomized SVD, exact arithmetic

$[\hat{U}_d, \hat{S}_d, \hat{V}_d] = \text{rsvd}(A)$: randomized SVD, double precision

$[\hat{U}_h, \hat{S}_h, \hat{V}_h] = \text{rsvd}(A)$: randomized SVD, half precision

Mode 3: Geometrically distributed singular values

$$\|A - USV^T\|_2 = 4.92\text{e-}03$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_d\hat{S}_d{\hat{V}_d}^T\right\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_h\hat{S}_h{\hat{V}_h}^T\right\|_2 = 4.92\text{e-}03$$

Mode 1: one large singular value

$$\|A - USV^T\|_2 = 1.00\text{e-}06$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_d\hat{S}_d{\hat{V}_d}^T\right\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_h\hat{S}_h{\hat{V}_h}^T\right\|_2 = 1.11\text{e-}05$$

What happens in finite precision?

Let's try different types of randsvd matrices from the MATLAB gallery:

```
A = gallery('randsvd',[100,40],1e6,mode); k=15;
```

$[U,S,V] = \mathrm{svd}(A)$: non-randomized SVD, exact arithmetic

$[\hat{U},\hat{S},\hat{V}] = \mathrm{rsvd}(A)$: randomized SVD, exact arithmetic

$[\hat{U}_d,\hat{S}_d,\hat{V}_d] = \mathrm{rsvd}(A)$: randomized SVD, double precision

$[\hat{U}_h,\hat{S}_h,\hat{V}_h] = \mathrm{rsvd}(A)$: randomized SVD, half precision

Mode 3: Geometrically distributed singular values

$$\|A - USV^T\|_2 = 4.92\text{e-}03$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_d\hat{S}_d\hat{V}_d^{\,T}\right\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_h\hat{S}_h\hat{V}_h^{\,T}\right\|_2 = 4.92\text{e-}03$$

Mode 1: one large singular value

$$\|A - USV^T\|_2 = 1.00\text{e-}06$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_d\hat{S}_d\hat{V}_d^{\,T}\right\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_h\hat{S}_h\hat{V}_h^{\,T}\right\|_2 = 1.11\text{e-}05$$

Use of low precision leads to an order magnitude loss of accuracy! Roundoff error can't be ignored!

What happens in finite precision?

Let's try different types of randsvd matrices from the MATLAB gallery:

```
A = gallery('randsvd',[100,40],1e6,mode); k=15;
```

$[U,S,V] = \mathrm{svd}(A)$: non-randomized SVD, exact arithmetic

$[\hat{U},\hat{S},\hat{V}] = \mathrm{rsvd}(A)$: randomized SVD, exact arithmetic

$[\hat{U}_d,\hat{S}_d,\hat{V}_d] = \mathrm{rsvd}(A)$: randomized SVD, double precision

$[\hat{U}_h,\hat{S}_h,\hat{V}_h] = \mathrm{rsvd}(A)$: randomized SVD, half precision

Error bound no longer holds!

Mode 3: Geometrically distributed singular values

$$\|A - USV^T\|_2 = 4.92\text{e-}03$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_d\hat{S}_d{\hat{V}_d}^T\right\|_2 = 4.92\text{e-}03$$
$$\left\|A - \hat{U}_h\hat{S}_h{\hat{V}_h}^T\right\|_2 = 4.92\text{e-}03$$

Mode 1: one large singular value

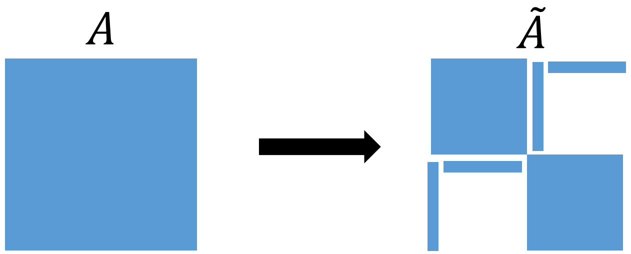
$$\|A - USV^T\|_2 = 1.00\text{e-}06$$
$$\|A - \hat{U}\hat{S}\hat{V}^T\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_d\hat{S}_d{\hat{V}_d}^T\right\|_2 = 1.17\text{e-}06$$
$$\left\|A - \hat{U}_h\hat{S}_h{\hat{V}_h}^T\right\|_2 = 1.11\text{e-}05$$

$$\left\|A - Q_h Q_h^T A\right\|_2 = 3.59\text{e-}06$$

Use of low precision leads to an order magnitude loss of accuracy! Roundoff error can't be ignored!

Example: Low-Rank Approximation

- Block low-rank approximation and hierarchical matrix representations arise in a variety of applications



- Work on mixed and low precision in block low-rank computations

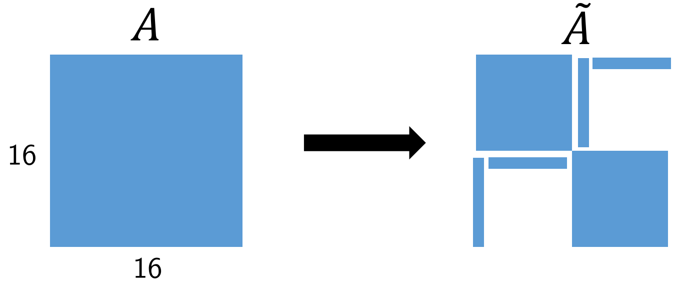
- [Higham, Mary, 2019]: block low-rank LU factorization preconditioner that exploits numerically low-rank structure of the error for LU computed in low precision
- [Higham, Mary, 2019]: Interplay of roundoff error and approximation error in solving block low-rank linear systems using LU
- [Buttari, et al., 2020]: block low-rank single precision coarse grid solves in multigrid
- [Buttari et al., 2021]: Mixed precision low rank approximation and application to block low-rank LU factorization

Example: Low-Rank Approximation

Inverse multiquadratic kernel:

$$A(i,j) = \frac{1}{\sqrt{1 + 0.1\|x - y\|^2}}, \qquad x, y \in \mathbb{R}^2$$

A is SPD. Low-rank approximation of A should also be SPD!

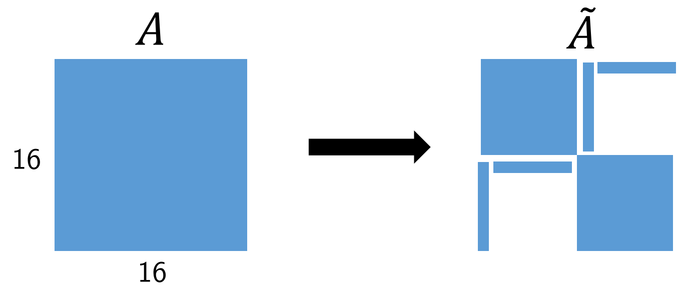


Example: Low-Rank Approximation

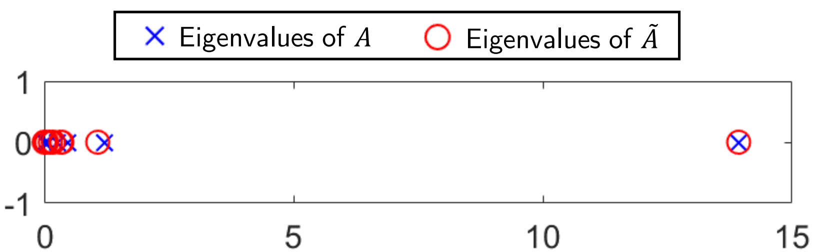
Inverse multiquadratic kernel:

$$A(i,j) = \frac{1}{\sqrt{1 + 0.1\|x-y\|^2}}, \qquad x, y \in \mathbb{R}^2$$

A is SPD. Low-rank approximation of A should also be SPD!



Exact arithmetic SVD:

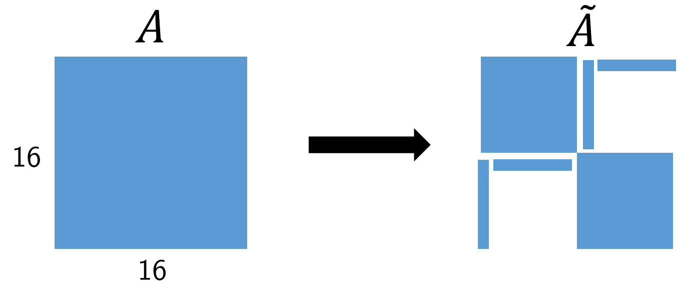


Example: Low-Rank Approximation

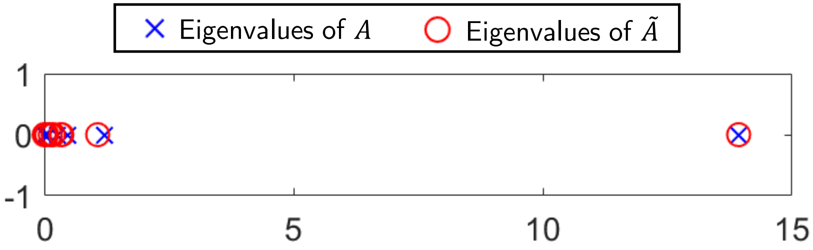
Inverse multiquadratic kernel:

$$A(i,j) = \frac{1}{\sqrt{1 + 0.1\|x - y\|^2}}, \qquad x, y \in \mathbb{R}^2$$

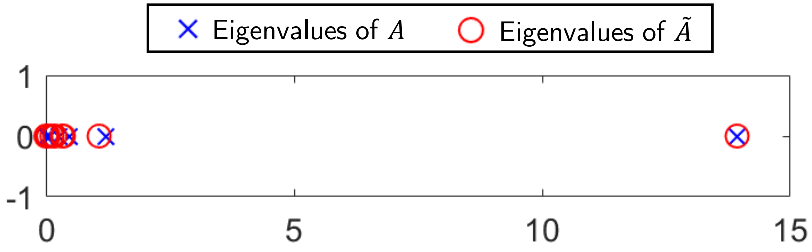
A is SPD. Low-rank approximation of A should also be SPD!



Exact arithmetic SVD:



Half precision SVD:

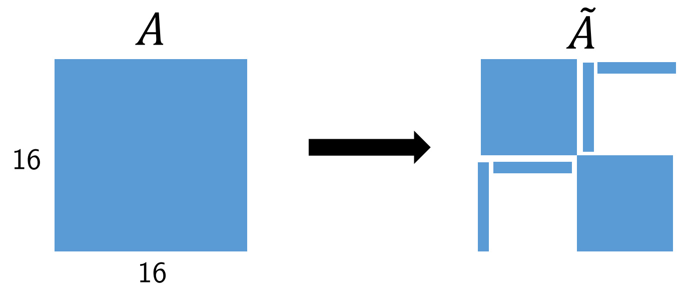


Example: Low-Rank Approximation

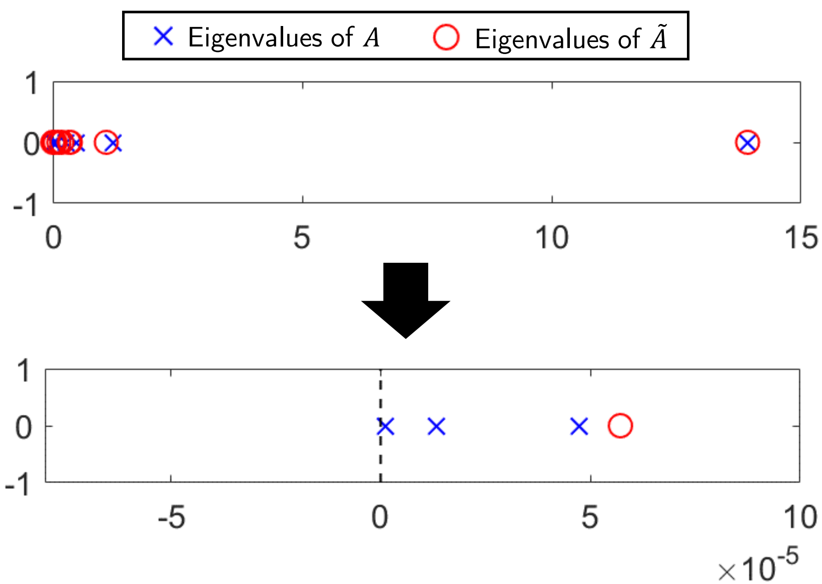
Inverse multiquadratic kernel:

$$A(i,j) = \frac{1}{\sqrt{1 + 0.1\|x - y\|^2}}, \qquad x, y \in \mathbb{R}^2$$

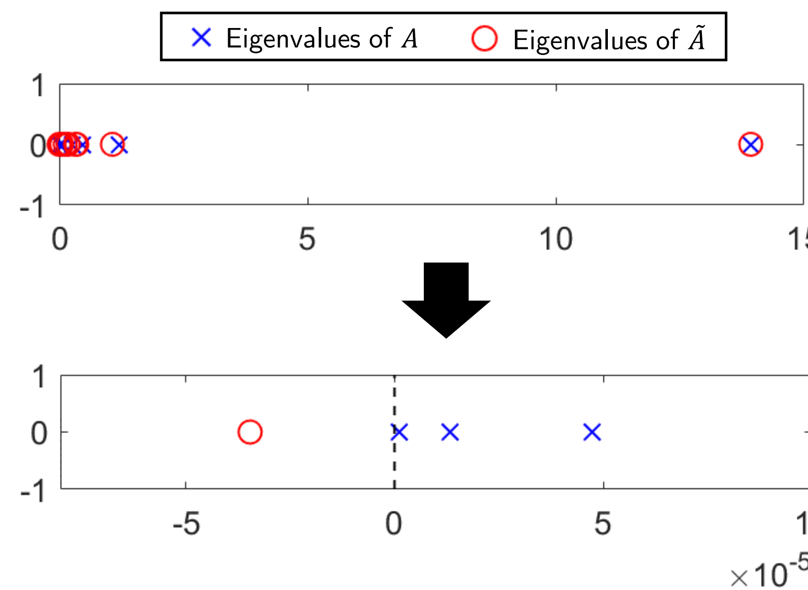
A is SPD. Low-rank approximation of A should also be SPD!



Exact arithmetic SVD:



Half precision SVD:

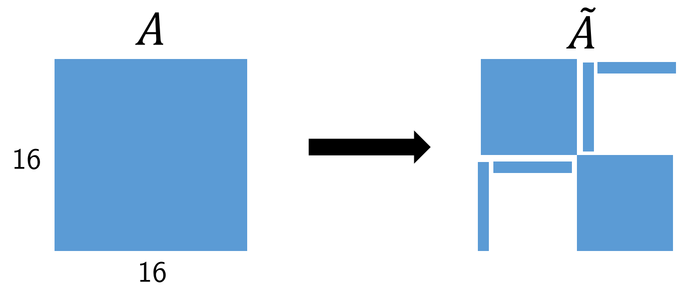


Example: Low-Rank Approximation

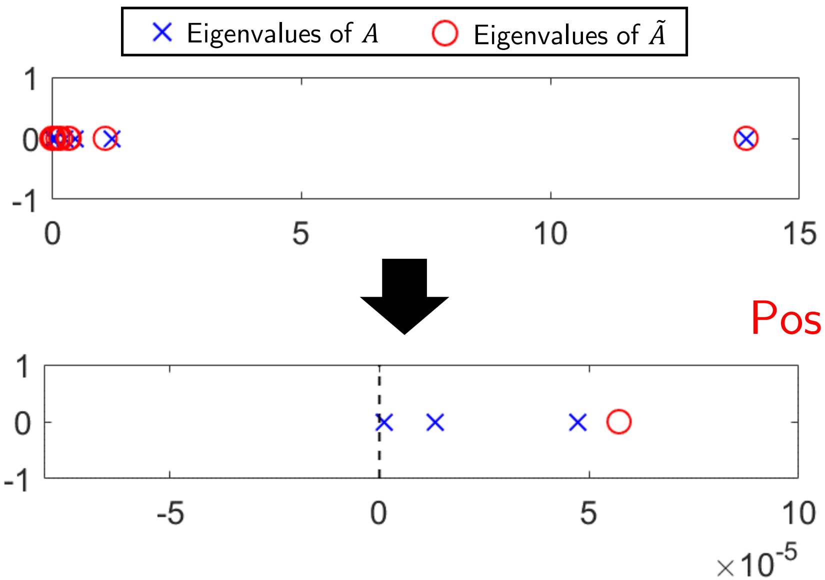
Inverse multiquadratic kernel:

$$A(i,j) = \frac{1}{\sqrt{1 + 0.1\|x - y\|^2}}, \qquad x, y \in \mathbb{R}^2$$

A is SPD. Low-rank approximation of A should also be SPD!

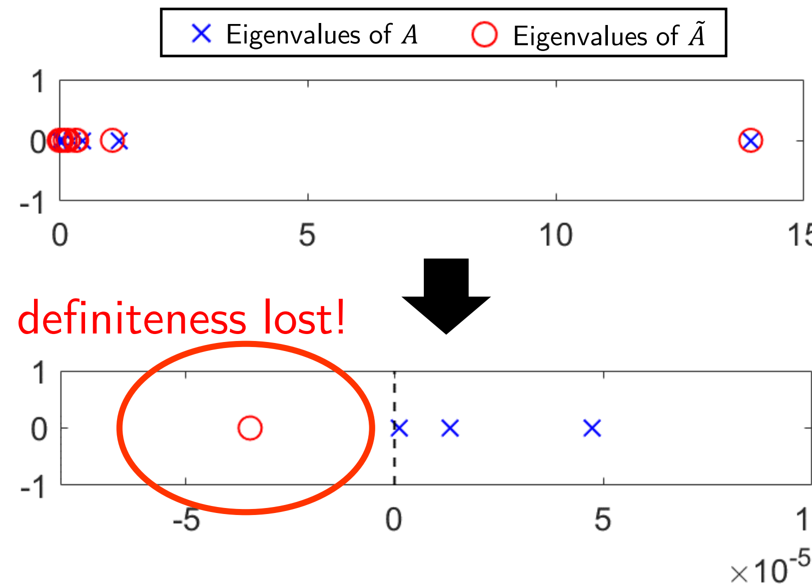


Exact arithmetic SVD:



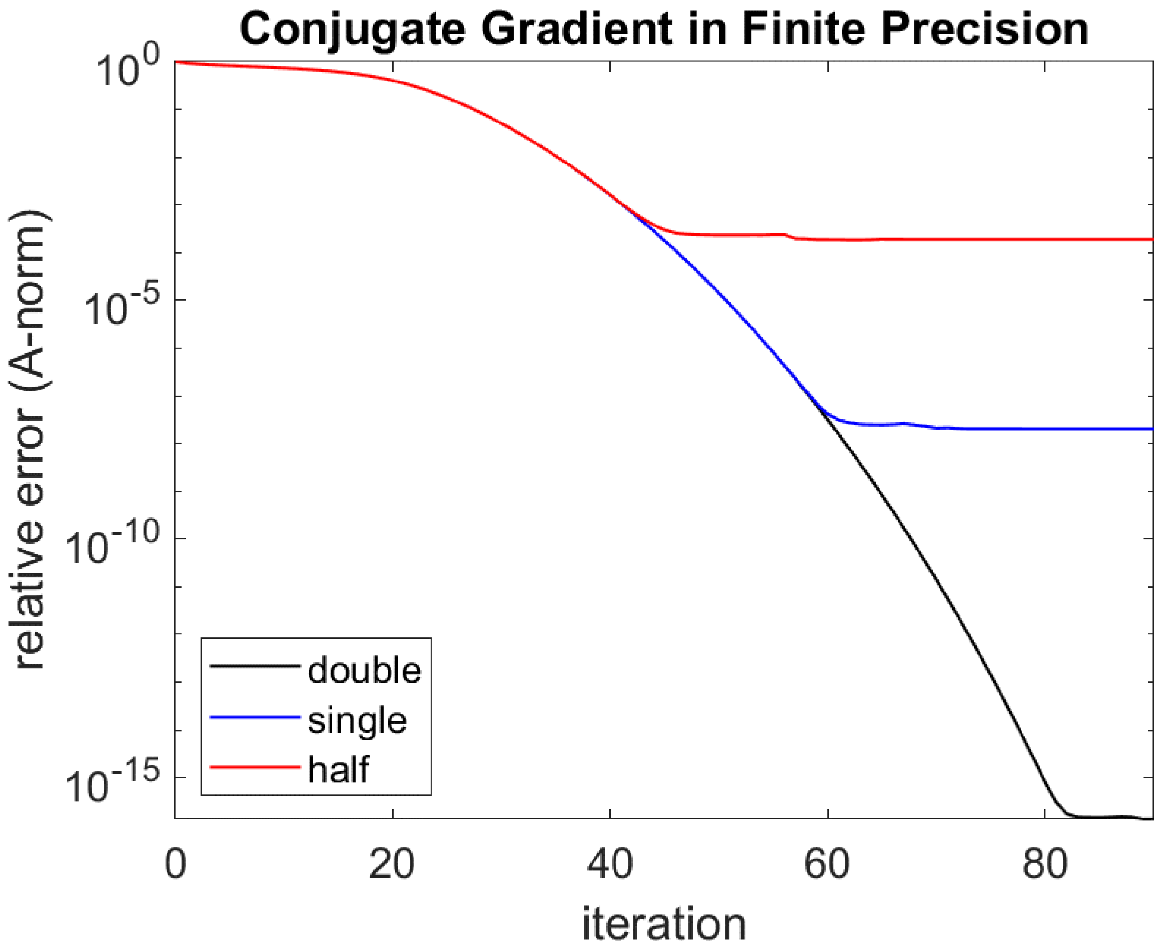
Half precision SVD:

Positive definiteness lost!



Example: Iterative Methods

```
A = diag(linspace(.001,1,100));
[V,~] = eig(A);
b = V'*ones(n,1);
```

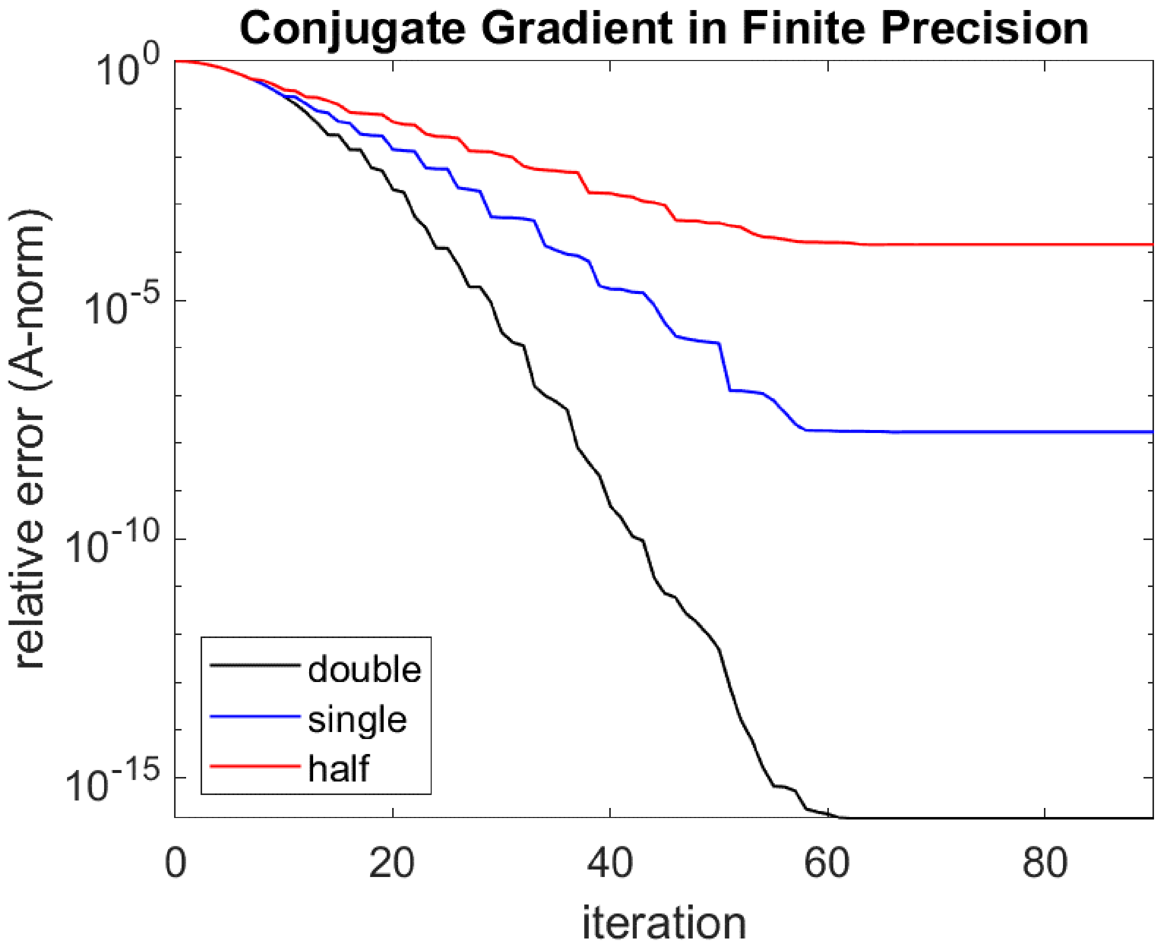


Example: Iterative Methods

$n = 100, \lambda_1 = 10^{-3}, \lambda_n = 1$

$\lambda_i = \lambda_1 + \left(\frac{i-1}{n-1}\right)(\lambda_n - \lambda_1)(0.65)^{n-i}, \quad i = 2, \dots, n-1$

```
[V,~] = eig(A);
b = V'*ones(n,1);
```



Takeaway

- Low precision can have massive performance benefits but must be used with caution!

- Many opportunities for using mixed and low precision computation in scientific applications

- Need to develop a theoretical understanding of how mixed precision algorithms behave; need to revisit analyses of algorithms and techniques that ignore finite precision

Thank you!

carson@karlin.mff.cuni.cz

www.karlin.mff.cuni.cz/~carson/