Scalable Data Generation for Evaluating Mixed-Precision Solvers

Piotr Luszczek
University of Tennessee, USA

Yaohung (Mike) Tsai
University of Tennessee, USA

Neil Lindquist
University of Tennessee, USA

Hartwig Anzt
Karlsruhe Institute of Technology, Germany
University of Tennessee, USA

Jack Dongarra
University of Tennessee, USA
Oak Ridge National Laboratory, USA
University of Manchester, UK

***Abstract*—We present techniques of generating data for mixed precision solvers that allows to test those solvers in a scalable manner. Our techniques focus on mixed precision hardware and software where both the solver and the hardware can take advantage of mixing multiple floating precision formats. This allows taking advantage of recently released generation of hardware platforms that focus on ML and DNN workloads but can also be utilized for HPC applications if a new breed of algorithms is combined with the custom floating-point formats to deliver performance levels beyond the standard IEEE data types while delivering a comparable accuracy of the results.**

I. Introduction

Modern accelerator hardware includes more floating-point formats that mandated by IEEE 754 standard. We denote half-precision 16-bit floating-point arithmetic storage as FP16 or $\mathbb{F}_{5,10}$ (5 bits for exponent and 10 bits for mantissa, FP32 in that notation is $\mathbb{F}_{7,24}$). FP16 is already showing up in enterprise and commodity hardware (e.g., NVIDIA's Tensor Core low-precision functional unit). A multitude of artificial neural network accelerators have been explored, including field-programmable gate arrays (FPGA) [1], [2], [3] and application-specific integrated circuits (ASIC) [4], [5], [6]. These devices achieve thousands of operations (floating point or fixed point) at very high power efficiencies. This more efficient hardware prompted research in code generators that take a neural network model design and produce a hardware description for seamless synthesis [7]. By comparison, ASIC-based accelerators enjoy multiple benefits over FPGAs, including higher clock speeds and lower power consumption. But the additional functionality of being reconfigurable is a unique aspect of FPGAs that make them a better fit for many applications. On-the-fly reconfiguration enables the user to: rapidly exchange and deploy improved neural model designs; implement fixes for correctness and performance issues as they are discovered; and deploy algorithmic changes that benefit rapid development of new neural learning concepts. Additionally, on-demand hardware synthesis permits reuse of the same hardware board across different neural networks for a staged provisioning in production. This is exemplified by Microsoft Brainwave [1], [2], NeuFlow [8], FPGAconvNet [3], and others [9], [10].

This research was supported by 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 Administration.

II. Related Work

Generalized Minimum Residual (GMRES) could be considered the most common method for mixed-precision solvers due to its numerical properties. Maintaining the stability and orthogonality of GMRES requires both algorithmic considerations as well as the properties of the system matrix [11], [12], [13], [14], [15], [16], [17].

HPL [18], HPCC [19], [20], HPCG [21], [22], [23], [24], [25] are benchmarks that are widely used across industrial, research, and academic High Performance Computing (HPC) installations for measuring dense operations, memory access patterns, and sparse iterative solvers, respectively. They all face the initialization problem of the system matrix A that is subsequently used to measure the solver performance. In order to achieve a scalable data generation process, a distributed memory random number generator is employed in these benchmarks which produces uniformly distributed elements between $-1/2$ and $1/2$: $a_{i,j} \sim \mathcal{U}(-1/2, 1/2)$. In case of HPCG, a standard 3D discretization is constructed via 27-point finite difference stencil thusly also constituting a scalable data generation mechanism.

III. HPL-AI Benchmark Definition

In the core of performance benchmarking portion of the HPL-AI benchmark is a solver for the following linear system:

$$Ax = b, \quad A \in \mathbb{F}_{64_b}^{n \times n},\ x, b \in \mathbb{F}_{64_b}^{n}$$

All input matrices and objects are in 64-bit floating-point format and the answer is expected to be accurate to the same precision regardless of the underlying method used in the solver. The specification prescribes a solver based on the Krylov subspace iteration with GMRES as the main method to accommodate the non-symmetric system matrix A:

$$x \leftarrow \mathrm{GMRES}(A, b, M)$$

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

where M represents the right-preconditiner matrix that is to be based on LU factorization of A in lower precision:

$$P \times A \rightarrow L \times U \qquad L, U \in \mathbb{F}_{32_b}, \; P \in \{0,1\}^{n \times n}$$

The computed and storage precision of the L and U factors may be lower than 32-bits and is hardware-specific in practice to showcase the underlying performance capacity. Along the same lines, the HPL-AI rules relax the pivoting requirements represented by matrix P which could be an identity $\mathbb{I}_n$ indicating the lack of pivoting. The main goal of the scalable matrix generator is to make non-pivoting LU factorization possible under suitable circumstances for numerical stability.

Some of the guiding principles for the design of the eventual generator of matrix A include:

- Ascertain that the un-preconditioned GMRES convergence is slow and possibly even stagnates without producing satisfactory solution.
- Limit numerical issues in non-pivoting LU factorization so that some implementation may forgo the requirement of pivoting in LU preconditioner without excessive pivot growth.
- The system matrix A must be constructable in $O(n^2)$ steps with a small constant multiplier to minimize time for constructing n by n matrix that fits the tested system and scale across the computational units.
- For any given matrix size n, there should be hundreds of matrices or even matrix classes to choose from so as to give the implementation harness a potential to evade predictability and delay detailed generation decision to late in runtime.

IV. Random and Named Matrices

Special matrices such as Cauchy: $[a_{i,j}] \equiv [1/(i-j)]$ and Hilbert $[a_{i,j}] \equiv [1/(i+j-1)]$ matrices may be cheaply constructed and they admit useful numerical properties such as QR factorization $\mathcal{O}(n^2)$ complexity [26] or an explicit inverse that is known for Hilbert matrices even though they are provably badly conditioned.

HPL-AI benchmark [27], [28] aims to use the GMRES-IR method [29] to solve a large linear system $Ax = b$ on an platforms with hardware accelerators that feature limited precision arithmetic at very high performance rate. This mimics how this type of machines are utilized by deep learning and mixed-precision solvers. Consequently, the HPL-AI rules explicitly permit non-pivoted LU factorization and thus the matrix generator needs to produce matrices that are numerically stable for such an approach to work successfully.

V. Proposed Scalable Matrix Generator

A natural starting point for creating a well-conditioned matrix is to use the strict diagonal dominance property. Row-wise, it is defined as: $|a_{ii}| > \sum_{j \neq i} |a_{ij}|$ for each $i = 1, 2, \ldots, n$ and the property may also be formulated column-wise by transposing the indices of the formula. The property of *strong diagonal dominance* imbues the matrix with two relevant guarantees,

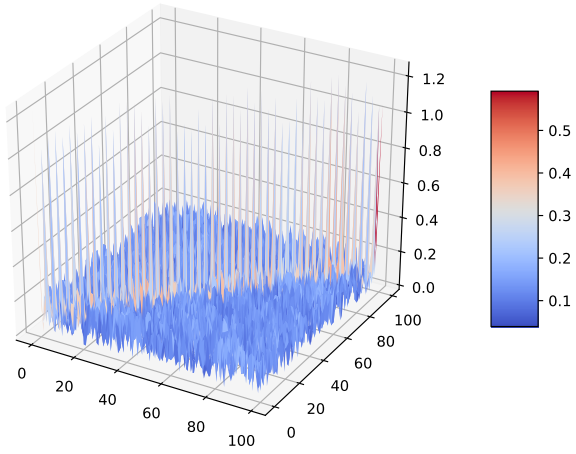

Fig. 1. Sample matrix with diagonal dominance departure factor $f_{d3} = 25\%$ with $a_{i,j} \sim \mathcal{U}(0,1)$.

namely well-conditioning and limited pivot growth even if no pivoting is used during the Gaussian elimination. This may be observed in the Schur complement which is the elimination's main computational pattern that performs the update of the remaining right portion of the matrix based on the current column k:

$$a_{i,j}^{(k)} \leftarrow \sum_{i,j>k} a_{i,k} a_{k,k}^{-1} a_{k,j} \tag{1}$$

for elements $a_{i,j}^{(k)}$ to the right of column k: $i, j > k$. Note how the independence of the updates in Eq. (1) are the main source of parallelism of the elimination that leads to scalable implementations across CPUs, GPUs, and distributed memory nodes.

TABLE I
The number of iterations required for convergence of GMRES with an identity preconditioner $M \equiv \mathbb{I}_n$ and with $M \equiv (L_{FP32} \times U_{FP32})^{-1}$ based on lower-precision L and U factors.

n	$M \equiv \mathbb{I}_n$	$M \equiv (L_{FP32} \times U_{FP32})^{-1}$
500	8	2
1000 ... 3000	7	2
3500 ... 10000	6	2

A straightforward way of constructing a diagonally dominant matrix A_{dd} is to additively increase the diagonal:

$$A_{dd} \equiv n \times \mathbb{I}_n + \mathcal{U}_n(0,1) = n \times \left(\overbrace{\mathbb{I}_n + \mathcal{U}_n(0, 1/n))}^{a_{i,j} \in [0,1]} \right) \tag{2}$$

which translates into a trivial Julia/MATLAB/Octave code: `n*eye(n)+rand(n)`. The issue with this matrix for the HPL-AI benchmark is it does not require a good preconditioner to guarantee a quick convergence with a standard GMRES implementation. This is summarized in Table I for matrices of sizes between 500 and 10000. We see that without

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

preconditioner, the iteration count is only marginally larger than when using high quality preconditioner that accurately approximates A^{-1} in FP32. Our goal is to differentiate between the two scenarios by adjusting the numerical properties of the system matrix A while still maintaining stability for non-pivoting factorizations.

We proceed by introducing *departure from diagonal dominance* factor that we note as f_{d3} throughout this writing. It is the fraction of diagonal dominance in Eq. (2) applied to the random matrix.

Definition V.1. *Departure from diagonal dominance factor* $f_{d3} \in (0, 1)$ *is the scaling applied to diagonal entries:*

$$A(f_{d3}) \equiv n \times (\mathbb{I}_n + f_{d3} \times \mathcal{U}_n(0, 1)) . \tag{3}$$

If f_{d3} *is constraint to* n *levels we denote it* $k_{d3} \in (1, n)$*:*

$$A(k) \equiv n \times \left(\mathbb{I}_n + \frac{k}{n} \times \mathcal{U}_n(0, 1) \right) . \tag{4}$$

In order to put f_{d3} factor in perspective, Fig. 1 visualizes sample matrix entries for diagonal dominance departure factor $f_{d3} = 25\%$ with $a_{i,j} \sim \mathcal{U}(0, 1)$.

Note that it is possible to use a different random distribution for the matrix entries. For example, in the HPL benchmark $a_{i,j} \sim \mathcal{U}(-1/2, 1/2)$. In our experiments, normal and uniform distribution had a similar effect of GMRES convergence and pivoting requirements. See Fig. 3 in §IX-A for more details.

VI. Eigenvalue Spectrum Analysis

Spectral properties of the matrix strongly influence the convergence behavior of GMRES algorithm. To test the influence of f_{d3} of the eigenvalue spectrum, in Figure 2 we plotted the eigenvalues on the complex plane for various matrix sizes and f_{d3} values. There are two main patterns emerging from these figures. The dominant eigenvalue is real, positive and its value depends on the size of the matrix in line with the analysis of random matrices [30], [31], [32], [33], [34]. The remaining eigenvalues are for the most part contained inside a unit circle around the origin. The large separation of the largest (in magnitude) eigenvalue from the others explains a fast convergence of the Krylov subspace solver. Introducing the f_{d3} factor diminishes the length of that separation and thus reduces the effectiveness of the unpreconditioned GMRES.

VII. Representation in Limited Range Precision

With strict diagonal dominance, the required representation range is linearly proportionate to the matrix dimension n. This is acceptable for floating-point formats equivalent to FP32's representation or better that admit suitably large exponents and can reproduce matrix element ranges of practical importance. More precisely, FP32 $\mathbb{F}_{7,24}$ reserves 7 bits out of the 32-bit storage for the exponent which is always signed[1]. Also, included in this category of sufficient range representations is BF16 or $\mathbb{F}_{7,8}$ that uses 7-bit exponent just like IEEE's FP32.

[1]Some bit combinations are prohibited in order to represent infinity and NaNs, both signaling and non-signaling.

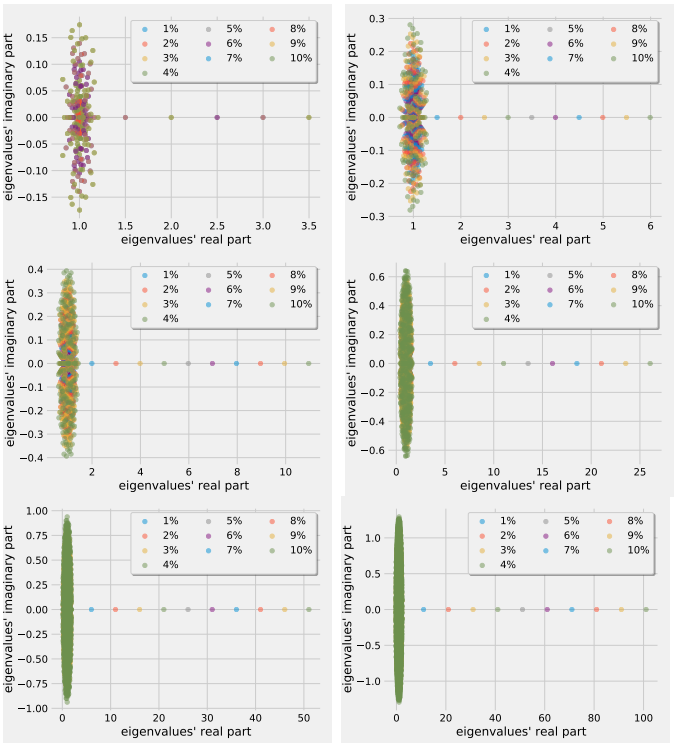

Fig. 2. Distribution of eigenvalues of random matrices of size 50, 100 (top row), 200, 500 (middle row), 1000, and 2000 (bottom row) and different f_{d3} factors from 1% to 10%.

However, the IEEE 754 standard also includes FP16 (for both storage and computation as of the 2019 update) has an issue because $\mathbb{F}_{5,10}$ overflows beyond ± 65504. To circumvent that, a combination of scaling techniques may be applied [27], [28]. One such method is shown in Eq. (2) on the right where we scale the matrix elements to the $(0, 1)$ and factor out the matrix dimension n that may be stored externally in higher precision.

VIII. Performance Model: Floating-Point Execution Rate, Bandwidth, and Latency

TABLE II
CPU performance and memory bandwidth for typical server chips and a single channel MMU.

DDR3 GB/s	DDR4 GB/s	FP64 Gflop/s	FP32 Gflop/s	FP16 Gflop/s
12.8	21.3	1000	2000	4000

TABLE III
GPU performance and memory bandwidth across three generations of server accelerators.

Pascal		Volta		Ampere	
HBM GB/s	FP16 Tflop/s	HBM GB/s	FP16 Tflop/s	HBM GB/s	FP16 Tflop/s
720	8	900	125	1555	312

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

TABLE IV
BANDWIDTH AND LATENCY VALUES FOR INFINIBAND NETWORK.

Generation	FDR	EDR	HDR
Single link bandwidth (Gbs)	14	25	50
Latency (ns)	700	500	300

Our performance model of FGMRES has three components related to computing LU preconditioner, matrix-vector multiplication, and dot-product:

$$t = t_{LU} + t_{MV} + t_{DOT} \tag{5}$$

The floating-point precision performance is the limiting factor for computing the preconditioner based on LU factorization:

$$t_{LU} = \frac{2}{3} \times \frac{n^3}{p_{FP}} \tag{6}$$

The typical values for the performance for CPUs and GPUs are given in Tables. II and III, respectively.

The matrix-vector multiplication is a bandwidth-bound operation and the number of GMRES iterations i_c determines the number of multiplications required:

$$t_{MV} = i_c \times \frac{n^2}{\min(b_{mem}, b_{net})} \tag{7}$$

The bandwidth depends on both the type of data transfers needed for the matrix-vector multiplication. The main memory bandwidth b_{mem} dominates in a single-node scenarios while the distributed memory setting needs to be modeled with consideration of the network bandwidth b_{net}. A set of sample values for these compute, bandwidth, and latency parameters are given it Tables II, III, and IV.

Finally, the dot-product is a reduction operation that exposes the algorithm to the network latency ℓ_{net} (memory latency on a node may be ignored):

$$t_{DOT} = \ell_{net} \times i_c \times \log_2 P \tag{8}$$

Typical network latency values for Infiniband are shown in Table IV. Note that these values are based on low-level hardware signaling overheads and software overheads may increase the latency substantially. Also, latency values is often a random variable rather than a single number and depends on the network traffic and the switches' ability to handle congestion. In practical situation, latency admits multi-modal distribution, especially in dynamically routed networks.

IX. PERFORMANCE RESULTS

A. Properties of Matrices with Diagonally Dominant Departure

Figure 3 shows the results with pivoting for matrices of sizes between 100 and 10000. For each matrix size, the figure shows the largest f_{d3} value (as a percentage of n) for which the resulting matrix does not require pivoting. Three types of random distributions were tested: uniform between 0 and 1, uniform around 0, and normal distribution. The uniform distribution asymptotically approaches around 4% value while

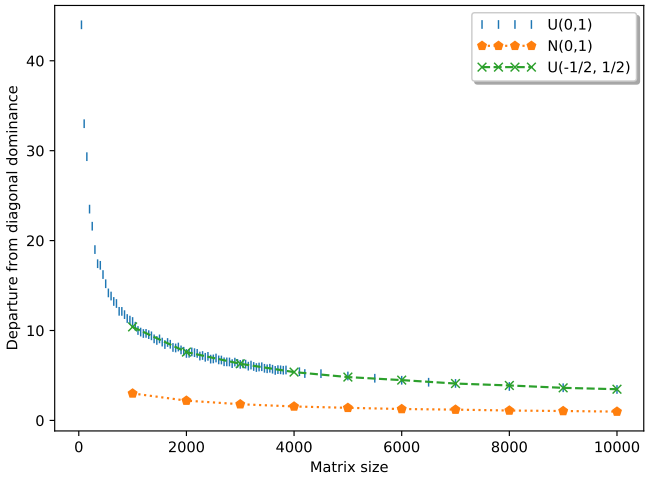

Fig. 3. Largest f_{d3} non-pivoted.

the normal distribution approaches approximately 1% mark. This is a positive result indicating that there is a non-empty range from which to draw the f_{d3} factor. The larger the factor, the more iterations of GMRES are required, on the order of hundreds for large matrix sizes as indicated in Figure 4.

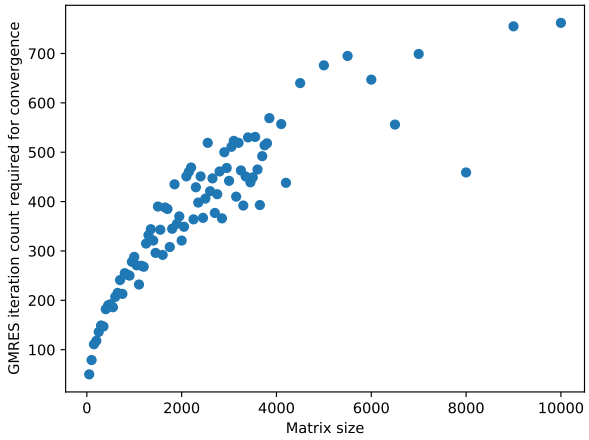

Fig. 4. The number of GMRES iterations required for convergence for the largest f_{d3} factor that allows non-pivoting factorization.

Figure 4 shows the distribution of GMRES iteration counts across matrix sizes for the largest f_{d3} factor that permits non-pivoting LU factorization. The saturation trend observed in Figure 3 also shows up here. At 700 iterations, GMRES is much less effective without a preconditioner, especially considering the performance model from Section VIII. That model as well as other practical considerations limit the iteration count for the HPL-AI benchmark to 50 and therefore using the proposed f_{d3} factor values meets this requirement.

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

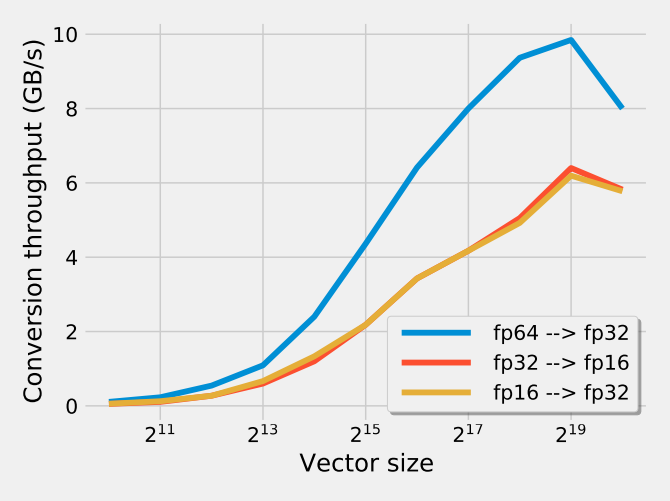

Fig. 5. Throughput of conversion between various floating-point formats on NVIDIA Volta TitanV GPU accelerator.

B. Conversion Throughput

The speed of conversion between floating-point formats is limited by the data access throughput and also relies on the abundant presence of Special Function Units (SFUs) in every streaming processor (SM) of the GPU accelerator. Depending on the platform, the ratio of SFUs to Floating Point Units (FPUs) is a small fraction below 1 and saturating the capacity of High Bandwidth Memory (HBM) of the GPU. Figure 5 establishes these facts experimentally and shows the conversion throughput on the Titan V GPU accelerator from NVIDIA. The conversion code was compiled with CUDA Toolkit version 10.1.243. We see that the achieved bandwidth for conversion is below maximum memory bandwidth of typical HBM version 2. This shows the importance of low complexity of per-element overhead for scalable data generation.

X. Conclusions and Future Work

We presented scalable matrix data generator for the HPL-AI benchmark that targets the specifics of the benchmark's algorithmic variants of GMRES and LU factorization and the floating-point formats of modern hardware accelerator hardware.

Our future work will focus on testing the benchmark and the proposed generator on the newly released NVIDIA Ampere GPU. The new accelerator introduced new tensor units and new floating-point format that lies somewhere between 16 and 32 with approximately 20 bits devoted to the mantissa. The details have not been fully disclosed and only observational evidence may be used to infer the properties of the new format and its computational behavior.

Another research direction is to consider a variety of preconditioning levels from diagonal to full dense factorization in the increasingly higher floating-point precisions.

References

[1] E. Chung, J. Fowers, K. Ovtcharov, M. Papamichael, A. Caulfield, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, and et al., "Serving DNNs in real time at datacenter scale with project brainwave," 2018, iEEE Micro, 38(2):8-20.

[2] J. Fowers, K. Ovtcharov, M. Papamichael, T. Massengill, M. Liu, D. Lo, S. Alkalay, M. Haselman, L. Adams, and e. a. Mahdi Ghandi, "A configurable cloud-scale DNN processor for real-time AI," 2018, in Proceedings of the 45th Annual International Symposium on Computer Architecture, pages 1-14. IEEE Press.

[3] S. I. Venieris and C.-S. Bouganis, "FPGAconvNet: a framework for mapping convolutional neural networks on FPGAs," 2016, in Field-Programmable Custom Computing Machines (FCCM), 2016 IEEE 24th Annual International Symposium on, pages 40-47. IEEE.

[4] N. P. Jouppi, C. Young, N. Patil, and D. Patterson, "A domain-specific architecture for deep neural networks," 2018, commun. ACM, 61(9):50-59, August.

[5] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," 2016, iEEE Journal of Solid-State Circuits, 52(1):127-138.

[6] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, and W. J. Dally, "EIE: efficient inference engine on compressed deep neural network," 2016, in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), pages 243-254. IEEE.

[7] K. Guo, L. Sui, J. Qiu, S. Yao, S. Han, Y. Wang, and H. Yang, "From model to FPGA: Software-hardware co-design for efficient neural network acceleration," 2016, in 2016 IEEE Hot Chips 28 Symposium (HCS), pages 1-27. IEEE.

[8] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: A runtime reconfigurable dataflow processor for vision," 2011, in CVPR Workshops, pages 109-116.

[9] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based accelerator design for deep convolutional neural networks," 2015, in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 161-170. ACM.

[10] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. Chung, "Accelerating deep convolutional neural networks using specialized hardware," 2015, february.

[11] C. C. Paige and Z. Strakoš, "Residual and backward error bounds in minimum residual Krylov subspace methods."

[12] C. C. Paige, "The effects of loss of orthogonality on large scale numerical computations."

[13] J. Drkošová, A. Greenbaum, M. Rosložník, and Z. Strakoš, "Numerical stability of GMRES," *BIT Numerical Mathematics*, vol. 35, pp. 309–330, September 1995.

[14] K. Świrydowicz, J. Langou, S. Ananthan, U. Yang, and S. Thomas, "Low synchronization Gram-Schmidt and GMRES algorithms," *Numerical Linear Algebra with Applications*, 2019.

[15] S. Cools, J. Cornelis, and W. Vanroose, "Numerically stable recurrence relations for the communication hiding pipelined conjugate gradient method," 2019, accepted in IEEE Transactions on Parallel and Distributed Systems, preprint arXiv:1902.03100.

[16] A. Björck and C. C. Paige, "Loss and recapture of orthogonality in the Modified Gram-Schmidt algorithm," *SIAM J. Matrix Anal. Appl.*, 1992.

[17] C. C. Paige, M. Rosložník, and Z. Strakoš, "Modified Gram–Schmidt (MGS), least squares, and backward stability of MGS-GMRES."

[18] J. J. Dongarra, P. Luszczek, and A. Petitet, "The LINPACK benchmark: Past, present, and future," *Concurrency and Computation: Practice and Experience*, vol. 15, no. 9, pp. 803–820, August 10 2003, dOI: 10.1002/cpe.728.

[19] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, "Introduction to the HPC benchmark suite," Lawrence Berkeley National Laboratories, Tech. Rep. LBNL-57493, 2005.

[20] P. Luszczek, J. Dongarra, and J. Kepner, "Design and implementation of the HPC Challenge benchmark suite," *CT Watch Quarterly*, vol. 2, no. 4A, 2006, http://www.ctwatch.org/quarterly/articles/2006/11/design-and-implementation-of-the-hpc-challenge-benchmark-suite/index.html.

[21] M. A. Heroux, J. Dongarra, and P. Luszczek, "HPCG technical specification," Sandia National Laboratories, Tech. Rep. SAND2013-8752, 2013.

[22] J. Dongarra, M. A. Heroux, and P. Luszczek, "High-Performance Conjugate-Gradient Benchmark: A New Metric for Ranking High-Performance Computing Systems," *International Journal of High Performance Computing Applications*, August 17 2015, dOI: 10.1177/1094342015593158. [Online]. Available: http://hpc.sagepub.com/content/early/2015/08/14/1094342015593158

978-1-7281-9219-2/20/$31.00 ©2020 IEEE

[23] ——, "High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems," *The International Journal of High Performance Computing Applications*, vol. 30, no. 1, pp. 3–10, 2016.
[24] ——, "A new metric for ranking high performance computing systems," *National Science Review*, 2016.
[25] ——, "The high-performance conjugate gradients benchmark," *SIAM News*, vol. 51, no. 1, pp. 12–12, January/February 2018.
[26] L. Gemignania, M. V. Barelb, and S. Delvaux, "Fast QR factorization of cauchy-like matrices," *Linear Algebra and its Applications*, vol. 428, pp. 697–711, 2008.
[27] P. Luszczek, J. Kurzak, I. Yamazaki, and J. Dongarra, "Towards numerical benchmark for half-precision floating point arithmetic," in *2017 IEEE High Performance Extreme Computing Conference*, 2017.
[28] P. Luszczek, I. Yamazaki, and J. Dongarra, "Increasing accuracy of iterative refinement in limited floating-point arithmetic on half-precision accelerators," in *Proceedings of High Performance Extreme Computing (HPEC) 2019*, Waltham, MA, USA, September 24-26 2019, best paper finalist.
[29] E. Carson and N. J. Higham, "A new analysis of iterative refinement and its application to accurate solution of ill-conditioned sparse linear systems," *SIAM J. Sci. Comput.*, vol. 39, no. 6, pp. A2834–A2856, 2017.
[30] A. Edelman, "Eigenvalues and condition numbers of random matrices," *SIAM J. on Mat. Anal. Appl.*, vol. 9, no. 4, pp. 543–560, October 1988.
[31] ——, "The complete pivoting conjecture for Gaussian elimination is false," *The Mathematica Journal*, vol. 2, pp. 58–61, 1992.
[32] A. Edelman and W. Mascarenhas, "On the complete pivoting conjecture for a Hadamard matrix of order 12," *Linear and Multilinear Algebra*, vol. 38, no. 3, pp. 181–188, 1995.
[33] A. Edelman, "On the distribution of a scaled condition number," *Math. Comp.*, vol. 58, no. 197, pp. 185–190, 1992.
[34] A. Edelman and H. Murakami, "Polynomial roots from companion matrices," *Math. Comp.*, vol. 64, no. 210, pp. 763–776, April 1995.

978-1-7281-9219-2/20/$31.00 ©2020 IEEE