Prospectus for the Next LAPACK and ScaLAPACK Libraries: Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC)*

James Demmel
University of California at Berkeley

Jack Dongarra
University of Tennessee, Oak Ridge National Laboratory, and University of Manchester

Julie Langou
University of Tennessee

Julien Langou
University of Colorado Denver

Piotr Luszczek
University of Tennessee

Michael W. Mahoney
ICSI and University of California at Berkeley

July 13, 2020

Abstract: The convergence of several unprecedented changes, including formidable new system design constraints and revolutionary levels of heterogeneity, has made it clear that much of the essential software infrastructure of computational science and engineering is, or will soon be, obsolete. Math libraries have historically been in the vanguard of software that must be adapted first to such changes, both because these low-level workhorses are so critical to the accuracy and performance of so many different types of applications, and because they have proved to be outstanding vehicles for finding and implementing solutions to the problems that novel architectures pose. Under the Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC) project, the principal designers of the Linear Algebra PACKage (LAPACK) and the Scalable Linear Algebra PACKage (ScaLAPACK), the combination of which is abbreviated Sca/LAPACK, aim to enhance and update these libraries for the ongoing revolution in processor architecture, system design, and application requirements by incorporating them into a layered package of software components—the BALLISTIC ecosystem—that provides users seamless access to state-of-the-art solver implementations through familiar and improved Sca/LAPACK interfaces.

The set of innovations and improvements that will be made available through BALLISTIC is the result of a combination of inputs from a variety of sources: the authors' own algorithmic and software research, which attacks the challenges of multi-core, hybrid, and extreme-scale system designs; extensive interactions with users, vendors, and the management of large high-performance computing (HPC) facilities to help anticipate the demands and opportunities of new architectures and programming languages; and, finally, the enthusiastic participation of the research community in developing and offering enhanced versions of existing dense linear algebra software components. Aiming to help applications run portably at all levels of the platform pyramid, including in cloud-based systems, BALLISTIC's technical agenda includes: (1) adding new functionality requested by stakeholder communities; (2) incorporating vastly improved numerical methods and algorithms; (3) leveraging successful research results to transition Sca/LAPACK (interfaces)

*Work supported in part by a grant from the NSF OAC: 2004235/2004541/2004763/2004850 (and collaborative proposals) – Collaborative Research: Frameworks: Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC).

to multi-core and accelerator-enabled versions; (4) providing user-controllable autotuning for the deployed software; (5) introducing new interfaces and data structures to increase ease of use; (6) enhancing engineering for evolution via standards and community engagement; and (7) continuing to expand application community outreach. Enhanced engineering will also help keep the reference implementation for Sca/LAPACK efficient, maintainable, and testable at reasonable cost in the future.

The Sca/LAPACK libraries are the community standard for dense linear algebra. They have been adopted and/or supported by a large community of users, computing centers, and HPC vendors. Learning to use them is a basic part of the education of a computational scientist or engineer in many fields and at many academic institutions. No other numerical library can claim this breadth of integration with the community. Consequently, enhancing these libraries with state-of-the-art methods and algorithms and adapting them for new and emerging platforms (reaching up to extreme scale and including cloud-based environments) is set to have a correspondingly large impact on the research and education community, government laboratories, and private industry.

1 Introduction

The Linear Algebra PACKage (LAPACK) and the Scalable Linear Algebra PACKage (ScaLAPACK), referred to in combination as Sca/LAPACK, are community standards for dense linear algebra and have been adopted and/or supported by a large community of users, computing centers, and high-performance computing (HPC) vendors. However, the convergence of several radical changes in microprocessor and system design has, unfortunately, also rendered most legacy software infrastructure—including numerical libraries—obsolete [1–4]. The **Basic ALgebra LIbraries for Sustainable Technology with Interdisciplinary Collaboration (BALLISTIC)** project aims add these capabilities to Sca/LAPACK to provide legacy users with seamless access to the appropriate implementations of state-of-the-art solvers without the user having to learn the details of a new library interface.

Accordingly, BALLISTIC will create a layered package of software components that is capable of running at every level of the platform deployment pyramid and achieve three complementary objectives: (1) deliver seamless access to the most up-to-date algorithms, numerics, and performance via familiar Sca/LAPACK interfaces, wherever possible; (2) make advanced algorithms, numerics, and performance capabilities available through new interface extensions, wherever necessary; and (3) provide a well-engineered conduit through which new discoveries at the frontiers of research in these areas can be channeled as quickly as possible to all application communities that depend on high-performance linear algebra libraries.

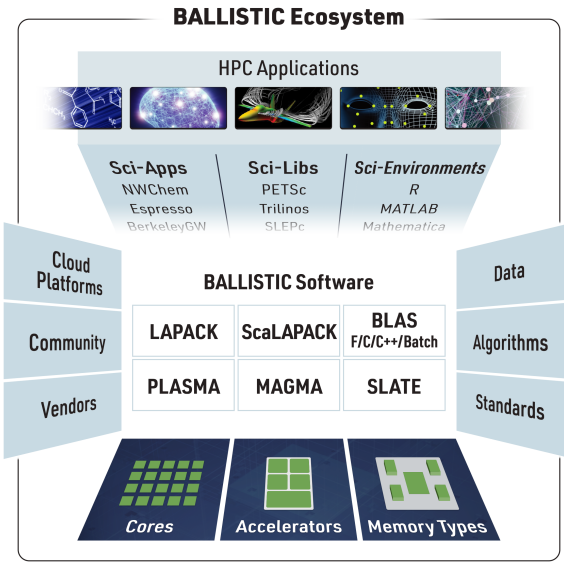

Figure 1: The BALLISTIC software ecosystem: BALLISTIC software within the context of applications, libraries, and environments it supports and the hardware technologies it must exploit.

Building on a community engagement effort that we plan to replicate in BALLISTIC, we have identified the improvements and innovations that we will include in the project by combining input from various sources: (1) the results (including designs and well-tested prototypes) of our own algorithmic and software research agenda, which has targeted multi-core, hybrid, and extreme-scale system architectures; (2) extensive and ongoing interactions with users, vendors, and the managers of large National Science Foundation (NSF) and US Department of Energy (DOE) supercomputing facilities to help anticipate the demands and opportunities of new architectures and modern programming languages; (3) through a survey of the Sca/LAPACK user base [5] and through cross-disciplinary engage-

ment with other areas of computer science and engineering that have helped us anticipate the demands and opportunities of new architectures and programming models; and from (4) the enthusiastic participation of the research community in developing and offering enhanced versions of existing Sca/LAPACK codes. Indeed, regarding item (4), papers proposing new algorithms typically compare their performance with that of Sca/LAPACK, and, over the years, several researchers have developed better algorithms that they would like to provide to us. In some cases, they have even done the same level of careful software engineering and testing that we insist on; in other cases, we must do this; and in yet other cases, much of the mathematical development remains. The opportunity to harvest this bounty of good ideas (and free labor) is not to be missed, and the plan we developed for BALLISTIC incorporates this input.

1.1 Overview of Projected Contributions

Building on this community input, we expect that the BALLISTIC project will make contributions to the whole spectrum of innovation—from concept to application. It is necessary, on the one hand, to develop new ideas to revolutionize the underlying numerical methods and algorithms. Equally important, however, is the need to implement the numerical algorithms in software and integrate them into real-world applications. Indeed, only algorithms implemented in new software and integrated into real-world applications can have any impact on the use of the extreme-scale HPC systems of the future. Hence, a project of this magnitude must ambitiously attempt to contribute from concept to application. To make such a feat possible, the project needs to be limited to a set of fundamental operations that are widely used in practice. BALLISTIC has been designed in precisely this way. In particular, BALLISTIC has a focus on developing new ideas related to dense and banded linear systems, least squares problems, and eigenvalue solvers into reusable software. Finally, the results obtained from these developments will be integrated into a number of challenging real-world applications by our application colleagues. In this way, BALLISTIC will develop new ideas and put them to use in real applications within a very short span.

Targeting the scientific community in general, and the NSF community in particular, the technical focus of our efforts include:

(1) science-driven development based on established collaborations (§2.1)
(2) adding new functionality requested by users and stakeholder communities to BALLISTIC (e.g., new mathematical functions); (§2.2)
(3) incorporating vastly improved numerical methods and algorithms (these could be new algorithms for existing mathematical functions); (§2.3)
(4) providing user-controllable autotuning capability for the deployed software; (§2.4)
(5) enhancing and maintaining collaborations with academia and industry (§2.5)
(6) continuing to expand interdisciplinary collaborations. (§2.6)
(7) enhancing engineering for evolution and growth via standards and community engagement; (§2.7)
(8) leveraging successful research (e.g., in Parallel Linear Algebra Software for Multi-core Architectures [PLASMA] [6], the Matrix Algebra on GPU and Multi-core Architectures [MAGMA] library [7], and Software for Linear Algebra Targeting Exascale [SLATE] [8]) to transition Sca/LAPACK (interfaces) to multi-core and off-load, accelerator-enabled versions; (§2.8) and
(9) introducing new interfaces and data structures that make using the library easier (§2.9).

Enhanced engineering will also help us maintain the reference implementation for Sca/LAPACK, thereby keeping the substantial code base efficient, maintainable, and testable at a reasonable cost in the future.

The BALLISTIC project, through the leading-edge research it channels into its software deliverables, will lead to the introduction of tools that will simplify the transition to the next generation of extreme-scale computer architectures. The main impact of the project will be to develop, push, and deploy software into the scientific community to make it competitive on a world-wide scale and to contribute to standardization efforts in the area.

2 Proposed Work

BALLISTIC will transfer the research and development that went into our Sca/LAPACK, MAGMA, PLASMA, and SLATE dense linear algebra libraries and engineer them specifically for HPC and cloud-based environments. Applications that currently require any of these libraries will be able to seamlessly replace them with BALLISTIC. We will provide routines for solving systems of linear equations, least-squares problems, symmetric and non-symmetric eigenvalue problems, and singular-value problems. In all areas, similar functionality will be provided for real and complex matrices in both single and double precision. New precisions (lower and higher) will be added to support specific needs of some new applications. Mixed-precision, iterative-refinement solvers will be available along with explicit matrix inversion routines.

To accomplish the above, our main objectives will be to (1) maintain and evolve the existing Sca/LAPACK software base; (2) develop novel software that exposes as much parallelism as possible, exploits heterogeneity, avoids communication bottlenecks, and helps meet emerging power constraints; (3) explore advanced scheduling strategies and runtime systems, focusing on the extreme scale and strong scalability in multi/many-core and hybrid environments; and (4) design and evaluate novel strategies and software support for both offline and online autotuning.

Supported architectures will include multi-core CPUs and GPUs and extensions for cloud-based environments. Currently, different architectures are supported through different libraries. For example, LAPACK and PLASMA are for shared-memory multi-core systems; MAGMA is for NVIDIA GPUs, with a Heterogeneous-Compute Interface for Portability (HIP) port in progress for AMD GPUs; and ScaLAPACK and SLATE are for distributed-memory systems. BALLISTIC will *unify support for these architectures into a single package*. The BALLISTIC software stack, as we envision it in this context, is illustrated in Figure 1.

In summary, the proposed library will: (1) use the same calling sequence as Sca/LAPACK. (2) "Under the hood," the routines will decide when to use existing Sca/LAPACK software or when to switch to "modern" implementations: (a) the user will be notified of a switch in routines via a flag; (b) the user can override the switch via a flag; (c) the decision to change the underlying software invocation will be made based on the size, architecture, and estimated time to solution; and (d) BALLISTIC will draw on PLASMA, MAGMA, and SLATE routines if they provide the best fit, which (i) may require data motion to use the "best fit," and (ii) data motion may be taken into consideration when making the decision. (3) The user can call the architecture-specific routines directly and bypass the legacy interface. (4) We will update the existing Sca/LAPACK routines to the state of the art to enhance numerics where appropriate. (5) Finally, we will add additional functionality to the existing Sca/LAPACK libraries, where the state of the art has changed.

The validation and dissemination of results will be done by maintaining and integrating new software solutions into challenging scientific applications in computational science domains. A number of leading application groups have agreed to collaborate in this effort. The deliverables also include a sustainable set of methods and tools for cross-cutting issues, such as scheduling and autotuning, packaged into open-source library modules.

BALLISTIC's plan to overhaul the Sca/LAPACK software framework is split into seven thrusts outlined in Section 1.1. The individual tasks corresponding to the thrusts are detailed throughout this section. These tasks will update the Sca/LAPACK numerics and algorithmic foundations to accommodate multi-core and hybrid architectures and allow for scalability, portability, high performance, and transparency for existing and future users. The tasks are organized to be roughly orthogonal. For example, a new mathematical function may require a different algorithm on each target architecture with its own corresponding performance tuning, interfaces, and software engineering infrastructure. This naturally leads to a very large number of possible tasks. This motivates us to: (1) reuse as much infrastructure as possible; (2) prioritize the many possible tasks by engaging with the user community; and (3) encourage the community to make contributions.

2.1 Science-Driven: Application Domains Benefiting from BALLISTIC Technology

Dense linear systems are found in many applications, including airplane wing design, radar cross-section studies, molecular dynamics, electronic structure calculations, cloud computing, and machine learning. See §2.6 for more details about some of our close collaborators and outreach to other communities.

2.1.1 Software libraries that use LAPACK. Many software libraries require an explicit link to an LAPACK-interface library. Examples include (a) high-level interfaces: Armadillo [9], blaze [10], MATLAB, NumPy [11], and R [12]; (b) solvers: Hypre [13], MUMPS [14], PETSc [15], SuperLU [16], and Trilinos [17]; (c) electronic-structure packages: ABINIT [18], Quantum ESPRESSO [19], and VASP [20]; and (d) finite element packages: deal.II [21] (2007 Wilkinson Prize for Numerical Software) and OpenSees [22].

2.2 Innovation with New Algorithms, Functionality, and Approaches

2.2.1 Add new algorithmic functionality to BALLISTIC. Requests for new functionality fall into three categories: (1) new mathematical functions, (2) new variants of existing functions, and (3) functionality present in LAPACK but missing from our other libraries. We consider these in turn as subtasks below.

2.2.2 Add new mathematical functions. We will integrate new algorithms as they become available. We have integrated the following new functionalities: **(1)** in LAPACK 3.9.0 (Nov 2019), we integrated QR-preconditioned singular value decomposition (SVD), Householder reconstruction; **(2)** in LAPACK 3.8.0 (Nov 2017), we integrated 2-stage Aasen tridiagonalization; **(3)** in LAPACK 3.7.0 (Dec 2016), we integrated TSQR and related functions, Aasen tridiagonalization, new data structure for Bunch-Kaufman and Rook-pivoting, 2-stage tridiagonalization for eigenproblem, and improved complex Jacobi SVD; **(4)** in LAPACK 3.6.1 (June 2016), we integrated the new blocked-back transformation for the non-symmetric eigenvalue problems [23]; **(5)** in LAPACK 3.6.0 (Nov 2015), we integrated the new blocked Hessenberg-triangular reduction routines from Kågström et al. [24] and Jacobi SVD [25, 26]; **(6)** in LAPACK 3.5.0 (Nov 2013), we integrated the rook pivoting LDL^T factorization from Ashcraft et al. [27]; **(7)** in LAPACK 3.4.0 (Nov 2011), we integrated TPQRT and GEQRT3 [28]; and **(8)** in LAPACK 3.3.0 (Nov 2010), we integrated the new CS decomposition from Sutton [29]. We strive for a short turnaround from research to software by ensuring community involvement. In general, the authors of the new algorithm will contribute their research code, and others will turn them into robust software. Below, we list algorithms that we plan to integrate into BALLISTIC—some are recent discoveries, while others are older requests.

2.2.3 Rank-revealing factorizations. The "gold standard" of such factorizations is the (truncated) SVD, which is either too expensive for many big-data applications or does not meet the needs of users who prefer an "interpolative decomposition" (sometimes called CX or CUR decompositions [32, 33]) that factors the matrix $A = CX$, where C is a subset of the columns of A. In either case, the performance goal, for matrices of low-rank $r \ll n = \#\text{cols} \leq m = \#\text{rows}$, is to perform $O(rmn)$ floating point operations per second (FLOP/s) while minimizing communication (discussed in more detail below). A variety of algorithms have been proposed for these factorizations—both deterministic and randomized [30, 32, 34, 35]—that exhibit various trade-offs between performance and the quality of the factorization (Figure 2). We will decide which of these is most important and should be implemented first.

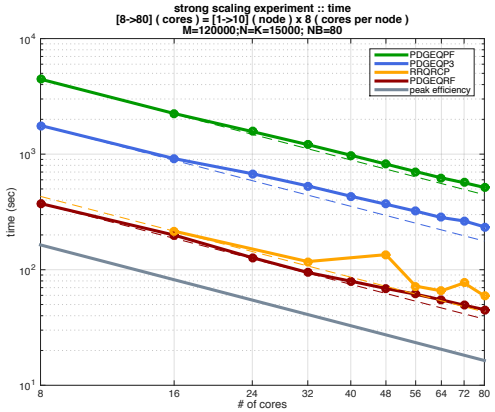

Figure 2: Preliminary scalability results of a new randomized QR algorithm with column pivoting [30, 31] implemented in ScaLAPACK. The new algorithm (orange) is much faster and scales much better than our current PDGEQPF and PDGEQP3. It scales as well and is almost as fast as QR without pivoting (red).

2.2.4 Other randomized algorithms. We will also incorporate the functionality underlying recent advances in Random-

ized Numerical Linear Algebra (RandNLA) [34, 36] to enable users to build future RandNLA methods directly on our code. Given an $m \times n$ matrix A as input, existing RandNLA algorithms use an approximate matrix-matrix multiplication of the form $A\Omega$ as a primitive, where Ω is an $n \times \ell$ sketching or projection matrix, and ℓ is an input parameter. If A is structured in such a way that an arbitrary matrix can be applied to it quickly, then Ω can consist of Gaussian/normal random variables. If A is arbitrary, then Ω is typically either (1) a Hadamard-based orthogonal transformation, to which fast Fourier techniques can be applied; (2) a Rademacher-based random projection, which exploits discrete random variables; or (3) a CountSketch-based random projection, which exploits structured random variables to achieve enhanced sparsity. Our focus will be two-fold: first, we will provide implementations of core primitives from RandNLA; and second, we will expose the appropriate information to the user to facilitate the diagnosis of potential failures caused by randomness. For the former focus, where a high-quality form (e.g., Rademacher-based and CountSketch-based sketches) is unavailable, we will provide implementations of these random projections. For the latter focus, we will exploit the fact that there are two main use cases for RandNLA methods. The first is a "sketch and solve" paradigm, in which there is a non-negligible chance of failure due to randomness in the algorithm; here, we will expose information to the user to enable diagnosis. The second is a "sketch and precondition" paradigm, in which the random sketch is used as a preconditioner, and thus the algorithms (e.g., Blendenpik [37] and LSRN [38]) tend to be more traditional and more resistant to randomness-induced failure. We will use these primitives in Section 2.3.5.

2.2.5 Low-rank updates. There are well-known algorithms that can take a one-sided matrix factorization (e.g., LU, Cholesky, QR) of a matrix A and quickly compute the corresponding factorization of the matrix $A + B$, where B has low-rank r, with cost $O(rmn)$. These were requested recently.

2.3 Incorporating Improved Functionality to BALLISTIC.

We will incorporate improved variants of existing routines based on research results and reviewing/revising existing implementations as described in this section.

2.3.1 Communication-avoiding methods based on our optimality results. In [39], which earned a SIAM best paper award for linear algebra, we extended known communication lower bounds for $O(n^3)$ dense sequential [40] and parallel [41] matrix multiplication to all direct linear algebra. This includes solving linear systems, least squares, eigenvalue problems, and SVD and the sequences of such operations for dense or sparse matrices on sequential or parallel machines. In another award-winning paper, [42, 43], we extended these lower bounds to Strassen-like algorithms (see also [44]). Given these lower bounds, we asked whether the current algorithms in Sca/LAPACK attained them, and in most cases they did not [39, Section 6], not even for parallel matrix multiplication. This led us to systematically invent new communication-avoiding (CA) algorithms that attain these lower bounds and can provide large speedups. These algorithms are described in a long sequence of papers on dense linear algebra algorithms for the Basic Linear Algebra Subprograms (BLAS), linear systems, least squares problems, eigenvalue problems, and SVD [45–69], as well as sparse-direct linear algebra [70], sparse-iterative linear algebra [71–79], tensor contractions [80–82], and graph theory [83]. We address both homogeneous and heterogeneous parallel machines in [58, 84–88].

2.3.2 Recursive algorithms. In dense linear algebra, recursive algorithms are communication avoiding by nature and enable oblivious cache efficiency [89]. Recursive algorithms are of significant interest, and we have already released three recursive algorithms in LAPACK, including Recursive LU (LAPACK 3.2, 2008), Recursive QR (LAPACK 3.4, 2011), and Recursive Cholesky (LAPACK 3.6, 2015). Recent work [90] also stresses the importance of recursive algorithms in dense linear algebra, and, as part of our work in BALLISTIC, we plan to include more recursive algorithms in LAPACK. The first step would be to rewrite some of the BLAS functionalities and then move on to inversion subroutines, $\mathrm{LDL}^{\mathrm{T}}$, and Householder reflector application subroutines. In [69], we proved that cache-oblivious algorithms cannot be write avoiding, so they may not be suitable for emerging nonvolatile memory technologies, where writes are much more

expensive than reads. For this, we will need to implement new algorithms like those in [69]. So, while recursive algorithms will not always be the best solution, they are a good solution in some important situations.

2.3.3 Reduction to tridiagonal and biadiagonal. The most time-consuming phase of eigensolvers for dense hermitian matrices involves a reduction of the input matrix to tridiagonal form. ELPA [91] implements a two-step reduction that is efficient for large matrices and core counts. In turn, EigenEXA [92] implements a reduction to pentadiagonal form and applies a modified version of divide-and-conquer [93]. We plan to explore ideas introduced in [61] for band reduction to improve the performance of our parallel eigensolvers.

2.3.4 Small bulge multi-shift QZ algorithm. The QR algorithm is useful for computing the Schur form of a nonsymmetric matrix, which in turn is useful for computing the eigen decomposition of a nonsymmetric matrix, which is the algorithm behind Matlab's `eig(A)` command. In LAPACK 3.1 (2006), we released a new QR algorithm with a small bulge multi-shift and early aggressive deflation strategy following the work of Braman, Byers, and Mathias [94, 95]. This algorithm provides a major speedup to the QR algorithm for computing the eigenvalues of a nonsymmetric matrix. Kågström and Kressner [96] presented a similar improvement for the QZ algorithm. The QZ algorithm is useful for computing the generalized Schur form of a nonsymmetric pencil, which in turn is useful for computing the eigen decomposition of a generalized nonsymmetric pencil, which is the algorithm behind Matlab's `eig(A,B)` command. As part of our proposed work, we plan to write a QZ algorithm with a small bulge multi-shift and early aggressive deflation.

2.3.5 Randomized Numerical Linear Algebra (RandNLA). RandNLA is an area, described as "arguably the most exciting and innovative idea to have hit linear algebra in a long time" [37], that exploits randomness for the development of improved matrix algorithms for problems like least-squares regression and low-rank matrix approximation [36]. Implementations of these algorithms can beat LAPACK in wall-clock time [37], and the randomness has also been used to obtain improved communication properties [38]. As part of our proposed work, we plan to incorporate recent developments from RandNLA to improve the performance of our core algorithms for least-squares and low-rank matrix approximation, building on our randomization developments in Section 2.2.

2.4 Autotuning Facilities

BALLISTIC's autotuning efforts will expand on our ongoing work to build the GPTune autotuner [97], which is based on machine learning—specifically multitask and transfer learning. GPTune is designed to tune codes as "black-boxes," running them for carefully chosen tuning parameter values and building a performance model based on measured runtimes that can then be optimized; and it can also be guided by existing (approximate) performance models. We have achieved a performance improvement of up to 1.5× over the existing Opentuner [98] and HpBandSter [99] autotuners for tuning ScaLAPACK's PDGEQRF on the Edison supercomputer. Our Python interface also enables easy use of these autotuners, in case one is better than another. The clear separation of the libraries and the autotuner will also enable others (e.g., vendors) to develop their own autotuners without the need to augment the libraries' codes. We plan to use autotuning to improve performance, and performance improvement over previously developed software and packages will be one of our metrics in determining and demonstrating that the software being developed has been an improvement over other software packages.

2.5 Close Collaboration Among Stakeholders: Academia and Industry

2.5.1 Engaging the community and providing leadership on standards. As noted in Section 1, the authors have a strong track record of reaching out to, and receiving the input from, various stakeholders in the success of their software. This project will energetically continue that tradition. Our work with the community to establish new standards deserves special attention. Because of the velocity of change in both hardware and software, the field of numerical libraries has historically received strong benefits from adopting standards. Therefore, one of BALLISTIC's objectives is to promote the creation, in a fast and flexible way, of standards

related to the project through close collaboration between the project partners, other prominent research groups, and industry leaders. Accordingly, we plan to engage with the stakeholder community in community-based standardization efforts—specifically to formulate requirements for specialized high-performance linear algebra kernels for small matrices (Batched BLAS) for heterogeneous systems, which would help a variety of emerging data-intensive applications. In conjunction with our application collaborators, we will work with hardware vendors and seek other agency funding to support the workshops required to create a new standard API for Batched BLAS. Also, the standardization efforts will extend to runtime system interoperability issues and the interface to autotuning frameworks.

2.5.2 Groups that provide "improved" (sometimes proprietary) versions of LAPACK. Various groups provide improved versions of LAPACK or at least "binaries" for easy installation. In this category, we include IBM ESSL, Intel MKL, Cray LibSci, Debian, Cygwin, Red Hat/Fedora, and the Apple Accelerate Framework. We collaborate with some of these groups, helping them with the particularities of our software or incorporating their patches and suggestions into ours.

We are certainly not opposed to a rewrite of our software. Indeed, within the context of the DOE Exascale Computing Project (ECP), PI Dongarra initiated the SLATE project [8] as a modern C++ replacement for ScaLAPACK, targeting modern HPC machines with multi-core CPUs and multiple hardware accelerators per node. While SLATE has an entirely new C++ interface, it also maintains a ScaLAPACK compatible interface where possible. BALLISTIC will leverage the developments in SLATE and maximize communication with the SLATE team for collaboration. We also have C++ prototypes of a new templated LAPACK library and intend to pursue this further. In fact, BLAS++ and LAPACK++ libraries, that the SLATE project exclusively depends on, are already used in scientific applications such as NwChemEx. However, we believe that maintaining and improving our existing software base is essential to the sustainability of the HPC software ecosystem, since it has long served as a basic reference software implementation and a reliable indicator of progress. To wit, not only is LAPACK an essential building block of other libraries (e.g., MAGMA and SLATE), some of this work also provides the building blocks for patents filed by others [100].

2.6 Continue to expand interdisciplinary collaborations.

Over the years, the Sca/LAPACK libraries have been very successful by providing the foundational dense linear algebra that many computational science and engineering domains require. The identification of these needs has been established through user surveys, workshops, meetings, and other user/developer interactions. More importantly, the development of new functionalities for emerging needs in computational science and engineering domains has been accomplished through interdisciplinary collaborations. We are building on this track record in BALLISTIC. As mentioned earlier, we have conducted a survey to identify needs and have established a number of interdisciplinary collaborations for their development in areas like: (1) complex flows in multi-scale systems using adaptive, finite volume methods (Chombo-Crunch [101]); (2) molecular dynamics (GROMACS [102]); (3) electronic structure calculations (Quantum ESPRESSO [19]); (4) geophysical imaging technologies combined with seismic methods for fossil fuel deposits and reservoir studies (EMGeo package [103]); (5) lattice QCD (Chroma library [104]); (6) DFT (BerkeleyGW [105] and MADDNESS [106]); (7) high order FEM (tensor contractions in packages like MFEM [107] and Nektar++ [108]); and (8) problems in magnetism, disordered alloys, and bulk amorphous metals with the LSMS method [109]. Interdisciplinary work is required, because new requirements for dense linear algebra algorithms are usually linked with application specifics that domain scientists can provide and must be exploited for the development of high-performance algorithms. Examples include specific matrix sizes (e.g., QR for tall-skinny matrices) that may require special versions that cannot be generated by autotuning generic parameterized implementations and must select arithmetic precision based on application specifics, applicability of low-rank approximations, design of new data abstractions and algorithms (e.g., for tensor contractions [82]), batch computations, and others.

2.7 Enhance engineering for evolution and growth via standards and community engagement.

We will ensure the longevity of BALLISTIC beyond the individual tasks and will continue to incorporate new software standards and update current software standards by: (1) using the OpenMP standard for multithreading to ensure portability between current and future multi-core architectures (e.g., Intel x86, ARM, IBM POWER); (2) using the OpenMP standard for accelerator offload to guarantee portability between current and future hardware accelerators (e.g., NVIDIA, AMD, and Intel); (3) ensuring that all code is compliant with recent language standards C (C11 and C18) and C++ (C++11, C++14, and C++17) and providing standard-compliant Fortran (2003, 2008, and 2018) interfaces; and (4) by maximizing the use of standard numerical library components like BLAS.

The long-term growth of the software will be supported through involving the broader community in the software development and maintenance process, specifically by: (1) hosting the project in a public repository that allows for immediate community feedback and community contributions through the mechanism of pull requests; (2) by engaging hardware vendors in the package adoption process (e.g., Intel MKL, Cray LibSci, and IBM ESSL); (3) by engaging software companies in the the package adoption process (e.g., MathWorks' MATLAB); and (4) by engaging the broader community through a public mailing list, including the hardware and software industry and academic institutions.

2.8 Leveraging Successful Research and Building on Established Technology

It is important to note that Sca/LAPACK are used in many-higher level tools (e.g., MATLAB, NumPy, SciPy, Julia, R, PETSc, and Trilinos). When an application scientist types `A\b` or `eig(A)` or `svd(A)`, they are most probably using one of the BALLISTIC packages in the background. BALLISTIC packages are well integrated, and the industry cares deeply about supporting them, because many application domains benefit from these packages. Section 4.1 describes how BALLISTIC fits into the national research infrastructure.

2.9 Project Plans and System and Process Architecture

In addition to the innovations in Section 2.2, BALLISTIC will pursue the following.

2.9.1 Extend scope and applicability of existing functions. The next three categories are topics involving the relaunched BLAS Standard Committee [110]. The first meeting was held in May of 2016 and included participants from industry (Intel, ARM, NVIDIA, Cray, MathWorks, and NAG), government laboratories (Sandia National Laboratories and Rutherford Appleton Laboratory), and universities (UTK, UCB, UIUC, LSU, University of Manchester, Umea University, and KAUST).

2.9.2 Batched BLAS and higher-level operations. Many users, including 35% of those in our survey [5], want to perform the the same operation—simultaneously—on many independent problems (e.g., matrix-multiplications or Cholesky). When the problems are small or medium sized, this approach can enable efficient parallelization and avoid overhead. We had already been, along with several vendors, working on these "batched" operations, and so we now have an opportunity to agree on a standard interface and semantics.

2.9.3 Reproducible BLAS and higher-level operations. Floating-point reproducibility is no longer guaranteed, because dynamically scheduled parallel resources may compute sums in different orders from run to run, and floating-point summation is not associative because of roundoff. This has serious implications for debugging and correctness, and 78% of the users in our survey said reproducibility was important, very important, or critical, and wanted it as an option for debugging, code validation, and verification—even with a (modest) performance penalty.

We recently developed an algorithm for summing floating-point numbers reproducibly, and also dot products and higher-level BLAS, that—unlike prior work—satisfies the design goals of making one pass over the data [111]. We propose to provide complete reference implementations of our reproducible BLAS on various target architectures, determine which higher-level operations are then automatically reproducible,

and—for those that are still not reproducible—try to create reproducible versions of them, too. To partially address this need, Intel recently released a (nonscalable) reproducible version of MKL [112]. Also, PI Demmel recently served on the IEEE 754 Floating Point Standard Committee and proposed a new floating point instruction (AugmentedAddition) to accelerate this algorithm [113], which was adopted in the standard.

2.9.4 New precisions (lower, higher, and mixed) and mixed types (real and complex). In previous work, we highlighted the advantages of algorithms that use mixed precision to save time and energy by using 32-bit arithmetic for most operations and 64-bit for the few operations that required the 64-bit accuracy [114–119]. Much remains to do in this area (e.g., extending this approach to more recent architectures and other algorithms). There are also new mixed-precision algorithms, where higher precision can be applied to a small but critical segment of a calculation to improve the numerical stability of the entire algorithm [120, 121].

Motivated by "big data," vendors have already begun to offer low-precision (16-bit) versions of BLAS. On the other hand, in our survey, some users have requested quad (128-bit) or arbitrary precisions of our libraries. Since the number of possible input, output, and internal precisions and types can lead to a combinatorial explosion in software that needs to be written, tested, and performance tuned, we have already begun collaborating with Intel (at their request) on refining the Extended Precision BLAS (XBLAS) standard developed by the previous BLAS standard effort [122]. Again, based on interactions with the user community, we will prioritize which operations need to be done at which precisions (starting with the BLAS).

2.9.5 Target architectures and frameworks. As industry, academia, and the computational science community in general prepare for exascale systems, we will target the prevailing hardware platforms and their future versions, which promise new and unprecedented levels of performance that will unfortunately be accompanied by new programmability issues stemming from the increased parallelism and multi-way heterogeneity. We plan support for modern computing hardware, including HPC CPUs, self-hosted accelerators, and off-load accelerators, which implies either unified or off-chip memory. Recent processors include the IBM POWER9, AMD Zen, and ARM v8a that further increase the complexity of what is regarded as a general-purpose processor. Advanced vectorization necessary for nearing peak performance, complicated cache hierarchy with a range of sharing/inclusion/eviction rules at different levels and cores, hardware multithreading, and functional unit sharing are among the many challenges faced by numerical linear algebra libraries—especially the ones dealing with dense matrices, because they are expected to deliver performance close to the machine's peak. Offloading the software burden to Level-3 BLAS proves challenging, as new algorithms require new computational kernels that require time for vendor optimization. Level-1 and Level-2 BLAS rely on high memory bandwidth, and the hardware complicates the programming with new technologies such as 3-D stacked memory, including High Bandwidth Memory (HBM) developed by AMD and Hynix and the Hybrid Memory Cube (HMC) developed by Micron. GPU vendors NVIDIA and AMD use HBM. CPU vendors IBM and Intel have lined up behind HMC, while Intel is developing its own variant of HMC, Multi-Channel DRAM (MCDRAM), designed specifically for Intel processors. We will need to combine these advances with already available NUMA memory structures. Finally, we will support the interconnects; including the system bus standards like PCI Express and NVLink for connecting CPUs and GPUs and multiple GPUs, as well as distributed-systems interconnects like Infiniband and Omni-Path; and we will work around their relative bandwidth deficiencies compared to on-node performance (over 10 TFLOP/s per single GPU with only 80 GB/s NVLink connection and 200 Gbps of NIC bandwidth).

2.9.6 Target cloud and container platforms. Spark is a widely used platform for in-memory analysis of large data sets [123]. Many computational scientists have been interested in using Spark to analyze their scientific data, whether it is collected from sensors or it is output from large simulations. In collaboration with Cray and Berkeley Lab, we compared the performance of HPC implementations (including ScaLAPACK) of various common linear algebra operations used for data analysis with native versions in Spark, and we measured up to a 20× speedup versus native Spark [124]. We are continuing work on an interface called Alchemist [125, 126], which enables Spark users call out to high-performance MPI library implementations,

including ScaLAPACK and many others, and—depending on user requirements—extends the interface to enable using our other linear algebra libraries. Alchemist will serve as the natural deployment platform to interface with cloud computing systems. The interface is general, and future development plans include extending it to enable ipython notebook users to call out to high-performance MPI library implementations directly from ipython. This is a new context in which our software will be used, quite different, in fact, from the ecosystem of established HPC software like MPI and OpenMP. Cloud frameworks like Hadoop and (more recently) Spark present a challenge for HPC libraries mainly due to different provisioning scenarios. We first plan to provide suitable wrappers to bridge the software interface gap. Recent proof-of-principle work in this direction was performed by PI Mahoney [127], in which newly developed interfaces for Python, Dask, and PySpark that enable the use of Alchemist with additional data analysis frameworks are described. For example, this work described the combination of Alchemist with RLlib, an increasingly popular library for reinforcement learning, and it considered the benefits of leveraging HPC simulations in reinforcement learning, another very non-traditional application area. In addition to this, we plan specific optimizations and tuning that would allow our libraries to take better advantage of the hardware—whatever the cloud scheduler chooses. With BALLISTIC supporting multiple architectures, users will be able move their applications seamlessly between various computer systems.

2.9.7 Implement user-controllable performance tuning for BALLISTIC components. All dense linear algebra packages need some level of tuning to achieve good performance, and currently none possesses any autotuning capabilities. This applies equally to both the legacy packages and the newer packages. The prime examples are: (1) the algorithmic blocking factor N_B in LAPACK and ScaLAPACK and (2) the shape and order of the process grid $P \times Q$ in ScaLAPACK. Although the performance of those packages depends critically on those settings, no facilities are provided to the user for making the right choice.

The situation is identical in the newer packages. MAGMA, for the most part, inherits the algorithmic blocking parameter N_B from LAPACK. PLASMA, on the other hand, uses N_B to describe 2-D tiling of the matrix while introducing another parameter, *internal blocking* I_B, to describe the algorithmic blocking within tiles and also exposing other tuning parameters (e.g., different shapes of reduction trees in tall-and-skinny matrix factorizations). At the same time, what both libraries do is use default settings and enable the user to change them to his or her liking. For BALLISTIC, we will develop configuration and autotuning facilities.

Configuration facilities enable the user to provide values for tuning parameters in a configuration file, pointed to by an environment variable. The user will be able to specify values of tunable parameters: per routine and per precision in a multidimensional space. The file will be in a human-readable but also parse-able format like JSON, TOML, YAML, or XML, depending on the integration platform.

2.9.8 Improve BALLISTIC interfaces and usability. Our periodic user surveys, conducted regularly over the history of this collaborative effort, have generated a long list of possible improvements that, after further prioritization, need to be made to fully satisfy our large user community. Below, we list the most notable items and propose ways to address them.

Automate data layout in ScaLAPACK (and other libraries): We propose improved ease of use by providing a wrapper around ScaLAPACK that would take input matrices stored in simpler, user-friendlier formats (e.g., 1-D blocked), automatically convert them to the more complicated 2-D block cyclic layouts needed for high performance, and then convert the output back, if requested.

More general matrix layouts, with details hidden or exposed: Some users requested that matrices be represented more simply, while others requested more general (and complicated) layouts. To satisfy these contradictory demands requires building several wrappers, each of which exposes the desired level of detail.

32 bit vs. 64 bit: Several users pointed out that there are inter-language incompatibilities between the data types used for integers, and also that 32-bit integers were often too small to hold the necessary values without overflow (e.g., workspace size). Based on a careful survey analysis, current linking practice, and ABI standards, the appropriate version will be delivered for the target platforms.

Automatic workspace allocation: This was requested by many users, and specifically rejected by others, since their applications required more complete and careful control over system resources (e.g., embedded systems). This speaks to having multiple wrappers that provide different functionality levels.

Optional arguments for requesting more information: We propose wrappers to enable the users who want more information to have it. We could use arguments to retrieve error bounds and other numerical quantities, or use flags to indicate internal decisions made by the code or how much work space was used.

Access via other languages and frameworks: Several users requested C++ wrappers to, for example, have one way to call our code for all data types. We can leverage and extend the BLAS++ and LAPACK++ wrappers that SLATE provides [128] for this purpose. Another example is Spark, as mentioned earlier.

Better exception handling: Exception handling (dealing with inputs containing Inf's and NaN's or runtime FP exceptions) was important/very important for 67% of survey responders. We carefully formulated the semantics of *correct* exception handling.[1] The reference BLAS implementations do not satisfy this criterion, and, consequently, we plan to produce a *reference implementation* suitable for debugging that propagates exceptions correctly, and a *tuned implementation* that at least guarantees termination.

2.9.9 Add LAPACK functionality not currently in our other libraries to these libraries. Of the $1,755$ functions in LAPACK (including all precisions), only 696 are also in ScaLAPACK, 546 in PLASMA (including statically as well as dynamically scheduled versions), 716 in MAGMA (including versions with CPU and GPU interfaces), and 238 in SLATE. Efforts are ongoing to add the missing functions deemed most important to these other libraries. This often means changing the algorithm or choosing one of several possibilities to best match the target architecture. Based on the user survey [5], the current wish list (top priority) consists of functions for (1) nonsymmetric eigenproblems (GEEV) in ScaLAPACK and PLASMA; (2) QR with column pivoting in ScaLAPACK and PLASMA using Level-3 BLAS, as in LAPACK's GEQP3; (3) band matrix factorizations and solvers in PLASMA and MAGMA (needed in 25% of the survey applications); (4) SVD in PLASMA (SVD is reportedly used in 42% of the survey applications); and (5) $\mathrm{LDL}^{\mathrm{T}}$ factorizations and solvers in ScaLAPACK and PLASMA.

2.9.10 Improving the existing software base. For historic reasons, the existing software base for Sca/LAPACK is written in Fortran and uses archaic software engineering techniques. Most scientific software written today is not based in Fortran, and a recent study of programming languages ranks Fortran 38^{th}, with the top-3 programming languages listed as Python, Java, and C [129]. As part of the BALLISTIC effort, we plan to develop a new prototype library in C. We will involve the community at large (open source, industry, users, contributors) in the design of the interfaces and the library as a whole.

3 Deliverables

Software development in BALLISTIC will be focused on quality, robustness, efficiency, portability, maintainability, and open-source collaborations. As described below, we will follow successful techniques proven in the context of other software packages developed by our respective groups.

Abide by coding standards: We will follow language standards for all future development and will strive to make all existing codes standards compliant. Specifically, all C/C++ codes will be compliant with C11 (with C18 corrections) and C++11 standards and their contemporary replacements, and all Fortran interfaces will be compliant with Fortran 2003's ISO C binding which will serve the newer Fortran compilers.

We will follow parallel programming standards, specifically, MPI for message passing and OpenMP for multithreading. For the development of GPU-accelerated codes, we will put emphasis on the use of OpenMP and OpenACC standards and only resort to NVIDIA CUDA where absolutely necessary.

We will follow a set of coding guidelines based on best industry practices [130] (e.g., Google C/C++ Style Guide, open-source codes by Intel and NVIDIA). The codes will be self documenting through extensive use of

[1] Briefly, the code always terminates, and any exception that causes an erroneous answer must somehow propagate to the output, either by having an Inf or NaN appear in the output and/or via an error flag.

the Doxygen system, which allows for automatic generation of interactive, browseable, and searchable online documentation. For auxiliary documentation, we will utilize Markdown to create Wikis and automatically produce HTML and LaTeX/PDF documentation.

We will only rely on established and standardized numerical libraries as components. Specifically, we will make it a priority to maximize the use of BLAS. At the same time, we will participate in the creation of new standards (e.g., Batched BLAS) and adopt them in our software as they mature.

Continue regular testing and continuous integration: The extremely wide adoption of our software over many years is testimony to its reliability and the trust of the community. This takes constant effort on our part, with a capability for users to upload bug reports (and track our responses), including very difficult and rare examples of eigenvalue or singular value problems, where a user encounters a convergence failure, floating point exception, or similar problem. We have maintained a cumulative list of all these "torture test" cases over many years and use them to help validate new releases. Our philosophy is also to provide optional condition numbers and/or error bounds for any problem we solve (linear systems, least squares, eigenproblems, etc). For details, see the chapter on Accuracy and Stability in the *LAPACK Users' Guide* [131, ch. 4]. Another trustworthiness issue is better floating exception handling, discussed further in Section 2.9.8

The BALLISTIC team will implement a software development process based on instant delivery through a public Bitbucket repository, where downloading or cloning the current state of the repository is the intended method of delivery. This will allow for fast responses to bug reports and feature requests, which are assisted by the online issue tracking system. Major releases will be marked bi-annually, to coincide with the ISC conference, held in Germany in June/July, and the SC conference, held in the United States in November.

Going forward, we will apply continuous integration (i.e., frequent [daily] commits), which is required to pass the build process and unit testing in the developer's local environment before committing to the main line. This will be facilitated through a testing harness, which allows for stress testing each computational routine. Every computational routine will be accompanied by one or more unit tester(s) and accept a range of input configurations (different matrix/vector sizes and properties). Our model for all testing suites for BALLISTIC libraries will be the strenuous testing suites of LAPACK and ScaLAPACK.

3.1 Compatibility with Other Popular Software Packages

Eigen [132], now in version 3, is a linear algebra library that originated in expression templates' implementation of BLAS routines. It has since expanded its scope include both dense and sparse functionality. The dense matrix factorization, decompositions, and solvers include common one-sided (Cholesky, LU, QR, LDLT) and two-sided algorithms (eigenvalues and singular values) with orthogonality utilities for Givens, Householder, and Jacobi rotations. The CPU parallelism is implemented through a thread-pool abstraction. For computational performance by means of accelerator offload, Eigen includes routines that are capable of using either NVIDIA's CUDA or OpenCL's SYCL backends. The latter implemented by triSYCL. In that sense, the execution scopes targeted by Eigen are more limited in comparison with that of BLAS, Sca/LAPACK, PLASMA, MAGMA, and SLATE libraries that cumulatively cover multicore CPUs, multi-GPUs, and distributed memory of hardware accelerated nodes.

In addition to the standard BLAS and LAPACK data types (32-bit and 64-bit floating point, real and complex variants), Eigen also includes newer formats such as FP16 (16-bit floating point with limited exponent range) and BFloat16 (16-bit floating point with extended exponent range). The support of all floating-point data types across all the available algorithms is not uniform but grows over time and often depends on the support available in the software packages and libraries that Eigen interfaces. Also, the basic vector and matrix data types are now accompanied by a tensor type with contraction and convolution operations implemented directly or with FFT and its inverse.

The sparse functionality in Eigen spans both direct as well as iterative Krylov subspace methods, matrix reordering, and preconditioning. The former class of direct solvers uses SuiteSparse for general sparsity patterns and KLU for special-structure sparsity patterns. The latter class includes CG, BiCGSTAB, and

least-squares CG.

Owing to a custom typing and naming, it would be impractical to insist on compatibility with the Eigen convention. As a trivial example, consider dense Cholesky factorization test in test/cholesky.cpp file:

```
LLT<SquareMatrixType,Lower> chollo(symmLo);
LLT<SquareMatrixType,Upper> cholup(symmUp);
```

The first line is for a lower-triangular Cholesky factorization LL^T, and hence the LLT datatype is intuitive for this purpose. On the other hand, the second line is for upper-triangular Cholesky factorization U^TU, and the name LLT could be counter-intuitive for some users. Using a trivial type alias called UTU could be helpful here. In ether case, these naming conventions are far removed from LAPACK's xPOTRF() or MATLAB's chol or even Trilinos's Teuchos syntax. Reconciling these issues might be excessively burdensome in practice.

For brevity, we did not differentiate supported and unsupported components of Eigen, assuming that both could be potentially beneficial to some users to an equal extent. For example, the tensor data type and operations are in the unsupported directory and might not share the code quality and testing coverage with the established Eigen components.

3.2 Evaluation Plan

We will evaluate the success of BALLISTIC using the following metrics: **(1)** Growth of the contributor base: to facilitate community participation and allow us to record and track the relevant activity (e.g, number of branches, number of pull requests, and contributors) that measures the community's involvement beyond the developers' groups, we have already transfered all BALLISTIC components to GitHub or BitBucket. **(2)** Number of sotware releases: we plan two major releases each year, along with point releases of component libraries, with additions and improvements, and we will closely monitor standard metrics (e.g., new lines of code produced, test coverage, cyclomatic complexity). **(3)** Number of downloads and citations: we will continue to monitor the number of downloads and citations, as they provide a rough lower bound of our number of users. **(4)** Level of users' satisfaction: we will continue to survey our users [5] to solicit their comments and criticisms and determine how happy they are with the software.

3.3 Sustained and Sustainable Impacts

Sustain commitments to community engagement: An important aspect of BALLISTIC sustainability is our interaction with the community and our openness to feedback and contributions in particular. To build and maintain the community needed for long-term self sustainability, we maintain active users' forums for all of our software efforts. For instance, the ScaLAPACK Users' Forum [133] provides an online portal for posting messages and answering questions related to Sca/LAPACK. The forum has different branches dealing with installation, linking, data format, algorithms, and performance. For the Sca/LAPACK Users' Forum, there are over 7,000 members and 1,600 topics. These mechanisms will help facilitate the incorporation of Sca/LAPACK into our interdisciplinary partners' applications and also garner feedback.

LAPACK already has a good track record of external contributions, and a significant portion of our software improvements come from contributions from faculty in various universities, researchers in labs, industry partners, and volunteers. One goal of BALLISTIC is to further develop and nourish a community of developers. The software is critical, and so we believe that it would be irresponsible to let it loose. However, the goal is to delegate as much as possible, and we will always need to support our contributors and moderate their contributions. Community growth is vital to our sustainability.

The trends we have seen recently include the following. (1) We are seeing more and more non-numerical people contributing. After we moved the software into a GitHub repository, people have contributed with Git and Travis CI [134], which is a rather new method of contribution. (2) We are seeing more and more industry interest and contributions, and we are in regular contact with Intel and MathWorks. Recently, Intel collaborated with us on LAPACKE, which is a C interface for LAPACK. MathWorks and Oracle have also

contributed several patches. It is also worth noting that these new contributors are in addition to our existing contributors from academia.

Use permissive software license: The goal of BALLISTIC, as with *all* of our previous work, is to (1) enable anyone (individual, industry, government) to contribute to the software growth and to (2) enable anyone to integrate the software, without hesitation, into their individual projects. For these reasons, we will continue to use the BSD 3-Clause License [135] for all of the software released under this project, since it has served this purpose well for all of our previous work over many years.

4 Potential Project Impacts

4.1 Contribution of BALLISTIC to National Research Infrastructure

The Sca/LAPACK libraries are the community standard for dense linear algebra, which is foundational for many computational science and engineering domains domains. They have been adopted and/or supported by a large community of users, computing centers, and HPC vendors, including: the National Center for Supercomputing Applications, the San Diego Supercomputer Center, the Texas Advanced Computing Center, the Pittsburgh Supercomputing Center, Intel, AMD, Cray, IBM, HP, Fujitsu, NEC, Numerical Algorithms Group, and MathWorks. Learning to use them, either natively or as part of a vendor package, is a basic component of the education of many, if not most computational scientists and engineers. No other numerical library can claim such breadth of integration with the community. Consequently, enhancing these libraries with state-of-the-art methods and algorithms and adapting them for new and emerging platforms (reaching up to extreme scale), should be expected to have a correspondingly broad and significant impact on the research and education community, government laboratories, and private industry.

BALLISTIC will offer benefits for computational science education by offering an extreme scale–ready set of components—for application developers by giving them a single point of contact for registering their requirements, and for vendors by organizing a community to help them assemble the complete software environment that their systems need for success. Our strong record of achievement in delivering robust, high-quality, high-performance software—and in leading previous community efforts of this kind—argue strongly for the prospects of widespread and positive effects of this project. Moreover, having helped lead community-developed standards efforts in numerical linear algebra software for decades, the developers are also well positioned to re-energize that standardization process to address the new functionalities (e.g., Batched BLAS) that many new applications require, and BALLISTIC's community engagement will provide opportunities to organize workshops to bring together leaders from the different stakeholder communities to achieve agreement on such standards.

4.2 Community Outreach and Education

At our home institutions, this project will have educational impact by involving undergraduate and graduate students. This situation presents excellent opportunities for interaction with postdoctoral and professional research staff, as well as with colleagues in academic, government, and industry research labs. Furthermore, the developers have taught and will continue to teach special topics courses and workshops in their areas of research. Outside our home institutions, we plan to contribute to relevant tutorials at conferences and industry workshops. We have previously taught performance and architecture related tutorials at various international conferences and user group meetings, and we expect performance engineering for emerging hybrid and multi/many-core architectures to be in great demand as a tutorial topic in the future. In addition, the developers' experience and association with various hardware and software should ensure rapid uptake of the freely available software that will result from BALLISTIC.

References

[1] Jack Dongarra, Pete Beckman, Terry Moore, Patrick Aerts, Giovanni Aloisio, Jean-Claude Andre, David Barkai, Jean-Yves Berthou, Taisuke Boku, Bertrand Braunschweig, Franck Cappello, Barbara Chapman, Xuebin Chi, Alok Choudhary, Sudip Dosanjh, Thom Dunning, Sandro Fiore, Al Geist, Bill Gropp, Robert Harrison, Mark Hereld, Michael Heroux, Adolfy Hoisie, Koh Hotta, Yutaka Ishikawa, Zhong Jin, Fred Johnson, Sanjay Kale, Richard Kenway, David Keyes, Bill Kramer, Jesus Labarta, Alain Lichnewsky, Thomas Lippert, Bob Lucas, Barney Maccabe, Satoshi Matsuoka, Paul Messina, Peter Michielse, Bernd Mohr, Matthias Mueller, Wolfgang Nagel, Hiroshi Nakashima, Michael E. Papka, Dan Reed, Mitsuhisa Sato, Ed Seidel, John Shalf, David Skinner, Marc Snir, Thomas Sterling, Rick Stevens, Fred Streitz, Bob Sugar, Shinji Sumimoto, William Tang, John Taylor, Rajeev Thakur, Anne Trefethen, Mateo Valero, Aad van der Steen, Jeffrey Vetter, Peg Williams, Robert Wisniewski, and Kathy Yelick. The international exascale software project roadmap. *International Journal of High Performance Computer Applications*, 25, 2011. Available: http://www.exascale.org/mediawiki/images/2/20/IESP-roadmap.pdf.

[2] Rick Stevens (chair). Scientific grand challenges: Architecture and technology for extreme scale computing. Technical report, Dept. of Energy Office of Advanced Scientific Computing Research and Office of Advanced Simulation and Computing, December 2009. Available: http://www.er.doe.gov/ascr/ProgramDocuments/Docs/Arch-TechGrandChallengesReport.pdf.

[3] J. Tinsley Oden, Omar Ghattas, and John Leslie King (chairs). A report of the National Science Foundation advisory committee for cyberinfrastructure task force on grand challenges. Technical report, National Science Foundation Advisory Committee for Cyberinfrastructure Task Force on Grand Challenges, March 2011. Available: http://www.nsf.gov/od/oci/taskforces/TaskForceReportGrandChallenges.pdf.

[4] Samuel H. Fuller and Editors. The future of computing performance: Game over or next level? *Committee on Sustaining Growth in Computing Performance; National Research Council Lynette I. Millett.*, 2011.

[5] Jack Dongarra, Jim Demmel, Julien Langou, and Julie Langou. 2016 dense linear algebra software packages survey. Technical Report UT-EECS-16-744 September 2016, University of Tennessee, September 2016. LAPACK Working Note 290 http://www.netlib.org/lapack/lawnspdf/lawn290.pdf.

[6] PLASMA website. http://icl.utk.edu/plasma/.

[7] MAGMA website. http://icl.utk.edu/magma/.

[8] SLATE website. http://icl.utk.edu/slate/.

[9] C. Sanderson and R. Curtin. Armadillo: a template-based C++ library for linear algebra. *Journal of Open Source Software*, 1, 2016.

[10] J. Treibig K. Iglberger, G. Hager and U. Rüde. Expression templates revisited: A performance analysis of current methodologies. *SISC*, 34:42–69, 2012.

[11] NumPy. Available: http://www.numpy.org.

[12] The R project for statistical computing. Available: https://www.r-project.org.

[13] Robert D. Falgout, Jim E. Jones, and Ulrike Meier Yang. Pursuing scalability for Hypre's conceptual interfaces. *ACMTOMS*, 31:326–350, 2005.

[14] P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. A fully asynchronous multifrontal solver using distributed dynamic scheduling. *SIMAX*, 23(1):15–41, 2001.

[15] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, Hong Zhang, and Hong Zhang. PETSc web page. http://www.mcs.anl.gov/petsc, 2016.

[16] Xiaoye S. Li. An overview of SuperLU: Algorithms, implementation, and user interface. *ACM Trans. Math. Software*, 31(3):302–325, September 2005.

[17] Michael A. Heroux, Roscoe A. Bartlett, Vicki E. Howle, Robert J. Hoekstra, Jonathan J. Hu, Tamara G. Kolda, Richard B. Lehoucq, Kevin R. Long, Roger P. Pawlowski, Eric T. Phipps, Andrew G. Salinger, Heidi K. Thornquist, Ray S. Tuminaro, James M. Willenbring, Alan Williams, and Kendall S. Stanley. An overview of the Trilinos project. *ACMTOMS*, 31(3):397–423, 2005.

[18] ABINIT. Available: http://www.abinit.org.

[19] Quantum Espresso. Available: http://www.quantum-espresso.org.

[20] VASP. Available: https://www.vasp.at.

[21] deal.II – an open source finite element library. Available: http://www.dealii.org.

[22] OpenSees. Available: http://opensees.berkeley.edu.

[23] Mark Gates, Azzam Haidar, and Jack Dongarra. Accelerating computation of eigenvectors in the dense nonsymmetric eigenvalue problem. In Michel Daydé, Osni Marques, and Kengo Nakajima, editors, *High Performance Computing for Computational Science – VECPAR 2014: 11th International Conference, Eugene, OR, USA, June 30 – July 3, 2014, Revised Selected Papers*, pages 182–191, Cham, 2015. Springer International Publishing.

[24] B. Kågström, D. Kressner, E.S. Quintana-Ortí, and G. Quintana-Ortí. Blocked algorithms for the reduction to Hessenberg-triangular form revisited. *BIT Numerical Mathematics*, 48(3):563–584, 2008.

[25] Zlatko Drmač and Krešimir Veselić. New fast and accurate Jacobi SVD algorithm. II. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1343–1362, 2008.

[26] Zlatko Drmač and Krešimir Veselić. New fast and accurate Jacobi SVD algorithm. I. *SIAM Journal on Matrix Analysis and Applications*, 29(4):1322–1342, 2008.

[27] C. Ashcraft, R. G. Grimes, and J. G. Lewis. Accurate symmetric indefinite linear equation solvers. *SIAM J. Matrix Analysis and Applications*, 20(2):513–561, 1998.

[28] E. Elmroth and F. G. Gustavson. Applying recursion to serial and parallel QR factorization leads to better performance. *IBM J. Res. Dev.*, 44(4):605–624, July 2000.

[29] Brian Sutton. Computing the complete CS decomposition. *Numer. Algorithms*, 50(1):33–65, 2009.

[30] Jed A. Duersch and Ming Gu. True BLAS-3 performance QRCP using random sampling. Technical Report 1509.06820, arXiv, 2015.

[31] Jianwei Xiao, Ming Gu, and Julien Langou. Fast parallel randomized QR with column pivoting algorithms for reliable low-rank matrix approximations. In *24th IEEE International Conference on High Performance Computing, Data, and Analytics (HIPC), Jaipur, India*, 2017.

[32] P. Drineas, M. W. Mahoney, and S. Muthukrishnan. Relative-error CUR matrix decompositions. *SIAM Journal on Matrix Analysis and Applications*, 30:844–881, 2008.

[33] M. W. Mahoney and P. Drineas. CUR matrix decompositions for improved data analysis. *Proc. Natl. Acad. Sci. USA*, 106:697–702, 2009.

[34] P. Drineas and M. W. Mahoney. RandNLA: Randomized Numerical Linear Algebra. *Communications of the ACM*, 59(6):80–90, 2016.

[35] James Demmel, Laura Grigori, and Sebastien Cayrols. Low rank approximation of a sparse matrix based on LU factorization with column and row tournament pivoting. *SIAM J. Sci. Comp.*, 40(2):C181–C209, March 2018.

[36] M. W. Mahoney. *Randomized algorithms for matrices and data*. Foundations and Trends in Machine Learning. NOW Publishers, Boston, 2011.

[37] H. Avron, P. Maymounkov, and S. Toledo. Blendenpik: Supercharging LAPACK's least-squares solver. *SIAM Journal on Scientific Computing*, 32:1217–1236, 2010.

[38] X. Meng, M. A. Saunders, and M. W. Mahoney. LSRN: A parallel iterative solver for strongly over- or under-determined systems. *SIAM Journal on Scientific Computing*, 36(2):C95–C118, 2014.

[39] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Minimizing communication in numerical linear algebra. *SIAM Journal on Matrix Analysis and Applications*, 32(3):866–901, 2011.

[40] J. W. Hong and H. T. Kung. I/O complexity: The red-blue pebble game. In *STOC'81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 326–333, 1981.

[41] D. Irony, S. Toledo, and A. Tiskin. Communication lower bounds for distributed-memory matrix multiplication. *J. Parallel Distrib. Comput.*, 64(9):1017–1026, 2004.

[42] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2011.

[43] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Graph expansion and communication costs of fast matrix multiplication. *Journal of the ACM*, 59(6), Dec 2012.

[44] G. Ballard, J Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Graph expansion analysis for communication costs of fast rectangular matrix multiplication. In *1st Mediterranean Conference on Algorithms*, Dec 2012. UC Berkeley EECS Tech Report UCB/EECS-2012-194.

[45] L. Grigori, J. Demmel, , and H. Xiang. Communication avoiding Gaussian elimination. In *Proceedings of the IEEE/ACM SuperComputing SC08 Conference*, November 2008.

[46] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication-optimal parallel and sequential Cholesky decomposition. *SIAM J. Sci. Comp.*, 32(6):3495–3523, 2010. conference version appeared in SPAA'09.

[47] J. Demmel, L. Grigori, M. Hoemmen, and J. Langou. Communication-optimal parallel and sequential QR and LU factorizations. *SIAM J. Sci. Comp.*, 34(1), Feb 2012.

[48] E. Solomonik and J. Demmel. Communication-optimal parallel 2.5D matrix multiplication and LU factorization algorithms. In *Euro-Par 2011 Parallel Processing*, volume 6853 of *Lecture Notes in Computer Science*, pages 90–109. Springer, 2011.

[49] E. Solomonik, A. Bhatele, and J. Demmel. Improving communication performance in dense linear algebra via topology aware collectives. In *Supercomputing 2011*. IEEE Computer Society, 2011.

[50] G. Ballard, J. Demmel, B. Lipshitz, and O. Schwartz. Communication-avoiding parallel Strassen: Implementation and performance. In *Supercomputing 2012*. IEEE Computer Society, 2012.

[51] Grey Ballard, Aydin Buluc, James Demmel, Laura Grigori, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. Communication optimal parallel multiplication of sparse random matrices. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13)*, July 2013. (also UC Berkeley Tech Report UCB/EECS-2013-13).

[52] Grey Ballard, James Demmel, Benjamin Lipshitz, Oded Schwartz, and Sivan Toledo. Communication efficient Gaussian elimination with partial pivoting using a shape morphing data layout. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'13)*, July 2013. (also UC Berkeley Tech Report UCB/EECS-2013-12).

[53] James Demmel, David Eliahu, Armando Fox, Shoaib Ashraf Kamil, Benjamin Lipshitz, Oded Schwartz, and Omer Spillinger. Communication-optimal parallel recursive rectangular matrix multiplication. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*. IEEE Computer Society, 2013. UC Berkeley EECS Tech Report UCB/EECS-2012-205.

[54] G. Ballard, J. Demmel, O. Holtz, B. Lipshitz, and O. Schwartz. Communication-optimal parallel algorithm for Strassen's matrix multiplication. In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2012.

[55] Grey Ballard, James Demmel, Olga Holtz, and Oded Schwartz. Sequential communication bounds for fast linear algebra. Technical Report UCB/EECS-2012-36, EECS Department, University of California, Berkeley, Mar 2012.

[56] Amal Khabou, James Demmel, Laura Grigori, and Ming Gu. LU factorization with panel rank revealing pivoting and its communication avoiding version. *SIAM J. Mat. Anal. Appl.*, 34(3), 2013. (also UC Berkeley Tech Report UCB/EECS-2012-15).

[57] Grey Ballard, James Demmel, and Ioana Dumitriu. Minimizing communication for eigenproblems and the singular value decomposition. Technical Report UCB/EECS-2011-14, EECS Department, University of California, Berkeley, Feb 2011.

[58] G. Ballard, J. Demmel, and A. Gearhard. Communication bounds for heterogeneous architectures (brief announcement). In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*, pages 257–258. ACM, 2011. UC Berkeley EECS Tech Report UCB/EECS-2011-13.

[59] M. Anderson, G. Ballard, J. Demmel, and K. Keutzer. Communication-avoiding QR decomposition for GPUs. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*, pages 48–58. IEEE Computer Society, 2011. UC Berkeley EECS Tech Report UCB/EECS-2010-131.

[60] L. Grigori, J. Demmel, and H. Xiang. CALU: A communication-optimal LU factorization algorithm. *SIAM J. Matrix Anal. Appl.*, 32(4):1317–1350, 2011. UC Berkeley EECS Tech Report UCB/EECS-2010-29.

[61] G. Ballard, J Demmel, and N. Knight. Avoiding communication in successive band reduction. *ACM Trans. Parallel Computing*, 1(2), Jan 2015.

[62] D. Becker, G. Ballard, J. Demmel, J. Dongarra, A. Druinsky, I. Peled, O. Schwartz, S. Toledo, and Yamazaki. Implementing a blocked Aasen's algorithm with a dynamic sscheduler on multicore architectures. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*. IEEE Computer Society, 2013.

[63] J. Demmel, L. Grigori, M. Gu, and H. Xiang. Communication-avoiding rank-revealing QR factorization with column pivoting. *SIAM J. Mat. Anal. Appl.*, 36(1), 2015. (also UC Berkeley Tech Report UCB/EECS-2013-46).

[64] G. Ballard, J. Demmel, O. Holtz, and O. Schwartz. Communication costs of Strassen's matrix multiplication. *Comm. ACM*, 57(2), Feb 2014.

[65] E. Solomonik, E. Carson, N. Knight, and J. Demmel. Tradeoffs between synchronization, communication, and work in parallel linear algebra computations. In *26th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'14)*, 2014.

[66] G. Ballard, J. Demmel, L. Grigori, M. Jacquelin, N. Knight, and H. D. Nguyen. Reconstructing Householder vectors from Tall-Skinny QR. *J. Parallel and Distributed Computing*, 85, 2015.

[67] A. Benson, J. Demmel, and D. Gleich. Direct QR factorizations for tall-and-skinny matrices in MapReduce architectures. In *2013 IEEE Intern. Conf. on Big Data (IEEE BigData 2013)*, Oct 2013.

[68] J. Demmel and H. D. Nguyen. Reproducible Tall-Skinny QR factorization. In *22nd IEEE Symp. Computer Arithmetic (ARITH'22)*, 2015.

[69] E. Carson, J. Demmel, L. Grigori, N. Knight, P. Koanantakool, O. Schwartz, and H. V. Simhadri. Write-avoiding algorithms. In *Proceedings of the 28th IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*, 2016. (also UC Berkeley Tech Report UCB/EECS-2015-163).

[70] P.-Y. David, J. Demmel, L. Grigori, and S. Peyronnet. Lower bounds on communication for sparse Cholesky factorization of a model problem (brief announcement). In *ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. ACM, 2010.

[71] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in computing Krylov subspaces. Technical Report UCB/EECS-2007-123, EECS Department, University of California, Berkeley, Oct 2007.

[72] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Avoiding communication in sparse matrix computations. In *IEEE Intern. Parallel and Distributed Processing Symposium (IPDPS'08)*. IEEE, 2008. long version appeared as UC Berkeley EECS Technical Report UCB/EECS-2007-123.

[73] J. Demmel, M. Hoemmen, M. Mohiyuddin, and K. Yelick. Minimizing communication in sparse matrix solvers. In *Proceedings of Supercomputing '09*. Sponsored by ACM SIGARCH and IEEE Computer Society, Nov 2009.

[74] M. Hoemmen. *Communication-Avoiding Krylov Subspace Methods*. PhD thesis, University of California, Berkeley, California, May 2010.

[75] Erin Carson and James Demmel. A residual replacement strategy for improving the maximum attainable accuracy of s-step Krylov subspace methods. *SIAM J. Mat. Anal. Appl.*, 35(1), 2014. (also UC Berkeley Tech Report UCB/EECS-2012-197).

[76] Erin Carson, Nicholas Knight, and James Demmel. Avoiding communication in nonsymmetric Lanczos-based Krylov subspace methods. *SIAM J. Sci. Comp.*, 35(5), Oct 2013. (also UC Berkeley Tech Report UCB/EECS-2011-93).

[77] A. Buluc, S. Williams, L. Oliker, and J. Demmel. Reduced-bandwidth multithreaded algorithms for sparse matrix-vector multiplication. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'11)*. IEEE Computer Society, 2011.

[78] E. Carson, J. Demmel, and N. Knight. An efficient deflation technique for the communication-avoiding Conjugate Gradient method. *Electronic Transactions on Numerical Analysis (ETNA)*, 23, 2014.

[79] E. Carson and J. Demmel. Accuracy of the s-step Lanczos method for the symmetric eigenproblem. *SIAM, J. Mat. Anal. Appl.*, 36(2), 2015.

[80] Edgar Solomonik, Devin Matthews, Jeff Hammond, and James Demmel. Cyclops tensor framework: reducing communication and eliminating load imbalance in massively parallel contractions. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*. IEEE Computer Society, 2013. UC Berkeley EECS Tech Report UCB/EECS-2013-11.

[81] E. Solomonik, D. Matthews, J. Hammond, J. Stanton, and J. Demmel. A massively parallel tensor contraction framework for Coupled Cluster computations. *J. Parallel and Distributed Computing*, 74(2), June 2014.

[82] A. Abdelfattah, M. Baboulin, V. Dobrev, J. Dongarra, C. Earl, J. Falcou, A. Haidar, I. Karlin, Tz. Kolev, I. Masliah, and S. Tomov. High-performance tensor contractions for GPUs. *Procedia Computer Science*, 80:108 – 118, 2016. International Conference on Computational Science 2016, {ICCS} 2016, 6-8 June 2016, San Diego, California, {USA}.

[83] Edgar Solomonik, Aydin Buluc, and James Demmel. Minimizing communication in all-pairs shortest paths. In *Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium (IPDPS'13)*. IEEE Computer Society, 2013. UC Berkeley EECS Tech Report UCB/EECS-2013-10.

[84] V. Volkov and J. Demmel. Benchmarking GPUs to tune dense linear algebra. In *Supercomputing 08*. IEEE, 2008.

[85] M. Baboulin, S. Donfack, J. Dongarra, L. Grigori, A. Rémy, and S. Tomov. A class of communication-avoiding algorithms for solving general dense linear systems on CPU/GPU parallel machines. *Procedia Computer Science*, 9:17–26, 2012.

[86] A. Haidar, S. Tomov, J. Dongarra, R. Solca, and T. Schulthess. A novel hybrid CPU-GPU generalized eigensolver for electronic structure calculations based on fine grained memory aware tasks. *International Journal of High Performance Computing Applications*, 28(2):196–209, May 2014.

[87] Azzam Haidar, Mark Gates, Stanimire Tomov, and Jack Dongarra. Toward a scalable multi-GPU eigensolver via compute-intensive kernels and efficient communication. In *ICS'13: 27th International Conference on Supercomputing (submitted)*, Eugene, Oregon, USA, June 10-14 2013.

[88] A. Haidar, R. Solca, M. Gates, S. Tomov, T. Schulthess, and J. Dongarra. Leading edge hybrid multi-GPU algorithms for generalized eigenproblems in electronic structure calculations. In *ICS'13: International Supercomputing Conference (submitted)*, Hamburg, Germany, June 2013.

[89] F. G. Gustavson. Recursion leads to automatic variable blocking for dense linear-algebra algorithms. *IBM J. Res. Dev.*, 41(6):737–756, November 1997.

[90] Elmar Peise and Paolo Bientinesi. Recursive algorithms for dense linear algebra: The ReLAPACK collection. Technical report, arXiv:1602.06763, 2016.

[91] Andreas Marek, Volker Blum, Rainer Johanni, Ville Havu, Bruno Lang, Thomas Auckenthaler, Alexander Heinecke, Hans-Joachim Bungartz, and Hermann Lederer. The ELPA library: scalable parallel eigenvalue solutions for electronic structure theory and computational science. *Journal of Physics: Condensed Matter*, 26:213201, 2014.

[92] Takeshi Fukaya and Toshiyuki Imamura. Performance evaluation of the Eigen Exa eigensolver on Oakleaf-FX: Tridiagonalization versus pentadiagonalization. In *Proceedings of the 2015 IEEE International Parallel and Distributed Processing Symposium Workshop*, pages 960–969. IEEE Computer Society, 2015.

[93] Françoise Tisseur and Jack Dongarra. A parallel divide and conquer algorithm for the symmetric eigenvalue problem on distributed memory architectures. *SIAM J. Sci. Comput.*, 20:2223–2236, 1999.

[94] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part I: Maintaining well-focused shifts and level 3 performance. *SIAM Journal on Matrix Analysis and Applications*, 23(4):929–947, 2002.

[95] Karen Braman, Ralph Byers, and Roy Mathias. The multishift QR algorithm. part II: Aggressive early deflation. *SIAM Journal on Matrix Analysis and Applications*, 23(4):948–973, 2002.

[96] Bo Kågström and Daniel Kressner. Multishift variants of the QZ algorithm with aggressive early deflation. *SIAM Journal on Matrix Analysis and Applications*, 29(1):199–227, 2007.

[97] W. M. Sid-Lakhdar, Aznavehand M. M., X. S. Li, and J. W. Demmel. Multitask and transfer learning for autotuning exascale applications. Technical report, 2019. Preprint: arXiv:1908.05792.

[98] J. Ansel, S. Kamil, K. Veeramachaneni, J. Ragan-Kelley, J. Bosboom, U. O'Reilly, and S. Amarsinghe. Opentuner: An extensible framework for program autotuning. In *Intern. Conf. Par. Arch. Compil. Techn.*, 2014.

[99] S. Falkner, A. Klein, and F. Hutter. BOHB: Robust and efficient hyperparameter optimization at scale. In *Proc. 35th Intern. Conf. Mach. Learn.*, pages 1437–1446, 2018. vol. 80 of Proc. of Machine Learning Research.

[100] C. Liao. Shared memory eigensolver, 2016. US Patent 9,262,799. https://www.google.com/patents/US9262799.

[101] David Trebotich, Mark F. Adams, Sergi Molins, Carl I. Steefel, and Chaopeng Shen. High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration. *Computing in Science & Engineering*, 16:22–31, 2014.

[102] GROMACS. Available: http://www.gromacs.org.

[103] Michael Commer and Gregory A. Newman. Three-dimensional controlled-source electromagnetic and magnetotelluric joint inversion. *Geophysical Journal International*, 178:1305–1316, 2009.

[104] R. G. Edwards and B. Joó. High-resolution simulation of pore-scale reactive transport processes associated with carbon sequestration. *Nucl. Phys B1*, 40:832, 2005. Proceedings of the 22nd International Symposium for Lattice Field Theory (Lattice2004).

[105] Jack Deslippe, Georgy Samsonidze, David A. Strubbe, Manish Jain, Marvin L. Cohen, and Steven G. Louie. BerkeleyGW: A massively parallel computer package for the calculation of the quasiparticle and optical properties of materials and nanostructures. *Computer Physics Communications*, 183:1269 – 1289, 2012.

[106] Robert J. Harrison, Gregory Beylkin, Florian A. Bischoff, Justus A. Calvin, George I. Fann, Jacob Fosso-Tande, Diego Galindo, Jeff R. Hammond, Rebecca Hartman-Baker, Judith C. Hill, Jun Jia, Jakob S. Kottmann, M.-J. Yvonne Ou, Laura E. Ratcliff, Matthew G. Reuter, Adam C. Richie-Halford, Nichols A. Romero, Hideo Sekino, William A. Shelton, Bryan E. Sundahl, W. Scott Thornton, Edward F. Valeev, Álvaro Vázquez-Mayagoitia, Nicholas Vence, and Yukina Yokoi. MADNESS: a multiresolution, adaptive numerical environment for scientific simulation. *CoRR*, abs/1507.01888, 2015.

[107] MFEM: Modular finite element methods. http://mfem.org.

[108] C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. Nektar++: An open-source spectral/element framework. *Computer Physics Communications*, 192:205–219, 2015.

[109] Y. Wang, G. M. Stocks, D. M. C. Nicholson, and W. A. Shelton. The first principles O[N] LSMS method and its applications to magnetic structure of alloys. *Computers & Mathematics with Applications*, 35:85–92, 1998.

[110] BLAS standard committee, 2016. www.netlib.org/utk/people/JackDongarra/WEB-PAGES/Batched-BLAS-2016.

[111] J. W. Demmel, P. Ahrens, and H. D. Nguyen. Efficient reproducible floating point summation and BLAS. EECS Tech Report EECS-2016-121.pdf, UC Berkeley, 2016.

[112] Introduction to conditional numerical reproducibility (CNR), 2012. https://software.intel.com/en-us/articles/introduction-to-the-conditional-numerical-reproducibility-cnr.

[113] IEEE standard for floating point arithmetic, IEEE Std 754-2019. https://doi.org/10.1109/IEEESTD.2019.8766229, 2019.

[114] Julie Langou, Julien Langou, Piotr Luszczek, Jakub Kurzak, Alfredo Buttari, and Jack Dongarra. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy. In *Proceedings of SC06*, Tampa, Florida, November 11-17 2006. See http://icl.cs.utk.edu/iter-ref.

[115] J. Kurzak and J. Dongarra. Implementation of the mixed-precision high performance LINPACK benchmark on the CELL processor. *University of Tennessee Computer Science Tech Report*, sep 2006.

[116] A. Buttari, J. Dongarra, J. Kurzak, P. Luszczek, and S. Tomov. Using mixed precision for sparse matrix computations to enhance the performance while achieving 64-bit accuracy. *ACM Transactions on Mathematical Software*, 34(4):17–22, July 2008.

[117] Marc Baboulin, Alfredo Buttari, Jack Dongarra, Jakub Kurzak, Julie Langou, Julien Langou, Piotr Luszczek, and Stanimire Tomov. Accelerating scientific computations with mixed precision algorithms. *Computer Physics Communications*, 180(12):2526–2533, 2009.

[118] S. Tomov, R. Nath, H. Ltaief, and J. Dongarra. Dense linear algebra solvers for multicore with GPU accelerators. In *Proc. of the IEEE IPDPS'10*, pages 1–8, Atlanta, GA, April 19-23 2010. IEEE Computer Society. DOI: 10.1109/IPDPSW.2010.5470941.

[119] A. Abdelfattah, H. Anzt, J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, S. Tomov, I. Yamazaki, and A. YarKhan. Linear algebra software for large-scale accelerated multicore computing. *Acta Numerica*, 25:1–160, 005 2016.

[120] Ichitaro Yamazaki, Stanimire Tomov, and Jack J. Dongarra. Mixed-precision Cholesky QR factorization and its case studies on multicore CPU with multiple GPUs. *SIAM J. Scientific Computing*, 37(3), 2015.

[121] Ichitaro Yamazaki, Stanimire Tomov, and Jack Dongarra. Stability and performance of various singular value QR implementations on multicore CPU with a GPU. *ACM Trans. Math. Softw.*, 43(2):10:1–10:18, September 2016.

[122] D. Bailey, J. Demmel, G. Henry, Y. Hida, J. Iskandar, W. Kahan, S. Kang, A. Kapur, X. Li, M. Martin, B. Thompson, T. Tung, and D. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Soft.*, 28(2):152–205, June 2002.

[123] Apache Spark, 2016. http://spark.apache.org.

[124] Alex Gittens, Aditya Devarakonda, Evan Racah, Michael F. Ringenburg, Lisa Gerhardt, Jey Kottalam, Jialin Liu, Kristyn J. Maschhoff, Shane Canon, Jatin Chhugani, Pramod Sharma, Jiyan Yang, James Demmel, Jim Harrell, Venkat Krishnamurthy, Michael W. Mahoney, and Prabhat. Matrix factorization at scale: a comparison of scientific data analytics in Spark and C+MPI using three case studies. *CoRR*, abs/1607.01335, 2016.

[125] Alex Gittens, Kai Rothauge, Michael W. Mahoney, Shusen Wang, Lisa Gerhardt, Prabhat, Jey Kottalam, Michael Ringenburg, and Kristyn Maschhoff. Alchemist: An Apache Spark <=> MPI interface. In *Cray User Group (CUG) Conference Proceedings*, 2018.

[126] A. Gittens, K. Rothauge, S. Wang, M. W. Mahoney, L. Gerhardt, Prabhat, J. Kottalam, M. Ringenburg, and K. Maschhoff. Accelerating large-scale data analysis by offloading to high-performance computing libraries using Alchemist. Technical report, 2018. Preprint: arXiv:1805.11800.

[127] K. Rothauge, H. Ayyalasomayajula, K. J. Maschhoff, M. Ringenburg, and M. W. Mahoney. Running Alchemist on Cray XC and CS series supercomputers: Dask and PySpark interfaces, deployment options, and data transfer times. Technical report, 2019. Preprint: arXiv:1910.01354.

[128] Mark Gates, Piotr Luszczek, Ahmad Abdelfattah, Jakub Kurzak, Jack Dongarra, Konstantin Arturov, Cris Cecka, and Chip Freitag. C++ API for BLAS and LAPACK. SLATE Working Note 2, ICL-UT-17-03, Innovative Computing Laboratory, University of Tennessee, 06 2017. Revision 02-21-2018.

[129] Stephen Cass. The top programming languages 2019. *IEEE Spectrum*, 2019. https://spectrum.ieee.org/computing/software/the-top-programming-languages-2019.

[130] ICL style guide. Available: https://bitbucket.org/icl/guides/wiki/iclstyleguide.

[131] Ed Anderson and Z. Bai and C. Bischof and Susan L. Blackford and James W. Demmel and Jack J. Dongarra and J. Du Croz and A. Greenbaum and Sven Hammarling and A. McKenney and Danny C. Sorensen. *LAPACK User's Guide*. Society for Industrial and Applied Mathematics, Philadelphia, 3rd edition, 1999.

[132] Gaël Guennebaud, Benoît Jacob, Philip Avery, Abraham Bachrach, Sebastien Barthelemy, Carlos Becker, David Benjamin, Cyrille Berger, Armin Berres, Jose Luis Blanco, Mark Borgerding, Romain Bossart, Kolja Brix, Gauthier Brun, Philipp Büttgenbach, Thomas Capricelli, Nicolas Carre, Jean Ceccato, Vladimir Chalupecky, Benjamin Chrétien, Andrew Coles, Marton Danoczy, Jeff Dean, Georg Drenkhahn, Christian Ehrlicher, Martinho Fernandes, Daniel Gomez Ferro, Rohit Garg, Mathieu Gautier, Anton Gladky, Stuart Glaser, Marc Glisse, Frederic Gosselin, Christoph Grüninger, Gaël Guennebaud, Philippe Hamelin, Marcus D. Hanwell, David Harmon, Chen-Pang He, Hauke Heibel, Christoph Hertzberg, Pavel Holoborodko, Tim Holy, Trevor Irons, Benoît Jacob, Bram de Jong, Kibeom Kim, Moritz Klammler, Claas Köhler, Alexey Korepanov, Igor Krivenko, Marijn Kruisselbrink, Abhijit Kundu, Moritz Lenz, Bo Li, Sebastian Lipponer, Daniel Lowenberg, David J. Luitz, Naumov Maks, Angelos Mantzaflaris, D J Marcin, Konstantinos A. Margaritis, Roger Martin, Ricard Marxer, Vincenzo Di Massa, Christian Mayer, Frank Meier-Dörnberg, Keir Mierle, Laurent Montel, Eamon Nerbonne, Alexander Neundorf, Jason Newton, Jitse Niesen, Desire Nuentsa, Jan Oberländer, Jos van den Oever, Michael Olbrich, Simon Pilgrim, Bjorn Piltz, Benjamin Piwowarski, Zach Ploskey, Giacomo Po, Sergey Popov, Manoj Rajagopalan, Stjepan Rajko, Jure Repinc, Kenneth Frank Riddile, Richard Roberts, Adolfo Rodriguez, Peter Román, Oliver Ruepp, Radu Bogdan Rusu, Guillaume Saupin, Olivier Saut, Benjamin Schindler, Michael Schmidt, Dennis Schridde, Jakob Schwendner, Christian Seiler, Martin Senst, Sameer Sheorey, Andy Somerville, Alex Stapleton, Benoit Steiner, Sven Strothoff, Leszek Swirski, Adam Szalkowski, Silvio Traversaro, Piotr Trojanek, Anthony Truchet, Adolfo Rodriguez Tsourouksdissian, James Richard Tyrer, Rhys Ulerich, Henry de Valence, Ingmar Vanhassel, Michiel Van Dyck, Scott Wheeler, Freddie Witherden, Urs Wolfer, Manuel Yguel, Pierre Zoppitelli, and Jonas Adler. Eigen v3. http://eigen.tuxfamily.org, 2011.

[133] ScaLAPACK user's forum. Available: http://icl.cs.utk.edu/lapack-forum.

[134] TRAVIS. Available: https://travis-ci.org.

[135] The BSD 3-clause license. Available: https://opensource.org/licenses/BSD-3-Clause.