HPL-MxP Benchmark: Mixed-Precision Algorithms, Iterative Refinement, and Scalable Data Generation

Jack Dongarra[1, 2, 3] and Piotr Luszczek[4, 1]

September 23, 2025*

Abstract

We present a mixed-precision benchmark called HPL-MxP that uses both a lower-precision LU factorization with a non-stationary iterative refinement based on GMRES. We evaluate the numerical stability of one of the methods of generating the input matrix in a scalable fashion and show how the diagonal scaling affects the solution quality in terms of the backward-error. Some of the performance results at large scale supercomputing installations produced Exascale-level compute throughput numbers thus proving the viability of the proposed benchmark for evaluating such machines. We also present the potential of the benchmark to continue increasing its use with proliferation of hardware accelerators for AI workloads whose reliable evaluation continues to pose a particular challenge for the users.

Keywords floating point representation, hardware accelerator offloading, high performance computing, numerical linear algebra, mixed-precision algorithms

Acknowledgements

Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

This research uses resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725. This research also uses resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

This work was partially funded by an NSF Collaborative Research Framework: Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC), a project of the University of Tennessee 2004541.

1 Introduction

The use of mixed-precision, while now prevalent across scientific and artificial intelligence (AI) applications, is not necessarily uniform in terms of ubiquity nor in terms of relative performance gains. As we show in the references, theory, and results below, there are gaps dependent on the discipline and data properties with the larger scientific application ecosystem. Updating these gaps in terms of simulations' needs relies on the fundamental software libraries that started offering some of the mixed-precision functionality: some of the solvers are available with proven theoretical bounds on the quality of the results. It is important to point out that the protracted sunset of Moore's Law [1] created a new avenue in the pursuit of the improvements in hardware performance, namely limiting the number of bits of the processed data thus leading to not only new floating-point precisions but also much skewed throughput rates for low-bit formats, either fixed- or floating-point. This started creating performance inversions for higher precision data as is the case for the upgrade from NVIDIA Hopper to Blackwell GPUs that lowers the performance of the IEEE standard 64-bit format.

In this environment, the HPL-MxP benchmark (formerly known as HPL-AI) seeks to highlight these emerging convergent trends and provide a unified framework for testing hardware capabilities relevant to both high-performance computing (HPC) and AI workloads. On the one hand, the traditional HPC continues focusing on high-fidelity simulations for modeling phenomena in physics, chemistry, biology, and so on. The specialized mathematical models that drive these computational endeavors require, for the majority of their data sets, 64-bit accuracy. On the other hand, the machine learning and data science methods that fuel advances in AI manage to achieve the desired results at 32-bit and now more often in even lower floating-point precision formats. This drastically lesser demand for computational accuracy fueled a resurgence of interest in both new hardware platforms and software implementations of new numerical methods. In combination, both of these advances and the HPL-MxP benchmark aim to showcase a mix of unprecedented performance levels and accompanying energy savings to achieve the simulation, classification, and recognition fidelity thought to be only possible with higher-accuracy data formats.

To resolve this newly emergent directions in hardware and software development, HPL-MxP aims to strike a balance between the divergent trends by delivering a combination of modern algorithms and efficient use of modern accelerator hardware while simultaneously drawing a connection to the established solver inside the decades-old HPL benchmark [2] and its deployment base of the largest supercomputing installations in the world. The method of choice used in the primary solver is a judicious use of the LU factorization in lower precision and the iterative refinement performed in a manner that brings the solution back to the 64-bit accuracy expected from the HPL results. The main innovation of HPL-MxP (in contrast to HPL) lies in dropping the requirement of

[0 1] University of Tennessee, [2] Oak Ridge National Laboratory, [3] University of Manchester, [4] MIT Lincoln Laboratory

THE STABILITY OF BLOCK ELIMINATIONS AND ADDITIVE MODIFICATIONS*

NEIL LINDQUIST†, PIOTR LUSZCZEK‡, AND JACK DONGARRA†

Abstract. The block elimination with additive modifications (BEAM) method was recently proposed as a alternative to LU with partial pivoting requiring less communication. Because of the novelty of BEAM, the existing theoretical analysis is lacking. To that end, we analyze both the numerical stability of the underlying block LU factorization and the effects of additive modifications. For the block LU factorization, we are able to improve the previous results of Demmel et al. from being cubic in the element growth to merely quadratic. Furthermore, we propose an alternative measure of element growth that is better aligned with block LU; this new measure of growth allows our analysis to apply to matrices that cannot be factored with pointwise LU. In the second part, we analyzed the modifications produced by BEAM and the effect they have on the condition number and growth factor. Finally, we show that BEAM will not apply any modifications in some cases that regular block LU can safely factor.

Key words. Block LU, numerical stability, growth factor, LU factorization, Gaussian elimination

MSC codes. 15A23, 65F05

1. Introduction.

Large, dense systems of linear equations are commonly solved using LU with partial pivoting; however the communication is increasingly a bottleneck on modern supercomputers. Recently, we proposed block elimination with additive modifications (BEAM) as a cheaper alternative to pivoting [14]. This method is able to achieve high performance by using the parallel dependencies of LU without pivoting, but it achieves better numerical stability by factoring the diagonal blocks with an SVD and modifying diagonal blocks with small singular values. However, there are several open questions about the numerical stability of the method, including the backward stability of BEAM's formulation of block LU and the effect of additive modifications. We study these questions to provide a better understanding of the stability of BEAM.

We begin by providing an outline of the BEAM algorithm in subsection 1.1. Then, section 2 analyzes block LU with a focus on the form that occurs in BEAM. We first propose a generalized definition of the growth factor in subsection 2.1 that better supports the blocked nature of this factorization; this growth factor is used in subsection 2.2 to improve the backward error bound of block LU. The analysis is

*Submitted to the editors DATE. Most of the analysis in the start of section 2 and in subsections 2.1 and 2.2 previously appeared in the first author's dissertation [13, Sec. 3.4.3].

Funding: This work was funded in part by the National Science Foundation under grant no. 2004541 and the Exascale Computing Project, a collaborative effort of the U.S. Department of Energy Office of Science and National Nuclear Security Administration.

Research was sponsored by the Department of the Air Force Artificial Intelligence Accelerator and was accomplished under Cooperative Agreement Number FA8750-19-2-1000. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Department of the Air Force or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein.

†Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA 37996 (nlindquist@acm.org, dongarra@icl.utk.edu)

‡MIT Lincoln Laboratory, Lexington, MA, USA 02421 and Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA 37902 (luszczek@icl.utk.edu)

ANALYSIS OF FLOATING-POINT MATRIX MULTIPLICATION COMPUTED VIA INTEGER ARITHMETIC*

AHMAD ABDELFATTAH†, JACK DONGARRA†‡, MASSIMILIANO FASI§, MANTAS MIKAITIS§, AND FRANÇOISE TISSEUR‡

Abstract. Ootomo, Ozaki, and Yokota [*Int. J. High Perform. Comput. Appl.*, 38 (2024), p. 297–313] have proposed a strategy to recast a floating-point matrix multiplication in terms of integer matrix products. The factors A and B are split into integer *slices*, the product of these slices is computed exactly, and AB is approximated by accumulating these integer products in floating-point arithmetic. This technique is particularly well suited to mixed-precision matrix multiply–accumulate units with integer support, such as the NVIDIA tensor cores or the AMD matrix cores. The number of slices allows for performance-accuracy tradeoffs: more slices yield better accuracy but require more multiplications, which in turn reduce performance. We propose an inexpensive way to estimate the minimum number of multiplications needed to achieve a prescribed level of accuracy. Our error analysis shows that the algorithm may become inaccurate (or inefficient) if rows of A or columns of B are *badly scaled*. We perform a range of numerical experiments, both in simulation and on the latest NVIDIA GPUs, that confirm the analysis and illustrate strengths and weaknesses of the algorithm.

Key words. matrix multiplication, floating-point arithmetic, integer arithmetic, tensor cores, mixed-precision, fixed-point arithmetic, error analysis.

MSC codes. 65F99, 65G50, 65Y10

1. Introduction.

The top three computers on the November 2024 Top500 list,[1] El Capitan, Frontier, and Aurora, are exascale supercomputers, capable of performing over 10^{18} floating-point operations per second (flop/s) in binary64 arithmetic, whose accuracy is essential for most scientific applications.

To achieve their impressive performance, modern supercomputers leverage hardware accelerators designed for machine-learning workloads, which typically do not require high precision and can provide meaningful results if fewer-than-32-bit floating-point arithmetics are used. Formats such as TensorFloat-32, bfloat16, and binary16 are widely available in hardware, and more recently vendors have started developing 8-bit formats for training and inference of deep neural networks: Graphcore has proposed three such formats [35], two of which are available in the Tile Vertex ISA [17]; NVIDIA, Arm, and Intel have proposed two [32], subsequently crystallized in the Open Compute Project 8-bit floating-point specification (OFP8) [31]; Tesla has proposed the Configurable Float8 format in its Dojo Technology white paper [46]; and Huawei has proposed the Ascend HiFloat8 format [28]. More examples can be found in the interim report of the IEEE P3109 working group [1], which is currently developing a standard for arithmetic formats for machine learning. To further complicate the landscape, integer arithmetic is often preferred for inference [50], and hardware accelerators are starting to be optimized for compact integer formats. The main features

*Version of 12 June 2025.

Funding: The work of the last author was supported by Engineering and Physical Sciences Research Council grant EP/W018101/1.

†Innovative Computing Laboratory, University of Tennessee, Knoxville, TN, USA (ahmad@icl.utk.edu, dongarra@icl.utk.edu)

‡Department of Mathematics, University of Manchester, Oxford Road, Manchester M13 9PL, UK (francoise.tisseur@manchester.ac.uk)

§School of Computer Science, University of Leeds, Woodhouse Lane, Leeds LS2 9JT, UK (m.fasi@leeds.ac.uk, m.mikaitis@leeds.ac.uk)

[1] https://www.top500.org/lists/top500/list/2024/11/

Hardware Trends Impacting Floating-Point Computations In Scientific Applications

Jack Dongarra
University of Tennessee
Oak Ridge National Laboratory
University of Manchester
Oak Ridge, TN, USA
dongarra@icl.utk.edu

John Gunnels
NVIDIA Corporation
Santa Clara, CA, USA
jgunnels@nvidia.com

Harun Bayraktar
NVIDIA Corporation
Santa Clara, CA, USA
hbayraktar@nvidia.com

Azzam Haidar
NVIDIA Corporation
Santa Clara, CA, USA
ahaidarahmad@nvidia.com

Dan Ernst
NVIDIA Corporation
Santa Clara, CA, USA
dane@nvidia.com

***Abstract*—The evolution of floating-point computation has been shaped by algorithmic advancements, architectural innovations, and the increasing computational demands of modern technologies, such as artificial intelligence (AI) and high-performance computing (HPC). This paper examines the historical progression of floating-point computation in scientific applications and contextualizes recent trends driven by AI, particularly the adoption of reduced-precision floating-point types. The challenges posed by these trends, including the trade-offs between performance, efficiency, and precision, are discussed, as are innovations in mixed-precision computing and emulation algorithms that offer solutions to these challenges. This paper also explores architectural shifts, including the role of specialized and general-purpose hardware, and how these trends will influence future advancements in scientific computing, energy efficiency, and system design.**

***Index Terms*—floating-point, computer architecture, GPU, CPU, emulation, mixed-precision**

I. Introduction

Floating-point computation is foundational to modern scientific applications, enabling the representation of real numbers across a wide range of magnitudes and providing the precision necessary for calculations in fields like physics, chemistry, and engineering. Over the decades, the evolution of floating-point computation has been influenced by the increasing complexity of scientific problems, technological advancements, and the rise of new computational paradigms, such as deep neural network- (DNN-) based AI algorithms [1].

This paper explores the history of floating-point computation, focusing on architectural innovations that have shaped the current landscape. It examines key developments, from early emulation to dedicated hardware, and highlights recent trends, including mixed-precision computing and reduced-precision floating-point types (Figure 1). The impact of these trends on scientific computing and AI is analyzed, along with the challenges they present regarding system design, energy efficiency, and performance.

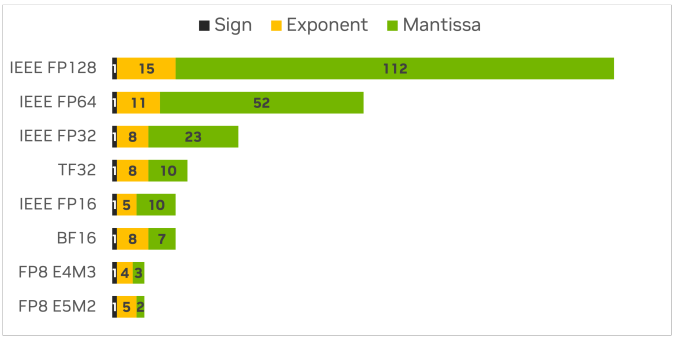

Fig. 1. Various floating-point (FP) representations used today in scientific computing and AI. The exponent bits determine the dynamic range of the FP number, while the mantissa bits determine the precision. Four IEEE FP types are shown: half (FP16), single (FP32), double (FP64), and quad (FP128). TensorFloat-32 (TF32), available on NVIDIA GPUs starting with the Ampere architecture, is a Tensor Core matrix multiply compute mode where input and output are FP32, but input operands are truncated. Bfloat16 (BF16), which was introduced by Google [2], has the same range as FP32 at the expense of mantissa bits. Two variants of FP8, with different splits of exponent and mantissa bits [3] are shown.

II. Benchmarks

Throughout this paper, we will refer to several community benchmarks that have emerged over time, each serving a critical role in evaluating and exposing performance characteristics and limitations of the underlying hardware, as well as providing a readily conveyed, widely understood record of progress. These benchmarks have become standard tools in the high-performance computing (HPC) community for assessing the efficiency and effectiveness of various computing systems. While in this paper, we focus on floating-point operation-focused benchmarks, other benchmarks exist. One such example is the Graph 500 [4], which measures a system's performance on graph-based problems important for large dataset analysis.

The most well-known benchmark is HPL [5] (High-